online Übungssystem & online Prüfungssystem

Handbuch für Aufgabenautoren II

Fortgeschrittene Aufgabenerstellung und weiterführende Themen

Autor:
Immo Schulz-Gerlach, ZMI, FeU-Softwaretechnik

Version:
3.11 — 07. Dezember 2023

FernUniversität in Hagen

Inhaltsverzeichnis

> ***Im Folgenden setzen wir voraus, dass Sie bereits mit Teil I des Handbuchs <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html> vertraut sind und sich explizit für die fortgeschrittene Aufgabenerstellung entschieden haben**, egal ob zur Neuerstellung einer kompletten Aufgabe oder zur Nachbearbeitung einer assistentengestützt erzeugten Aufgabe, wenn Ihnen gewisse vom Editor generierte Aspekte der Aufgabe nicht zusagen.*

Letzte Änderungen

in Version 3.11

- Neue Variablen `$QR` und `$QR(https://...)` zum Einfügen von QR-Codes.

in Version 3.10

- Neue Version 2.2 des Multiple-Choice-Bewerters 2.0.
- Die ausführlichen Erläuterungen und Definitionen zu den verschiedenen Multiple-Choice-Bewertern wurden aus diesem Handbuch entfernt, um es etwas kürzer und übersichtlicher zu halten und Redundanzen zu entfernen, denn sie werden ohnehin auch noch in einem separaten Handbuch »Bewertung von Mehrfachauswahlaufgaben« <https://online-uebungssystem.fernuni-hagen.de/download/XausNBewertung/XausNBewertung.html> bereitgestellt.

in Version 3.9

- Abschnitt zur Konfiguration des internen Aufgabenbewerters zu Begriffs-Fragen ergänzt um die neue Möglichkeit, Synonymen abweichende Teilpunktzahlen zuzuordnen.
- Im selben Abschnitt wurde der Text zum Escaping von Sonderzeichen (Pipe und Backslash) sowie zu Leerzeichen in der Wortliste präzisiert.
- Dokumentation des Verfahrens zur Speicherung einer Teilaufgabenselektion bei Randomisierung. Das Verfahren wird vor allem nun vom Aufgabenerstellungsassitenten für automatisch bewertete Aufgaben bei Randomisierung mit Zufallsreihenfolge eingesetzt, um eine individuelle Sortierung der Autokorrekturen analog zur persönlichen Zufallsreihenfolge der Fragen in der Aufgabenseite zu ermöglichen. Es steht aber auch in der fortgeschrittenen Aufgabenerstellung allgemein zur Verfügung, vor allem mit Blick auf die Entwicklung eigener Korrekturmodule.

in Version 3.8

Als modernerer Ersatz für die alte JavaScript-basierte Formular-Eingabevalidierung wurde eine auf HTML-5-Inputtypes aufbauende neuere Lösung eingeführt, dokumentiert in separatem Handbuch <https://online-uebungssystem.fernuni-hagen.de/download/JSFormValidation/FormularValidierung.html> .

- Dazu wurden insbesondere nun auch Codebeispiele in diesem Handbuch, die noch auf die alte Validierung ausgelegt waren, aktualisiert/umgestellt.
- Auch gab es im Rahmen des HTML-Numer-Input-Supports z.B. Erweiterungen des internen Aufgabenbewerters, entsprechend wurde hier der Abschnitt zur Konfiguration der internen Module ergänzt.

in Version 3.7

- Das Online-Übungssystem bindet ab sofort einen Formeleditor für die Eingabe von mathematischen Formeln ein. (Bisher mussten diese immer von Hand als LaTeX-Code eingegeben werden, der Formeleditor soll diese Erstellung von LaTeX-Code erleichtern.) Auch wenn dieser Formeleditor wohl in erster Linie von Aufgabenautoren genutzt werden sollte, kann er auch Studierenden zur Formeleingabe genutzt werden:
  - Insbesondere können Eingabefelder speziell zur Eingabe einer einzelnen Formel in einem Aufgabenformular eingebunden werden. Um diese mit dem Formeleditor auszustatten, gibt es nun eine neue Syntax (`<input type="latex">`).
- Falls Sie als Aufgabenautor Eingabefelder der Form `<input type="number">` verwenden wollen, gilt, dass der Browser Zahleneingaben darin immer in eine „technische" Form normalisiert, insbesondere immer einen Punkt als Dezimaltrenner verwendet. Um solche Einsendungen in Quittungs- und Korrekturseiten wieder als lokalisierte Zahlen mit Komma als Dezimaltrenner anzuzeigen, gibt es nun ein neues Variablenformat (`$FeldA1DECIMAL`).
- **In Version 3.7.1:** Es wird nun empfohlen, den `action`-URL im `form`-Tag von Aufgabenformularen mit einem Slash zu terminieren.

in Version 3.6

- Der interne Aufgabenbewerter unterstützt für die Bewertung von Texteingaben (Fragetypen Begriffe und Begriffsfolge / Lückentext) ab sofort die Festlegung von Musterlösungs-Begriffsmustern über reguläre Ausdrücke.
  - Der Abschnitt zur Bewerterkonfiguration wurde diesbezüglich ergänzt.
  - Außerdem wurde auch die Antworthäufigkeits-Statistik-Funktion analog erweitert, um ebenfalls Eingaben, die auf eine als regulärer Ausdruck angegebene Musterlösung passen, entsprechend erkennen, zusammenfassen und als „korrekte Antwort" klassifizieren zu können. Der entsprechende Handbuchabschnitt zur Konfiguration der Statistik wurde dazu entsprechend erweitert.

in Version 3.5

- Ergänzungen / Anpassungen in Bezug auf das neue (in den Kursparametern aktivierbare) Feature, Punktzahlen mit begrenzter Anzahl an Nachkommastellen zu erlauben.

Fortgeschrittene Aufgabenerstellung

In der fortgeschrittenen Aufgabenerstellung haben Sie die größtmögliche Freiheit der Aufgabengestaltung, benötigen jedoch ungleich mehr Kenntnisse – insbesondere in der Webseiten-Beschreibungssprache HTML, ggf. darüber hinaus Kenntnisse in CSS oder JavaScript. Falls Sie sogar eigene Vorkorrektur- oder Bewertermodule <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#moduleoverview> schreiben wollen, werden entsprechende Programmierkenntnisse in der für die Realisierung gewählten Sprache benötigt sowie ggf. (falls Sie nicht das von uns bereitgestellte Java-Framework zur Korrekturmodul-Entwicklung verwenden) grundlegende Kenntnisse im Programmieren von SOAP-Webservices.

Eine Aufgabeneinrichtung im Online-Übungssystem besteht in erster Linie aus einer Menge von HTML-Dateien, den im Grundlagen-Kapitel bereits angesprochenen Standard-Kursressourcen <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#kr> , die im

nachfolgenden Abschnitt genauer vorgestellt werden sollen. Darüber hinaus sind noch ein paar Einstellungen wie insbesondere der Aufgabenname, erreichbare Punktzahl, Korrekturart und ggf. Zuordnung von (Vor-)Korrekturmodulen samt deren Konfiguration vorzunehmen. Darauf wird anschließend im Rahmen der Beschreibung des Aufgabenerstellungsprozesses eingegangen.

Die Standard-Kursressourcen zu einer Aufgabe

Zur `y`-ten Aufgabe im `x`-ten Aufgabenheft gehören die folgenden Standard-Kursressourcen <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#kr> :

Dateiname	Inhalt	Pflicht?
`aufgabex.y.html`	Aufgabenstellung und -Formular	Ja
`quittungx.y.a.html`	Einsendebestätigungsseite für Teilaufgabe `A`	Ja (eine Datei pro Teilaufgabe)
`korrekturx.y.html`	Korrekturseitenvorlage	Ja
`musterx.y.html`	Musterlösungstext zur Aufgabe	Nein, aber für handbewertete Aufgaben empfohlen
`hinweisex.y.html`	Hinweise für Korrekturkräfte	Nein
`antwortstatistikx.y.xml`	Einrichtung einer Antworthäufigkeits-Statistik	Nein

Der jeweilige Einsatzzweck der einzelnen Dateien sei an folgendem *Szenario* verdeutlicht:

- Ein Student loggt sich ein, um eine Aufgabe zu bearbeiten. Er wählt in der Aufgabenübersicht die zu bearbeitende Aufgabe aus, z.B. Aufgabe Nr. 3 im Aufgabenheft 1.
- Die Webseite, die ihm nun angezeigt wird, ist im Wesentlichen `aufgabe1.3.html`.
  - Diese enthält zum Ersten die Aufgabenstellung (oder einen Link darauf, falls diese extern angeboten wird).
  - Zum Zweiten enthält die Seite das Aufgabenformular, also Eingabemöglichkeiten für die Studenten. Dieses kann z.B. aus Textfeldern, Checkboxen oder Dateiupload-Möglichkeiten bestehen.
  - Falls der Student die Aufgabe schon einmal bearbeitet hatte, werden ihm seine letzten Eingaben in diesem Formular wieder angezeigt, so dass er sie einsehen und ggf. nachbearbeiten kann.
- Der Student gibt dort nun seine Lösung ein und sendet die Eingaben ein (vor Einsendeschluss[1]). Falls die Aufgabe in („technische") Teilaufgaben <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#ta> unterteilt ist, kann er nur Eingaben zu einer dieser Teilaufgaben auf einmal einsenden.
  - Als Reaktion auf jede Einsendung wird dem Studenten eine Bestätigungsseite („Quittung") angezeigt. Diese bestätigt den Eingang seiner Einsendung und zeigt im Normalfall die eingegangenen und gespeicherten Daten nochmals an, damit der Student sich überzeugen kann, dass seine Eingaben auch wirklich fehlerfrei eingegangen sind.
    - Die Quittung wird dynamisch erzeugt, indem eine entsprechende Quittungs-Seitenvorlage zu der bedienten Teilaufgabe mit den konkreten Einsendungen des Studenten gefüllt wird.
    - Hat der Student zu Teilaufgabe `A` eingesendet (oder ist die Aufgabe nicht in mehrere getrennt einzusendende Teilaufgaben untergliedert), so wird z.B. die Ressource `quittung1.3.a.html` als zu füllende Seitenvorlage verwendet, bei

Einsendungen zu Teilaufgabe `B` entsprechend `quittung1.3.b.html` etc.

- Falls ein Vorkorrekturmodul <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#moduleoverview> verwendet wird, so wird dieses direkt nach Einsendung ausgeführt, und die erzeugte Vorkorrektur wird in aller Regel ebenfalls mit in die Quittung aufgenommen, um dem Studenten ein Sofort-Feedback zu geben, auf Basis dessen er seine Einsendung nachbearbeiten kann.

- Obige Schritte wiederholen sich für ggf. mehrere Teilaufgaben der Aufgabe oder weitere Aufgaben des Aufgabenhefts.
- Irgendwann findet das so genannte *Heft-Schließen* statt:
  - Falls in den Kursparametern aktiviert (oder ein Heft-Schließen-Knopf <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#heftschliessen> in eine Aufgabenseite eingefügt wurde, siehe unten), kann ein Student nach Bearbeitung aller Aufgaben schon vor Einsendeschluss sein »Heft schließen« und damit seine Lösungen vorzeitig abgeben. Damit nimmt er sich selbst die Möglichkeit, die Einsendungen nochmals zu überarbeiten, gibt aber der Kursbetreuung die Möglichkeit, schon vor Einsendeschluss mit der Korrektur zu beginnen.
  - Schließt der Student das Heft nicht selbst (weil er es nicht möchte oder die Funktion abgeschaltet ist), so schließt entweder ein Kursbetreuer oder ein automatischer Job des Übungssystems das Heft nach Einsendeschluss. (Man könnte dies auch als „Einsammeln der noch nicht abgegebenen Hefte“ beschreiben.)
  - In jedem Fall wird im Rahmen des Heft-Schließens eine *Korrekturseite* für den Studenten und die von ihm bearbeitete Aufgabe 1.3 erzeugt und gespeichert.
  - Dies erfolgt analog zur Erstellung der Quittung: Es wird eine Seitenvorlage (hier `korrektur1.3.html`) geladen und mit den Daten gefüllt, die der Student zur Aufgabe eingesendet hat.
    - Hinweis: Die Eingaben des Studenten existieren nun doppelt: Einmal in Reinform in der Datenbank, einmal in der so erzeugten Korrekturseite. Letztere lässt sich jederzeit wieder neu erzeugen, indem der Vorgang wiederholt wird, also erneut die Seitenvorlage mit den Eingaben aus der Datenbank gefüllt wird (erneutes Heft-Schließen). Das kann nötig werden, falls z.B. ein Korrektor versehentlich studentische Eingaben aus der Korrekturseite gelöscht hat (sofern er dazu die Möglichkeit hat) oder sich im Nachhinein herausstellt, dass die Seitenvorlage nicht in Ordnung war, z.B. nicht alle Eingaben zu allen Eingabefeldern der Aufgabe auflistete.
  - Falls ein Korrektur-/Bewertermodul <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#moduleoverview> verwendet wird, also eine automatische Korrektur und Bewertung <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#ab> stattfinden soll, so wird dieses direkt beim Heft-Schließen ausgeführt und die von ihm erzeugte Autokorrektur wird mit in die erzeugte *Korrekturseite* eingefügt.
  - Falls ein Vorkorrekturmodul <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#moduleoverview> verwendet wird, so wird auch dessen Ausgabe (die direkt bei Einsendung erzeugt wurde, s.o.) im Standardfall mit in die *Korrekturseite* eingefügt. So kann einerseits der Student später diese Vorkorrektur immer im Rahmen der Korrektur mit einsehen (sieht also sowohl die automatische Vorkorrektur als auch die manuelle Nachkorrektur und Bewertung in derselben Korrekturseite). Andererseits erhält so auch der Korrektor Einblick in die Vorkorrektur und kann diese bei seiner Korrektur berücksichtigen.

- Nach Einsendeschluss des Aufgabenhefts loggt sich der Student erneut ein und ruft die Seite »Ergebnisse und Musterlösungen« auf. Sofern eine `muster1.3.html` vom Aufgabenautor erzeugt wurde, kann der Student diese normalerweise direkt ab Einsendeschluss einsehen. (Es gibt auch Optionen in den Kursparametern, die Musterlösungen nicht vor oder nur zusammen mit Korrekturen[2] freizugeben.)
- Angenommen, die Aufgabe werde manuell bewertet <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#hb> , dann muss der Student noch bis zur Fertigstellung der Korrektur warten, er sieht hier nur die Anzeige »im Korrekturprozess«.[3]
- Der Korrektor, dem die Einsendung zur Korrektur zugewiesen wurde, loggt sich ein, um sie zu korrigieren.
    - Dem Korrektor wird nun die erzeugte *Korrekturseite* (s.o.) angezeigt. Er sieht also die Einsendungen des Studenten, die er korrigieren und bewerten soll, sowie ggf. das Ergebnis einer automatischen Vorkorrektur.
    - Bei In-Browser-Korrektur <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#ibk> enthält diese Korrekturseite auch ein Eingabeformular für den Korrektor, in das er seine Kommentare eintragen kann. Bei herkömmlicher HTML-Korrektur <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#htmlk> fehlen entsprechende Formularfelder in der HTML-Datei, aber der Korrektor erhält die Möglichkeit, die gesamte HTML-Datei (Kopie der `korrektur1.3.html` mit darin eingefügten studentischen Eingaben) in einem HTML-Editor zu bearbeiten.
    - In jedem Fall speichert der Korrektor seine Korrektur. Dabei wird eine *neue Version der Korrekturseite* gespeichert, die neben der Einsendung des Studenten nun auch noch die Kommentare des Korrektors enthält. Die bisherige, unkorrigierte Version wird dabei überschrieben.
    - Sofern vorhanden, kann der Korrektor während der Korrektur auch eine Korrekturhinweisseite (`hinweise1.3.html`) einsehen. Hierin hinterlegen Aufgabenautoren z.B. Informationen wie ein Schema zur Punktevergabe, Hinweise auf typische Fehler und wie diese zu korrigieren sind oder Ähnliches.
- Nachdem der Korrektor die Korrektur als fertig markiert und ein Kursbetreuer sie freigegeben hat, erhält der Student automatisch eine E-Mail, dass Korrekturen für ihn freigegeben wurden. Er loggt sich erneut ein und kann die fertige Korrekturseite nun einsehen.
- Bei bestimmten Aufgabentypen, in denen nur wenige mögliche Antworten (ggf. in Kombination) gegeben werden können – wie z.B. bei Multiple-Choice-Aufgaben –, könnten sich Kursbetreuer nach Einsendeschluss für eine Statistik interessieren, welche Antworten wie häufig gegeben wurden. Bei Multiple Choice z.B. kann es interessant sein, zu wissen, welche korrekten Antworten von fast allen Teilnehmern gegeben wurden und welche nicht, bzw. welche Antworten zwar falsch waren, aber dennoch besonders häufig fielen.
    - Eine solche Statistik kann nicht generisch für beliebige Eingaben zu beliebigen Aufgaben erzeugt werden, sondern benötigt gewisse Kenntnisse über die Art der Aufgabe, z.B. ob es sich um eine X-aus-N-Frage handelt, so dass Einsendungen aus einer Aufzählung von Antwortalternativen in der Art „A, C, D“ bestehen und folglich nach Komma in Teilantworten zerlegt werden sollen, oder ob es sich um eine Zahlenfrage handelt, deren Antworten, gerundet auf eine bestimmte Nachkommastellenzahl, gezählt werden sollen – und welche Antworten überhaupt korrekt oder falsch sind.
    - Zu einer Aufgabe können mehrere Statistiken erzeugt werden, z.B. weil die Aufgabe aus mehreren einzelnen Fragen besteht. Es können sogar zu einer Frage mehr als eine Statistik erzeugt werden, z.B. bei Multiple-Choice-Fragen eine Statistik zur Häufigkeit der Nennung der einzelnen Antwortalternativen und eine zweite Statistik zur Häufigkeit der

gegebenen Antwortkombinationen.

- Zu jeder Aufgabe, für die Antworthäufigkeits-Statistiken angeboten werden sollen, ist daher eine aufgabenspezifische Einrichtung vorzunehmen, und zwar in Form einer XML-Ressource `antwortstatistikx.y.xml`. Der Abschnitt Antworthäufigkeits-Statistik zu einer Aufgabe aktivieren/konfigurieren gegen Ende dieses Kapitels geht genauer auf den Aufbau dieser Ressource ein.

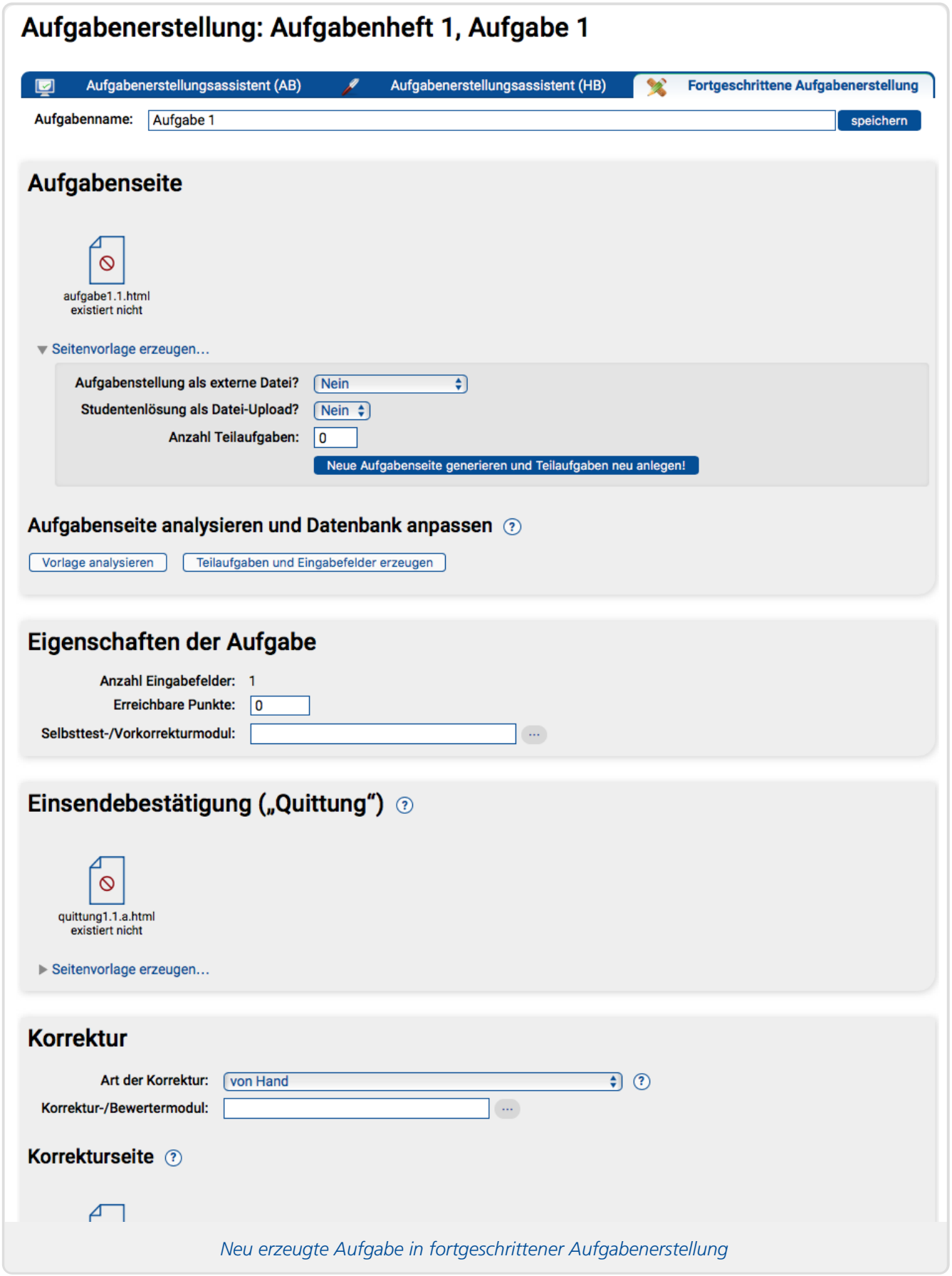
Erstellung einer neuen Aufgabe

Die Aufgabenerstellung besteht nun also in erster Linie in der Erstellung der oben genannten Kursressourcen und abschließend oder „zwischendurch" der Speicherung der Einstellungen zur Aufgabe. Es gibt verschiedene Möglichkeiten, bei der Erstellung neuer Aufgaben vorzugehen, insbesondere:

1. Ein Extrem: Offline-Erstellung der Standard-Kursressourcen-Dateien mit einem beliebigen Texteditor Ihrer Wahl und Upload einer ZIP-Datei mit allen Kursressourcen zu einer oder mehreren Aufgaben ins Online-Übungssystem. Das Online-Übungssystem erkennt Standard-Kursressourcen im Upload und erzeugt bei Bedarf die entsprechenden Datenbankeinträge für das Aufgabenheft (falls es noch nicht existiert) und die Aufgabe[4], d.h. es ist nicht unbedingt nötig, vor dem Upload das Aufgabenheft und in der Aufgabenheft-Verwaltung erst eine neue Aufgabe anzulegen. Nach dem Upload ist dann die jeweilige erzeugte Aufgabe aber noch nachzubearbeiten, um alle benötigten Einstellungen wie Name, Punkte, Korrekturmodus etc. einzustellen. Auch die (im Folgenden noch beschriebene) Erzeugung der Teilaufgaben und Eingabefelder muss noch angestoßen werden.
2. Ein anderes Extrem: Sie erstellen die Aufgabenressourcen nicht von Grund auf selbst, sondern verwenden zunächst einen der in Teil I <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html> beschriebenen *Aufgabenerstellungsassistenten*. Auch falls Sie die Aufgaben damit nicht exakt so erstellen können, wie Sie sie endgültig haben wollen, können Sie damit eine erste Version erstellen und diese anschließend in der fortgeschrittenen Aufgabenerstellung nachbearbeiten – also z.B. die generierten HTML-Ressourcen abändern, nachträglich ein Vorkorrekturmodul hinzufügen etc.
3. Ein weiterer möglicher Weg liegt dazwischen: Sie erstellen die Aufgabenressourcen direkt selbst, aber „online". Sie erzeugen ggf. ein neues Aufgabenheft, dazu eine Aufgabe, konfigurieren diese und lassen sich dabei Kursressourcen generieren, die bereits einen gewissen Rahmen enthalten, aber noch um die eigentlichen Inhalte ergänzt werden müssen. Das Bearbeiten kann ebenfalls online erfolgen, das empfiehlt sich aber im Wesentlichen später für kleinere Korrekturen. Umfangreiche Arbeiten an HTML-Dateien sind im Zweifel doch besser mit einem lokalen Editor-Programm vorzunehmen – wozu die erzeugten Ressourcen heruntergeladen, bearbeitet und wieder hochgeladen werden können.

Die nachfolgenden Abschnitte beschreiben die *dritte* Möglichkeit genauer und erklären dabei die verschiedenen Konfigurationsmaßnahmen und den Aufbau der Standard-Kursressourcen. Die Ausführungen lassen sich aber auch auf die ersten beiden Varianten übertragen und sind natürlich ebenfalls beim Bearbeiten existierender Aufgaben als Referenz nützlich.

Wir gehen also im Rest dieses Kapitels davon aus, dass – wie in Teil I <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html> unter „Vorbereitungen" beschrieben – ein Aufgabenheft angelegt, diesem eine neue Aufgabe hinzugefügt und dann an Stelle eines der Aufgabenerstellungsassistenten direkt die fortgeschrittene Aufgabenerstellung ausgewählt wurde. Sie sehen dann eine Seite wie die Folgende:



Neu erzeugte Aufgabe in fortgeschrittener Aufgabenerstellung

Als Aufgabenname wird `Aufgabe y` vorgeschlagen, wobei `y` für die Nummer der Aufgabe (innerhalb ihres Aufgabenhefts) steht. Wollen Sie Ihre Aufgaben anders benennen (Siehe Abschnitt zur Aufgabenbenennung in Teil I <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html>), können Sie oben einen anderen Namen eintragen und speichern.

Aufgabenformular erstellen

Nach dem Erstellen und Benennen der Aufgabe ist der wichtigste Schritt die Erstellung der Aufgabenseite (`aufgabex.y.html`). Sie könnten zwar eine solche HTML-Datei komplett extern erstellen und hochladen, das Übungssystem bietet Ihnen hier aber die Generierung einer Seitenvorlage an, die Sie anschließend nur noch mit Inhalten füllen müssen.

Generieren und Bearbeiten einer Seitenvorlage

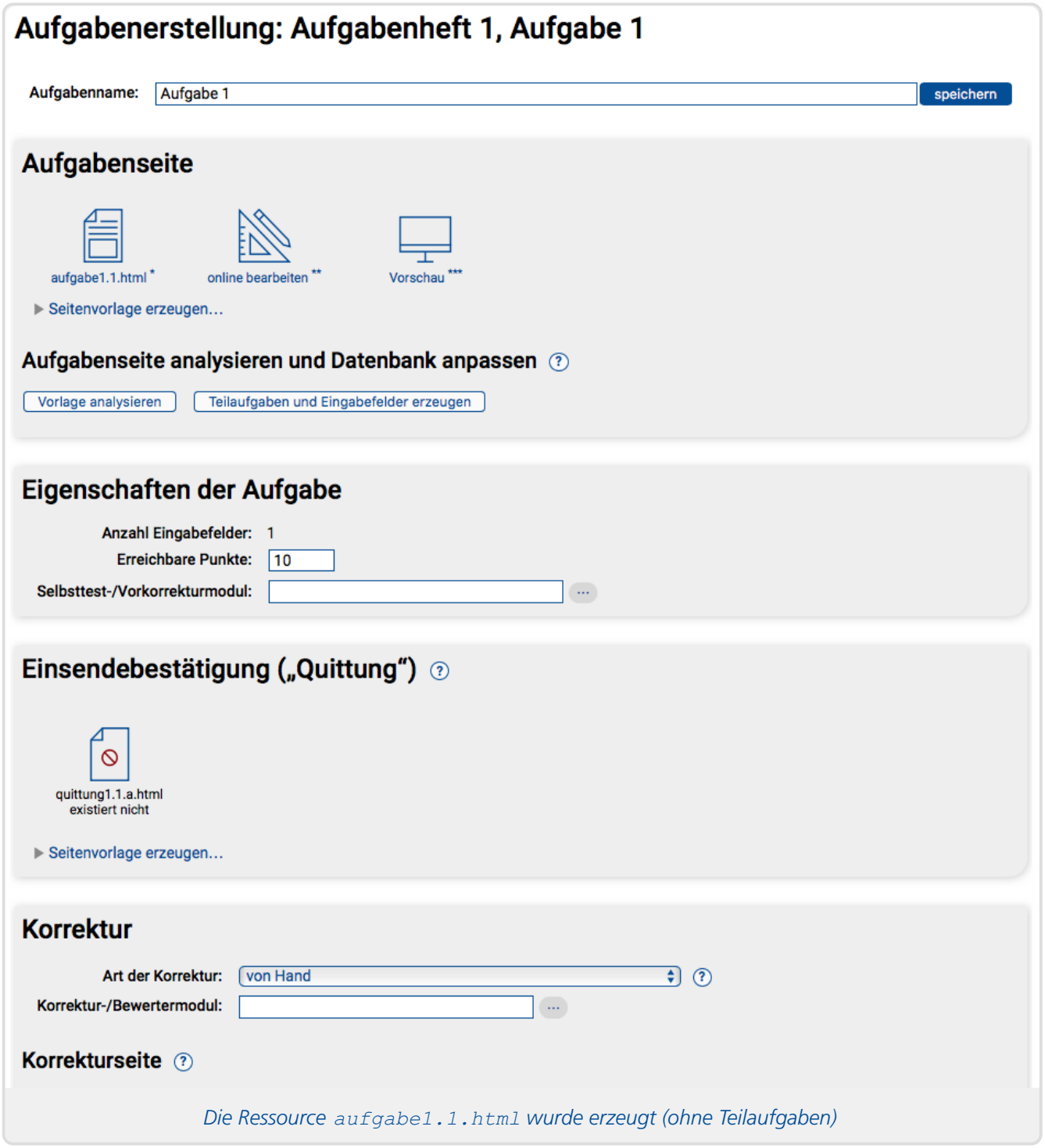
Stellen Sie dazu zunächst unter »*Aufgabenstellung als externe Datei?*« ein, ob Sie vorhaben, die Aufgabenstellung direkt in die HTML-Seite zu schreiben oder ob Sie lieber eine Datei verlinken möchten. Im letzteren Fall wird bereits ein solcher Link in die zu generierende HTML-Datei eingefügt, aber natürlich muss die entsprechende Datei dann noch erstellt und als Kursressource hochgeladen werden.

Falls Sie unter »*Studentenlösung als Datei-Upload?*« `Nein` auswählen, wird das Aufgabenformular für reine Texteingaben optimiert. Falls in der Aufgabe mindestens ein Datei-Upload möglich sein soll, sollten Sie hier `Ja` auswählen. `Ja` bewirkt die Erzeugung eines Formulars für *Multipart*-Submit, welches Datei-Uploads unterstützt, aber auch normalen Text-Inhalt einsenden kann (siehe Abschnitt zum `form`-Tag).

Unter »*Anzahl Teilaufgaben*« können Sie vorgeben, in wie viele „technische“ Teilaufgaben <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#ta> Sie Ihre Aufgabe zerlegen wollen (siehe obige Ausführungen zu diesem Thema). Für keine Zerlegung in mehrere Teilaufgaben können Sie `0` eintragen – technisch existiert dann genau eine Teilaufgabe `A`, die allein die gesamte Aufgabe bildet.

Per »*Neue Aufgabenseite generieren und Teilaufgaben neu anlegen!*« erzeugen Sie die Seitenvorlage.

Im Ergebnis sieht der Aufgabeneditor dann z.B. wie folgt aus:



Die Ressource `aufgabe1.1.html` wurde erzeugt (ohne Teilaufgaben)

Man sieht, dass nun eine `aufgabe1.1.html`-Ressource existiert. Die beiden nachfolgenden Abschnitte »Eigenschaften der Aufgabe« und »Einsendebestätigung („Quittung“)« werden in dieser Form nur angezeigt, falls die Aufgabe nicht aus mehreren Teilaufgaben besteht. Andernfalls gibt es dort einen Block pro Teilaufgabe, der jeweils dieselben Inhalte (teilaufgabenspezifische Eigenschaften und Quittung) aufweist.

Das *erste Icon*, das mit »aufgabe1.1.html« beschriftet ist, öffnet die Aufgabenressource in ihrer „Rohform“ direkt im Browser. Mit einer Browser-Editor-Suite wie Mozilla SeaMonkey <https://www.seamonkey-project.org> ist damit auch eine direkte Bearbeitung der Ressource möglich: SeaMonkey kann die Ressource vom Browser direkt in den Editor („Composer“) laden und der kann seine Änderungen (per HTTP PUT) direkt wieder ans Übungssystem senden. Das ist aber nicht mehr die empfohlene Art der Aufgabenbearbeitung.

Über das *zweite Icon* (»online bearbeiten«) können Sie die Kursressource direkt in einem In-Browser-

Editor öffnen. Dabei haben Sie die Wahl zwischen einem reinen HTML-Quelltext-Editor und einem WYSIWYG-Editor. Letzterer ist allerdings zumindest für die Aufgabenformular-Ressource nur bedingt geeignet, da er keine HTML-Formulare bearbeiten kann. Er eignet sich z.B. für kleinere Korrekturen an einem schon existierenden Aufgabentext, aber nicht zur kompletten Erstellung des Aufgabenformulars. Der HTML-Quelltext-Editor dagegen ist durchaus geeignet, das Aufgabenformular zu erstellen, bietet aber mangels Features wie Syntax-Highlighting, Auto-Completion, Auto-Tabbing oder Markup-Validation doch deutlich weniger Komfort als ein dedizierter Offline-HTML-Editor.

Empfohlen für umfangreichere Arbeiten am HTML-Code der Aufgabenseite (insbesondere die Neuerstellung komplexer Aufgabenformulare) wird daher der Download des HTMLs und eine Offline-Bearbeitung mit einem dedizierten Editor. Der Download ist wiederum über den ersten Link »aufgabe1.1.html« möglich (z.B. durch Rechtsklick auf den Link / das Icon und Auswahl des browserabhängigen Download-Menüpunkts). Alternativ kann eine ZIP-Datei mit *allen* Kursressourcen in der Kursressourcen-Verwaltung heruntergeladen werden. Die fertige Datei laden Sie über die »Kursressource-Upload«-Funktion wieder hoch.

Der HTML-Quelltext einer so generierten Ressource kann z.B. wie folgt aussehen (hier: keine Teilaufgaben, keine Dateieinsendungen):

```
<html>
<head>
  <title>&Uuml;bungen zu "$Kursname", Aufgabe
$AufgabenheftNr.$AufgabenNr</title>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
  <div style="background-color:#ccc; padding: 0.2em">
    Kurs $KursNr, &bdquo;$Kursname&ldquo;, $Versionsname<br>
    Aufgabenheft $AufgabenheftNr, Aufgabe $AufgabenNr
  </div>
  <h1>$Aufgabenname $MaximalPunkteUeberschrift</h1>
  $EMBED
  <form method="POST"

action="$WebAssignServer/$Veranstaltername/Einsendung/$KursNr/$VersionsNr/$Aufg
abenheftNr/$AufgabenNr/">
    <h2>Aufgabe 1 )</h2>
    <p>Hier die Aufgabenstellung einf&uuml;gen</p>
    <h2>L&ouml;sung 1 )</h2>
    <textarea type="text" name="FeldA1" rows="35" cols="75"> </textarea><br>
    <input type="submit" name="einsendenA"  value="Lösung $AufgabenNr
einsenden">
  </form>
  $/EMBED
</body>
</html>
```

An den generierten Überschriften etc. können Sie feilen, Ihren Aufgabentext z.B. im dafür vorgesehenen Absatz einfügen und das derzeit aus genau einer Textarea bestehende Eingabeformular beliebig anpassen, indem Sie die Textarea ersetzen oder weitere Formularelemente hinzufügen. Das ist weitgehend „gewöhnliche“ HTML-Bearbeitung. Entsprechende Kenntnisse in HTML, CSS und ggf. (bei Bedarf) JavaScript setzen wir bei der fortgeschrittenen Aufgabenerstellung voraus (s.o.).

In den folgenden Abschnitten werden lediglich die Übungssystem-spezifischen Besonderheiten dokumentiert.

Insbesondere nach der Bearbeitung der `aufgabe1.1.html`-Datei ist das *dritte Icon* (»Vorschau«) nützlich. Während das erste Icon (»aufgabe1.1.html«) Ihnen die „Rohfassung" der Seite zur Ansicht und Bearbeitung präsentiert, sehen Sie unter »Vorschau« die „Live-Fassung" der Seite (ähnlich wie ein Student): Alle Variablen (s.u.) werden durch konkrete Werte ersetzt, und in der Vorschau sind Einsendungen zum Testen der Aufgabe möglich (sofern ein Teststudent namens `7777777` zum Kurs angemeldet ist, und sofern bereits eine Quittungsseite erstellt wurde, s.u.).

Encoding

Sie arbeiten hier mit Textdateien, die Sie auch extern bearbeiten und wieder hochladen können, und Textdateien können auf verschiedene Weise kodiert sein. Das Online-Übungssystem ist mittlerweile relativ tolerant und versucht, das Encoding Ihrer Seiten automatisch zu erkennen. Auf der sicheren Seite sind Sie, wenn Sie gebräuchliche Charsets wie `UTF-8`, `ISO-8859-1`, `Windows-1252` oder ähnliche verwenden und deren Gebrauch im HTML-Kopf deklarieren. Erkannt werden dabei sowohl die „alten" HTTP-Header-Entsprechungen in Meta-Tags der Art:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

als auch HTML-5-Metatags der folgenden Art:

```
<meta charset="utf-8">
```

Im Falle von X(HT)ML-Dateien wird auch das `encoding`-Attribut der XML-Deklaration unterstützt:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Falls Sie kein Charset deklarieren, versucht das Übungssystem, dieses zu erraten, aber es kann natürlich nicht garantiert werden, dass das immer korrekt funktioniert. Sofern Sie ausschließlich US-ASCII-Zeichen verwenden und alle Sonderzeichen durch HTML-Entitäten (z.B. `&auml;` statt `ä`) kodieren, kann ausnahmsweise auf eine Deklaration verzichtet werden – empfohlen wird das aber ausdrücklich nicht.

(Siehe auch: Unicode-Support und dynamische Charset-Erkennung <https://online-uebungssystem.fernuni-hagen.de/download/unicode/unicode-support.html>)

TeX-Formelsatz, Quelltext-Syntax-Highlighting

Sie können in den HTML-Texten TeX-Formeln mit Hilfe einer bestimmten Syntax einbinden, außerdem können unter gewissen Bedingungen auch Studenten TeX-Formeln in ihren Lösungen verwenden. Siehe dazu das gesonderte Handbuch zum TeX-Formelsatz im Online-Übungssystem <https://online-uebungssystem.fernuni-hagen.de/download/TeXIntegration/TeXIntegration.html> .

Falls Sie Aufgaben zu Programmierkursen anbieten, können Sie ein automatisches Syntax-Highlighting verwenden – sowohl für Quelltextauszüge in den Aufgabenstellungen als auch zur Formatierung von Quelltext-Einsendung der Studenten in Quittungen und Korrekturen. Setzen Sie

Code dazu in HTML-`code`-Elemente innerhalb von `pre`-Elementen. Details finden Sie im Handbuch zu WYSIWYG-Editoren, In-Browser-Korrektur und Syntax-Highlighting <https://online-uebungssystem.fernuni-hagen.de/download/WYSIWYG_IBK/WYSIWYG_IBK.html> .

Kommentare

HTML-Dateien können ja bekanntlich Kommentare der Form

```
<!-- Dies ist ein Kommentar. -->
```

enthalten. Falls Sie derartige Kommentare im Aufgabenformular einbinden, werden diese jedoch mit an den Browser des Studenten gesendet. Der Student sieht sie im Normalfall nicht, da sie nicht in der gerenderten Webseite angezeigt werden, kann sie aber durch Umschalten in die HTML-Quellcode-Ansicht jederzeit einsehen.

Falls Sie interne Kommentare zu bestimmten Dokumentationszwecken in den Aufgabenformularen speichern wollen, die Studenten nicht sehen sollen, verwenden Sie die folgende Syntax:

```
<!--| Dies ist ein „privater“ Kommentar, der nicht an den Browser gesendet wird.
|-->
```

Das Online-Übungssystem erkennt die Pipe-Symbole (`|`) an den Kommentar-Klammern und filtert diese Tags in der „Live-Ansicht“ der Aufgabenstellung aus, sendet sie also nicht an den Browser des Studenten weiter. Nur in der Rohfassung für Aufgabenautoren sind diese Kommentare sichtbar. (Das gilt nicht nur für die Aufgabenformulare `aufgabex.y.html`, sondern auch für die anderen im Folgenden noch zu erstellenden Standard-Kursressourcen.)

Kommentare der Aufgabenerstellungsassistenten

Falls Sie eine per Aufgabenerstellungsassistent generierte Aufgabenressource öffnen, finden Sie darin übrigens einige solcher Kommentare. Das folgende Listing zeigt z.B. einen Ausschnitt aus einer mit dem HB-Assistenten erzeugten Aufgaben-HTML:

```
<!--|EL:UEBERSCHRIFT|-->
<h2>a) Kochendes Wasser (10 Punkte)</h2>
<!--|/EL:UEBERSCHRIFT|-->

<!--|EL:AUFGABENTEXT|-->
<p>Nehmen Sie Stellung zu folgender Behauptung:</p>
<p><em>&bdquo;Wasser kocht/verdampft grunds&auml;tzlich erst ab einer
Temperatur von 100&deg;C.&ldquo;</em></p>
<!--|/EL:AUFGABENTEXT|-->
```

Derartige Kommentare sind Markierungen der Aufgabenerstellungsassistenten, die diese benötigen, um eine existierende Aufgaben-HTML-Seite wieder im Assistenten öffnen und nachbearbeiten zu können. Sofern Sie die so erzeugte Aufgabendatei in der fortgeschrittenen Aufgabenerstellung in einer Weise verändern wollen, die ohnehin nicht mit dem Assistenten erstellbar gewesen wäre, so dass auch eine spätere Bearbeitung Ihrer geänderten Aufgabendatei im Assistenten nicht möglich ist, so können Sie diese Kommentare getrost löschen. Allgemein ist zu sagen, dass manuelle

Änderungen an assistentengenerierten Ressourcen bei späterer Nachbearbeitung im Assistenten und somit Neugenerierung der Ressourcen durch die Assistenten verloren gehen können. Das Übungssystem warnt daher auch bei jedem Versuch, eine Aufgabe im Assistenten zu bearbeiten, nachdem sie bereits in der fortgeschrittenen Aufgabenerstellung bearbeitet worden ist.

Variablen des Online-Übungssystems

Betrachten wir noch einmal das oben stehende HTML-Listing, so fallen die vielen Schlüsselwörter auf, die durch `$` eingeleitet werden. Dabei handelt es sich um Variablen, die bei Auslieferung der Seite an den Endnutzer vom Online-Übungssystem dynamisch durch konkrete Werte ersetzt werden.

Die Variable `$Aufgabenname` z.B. wird ersetzt durch den Namen der Aufgabe, den Sie in der Aufgabeneinrichtung festlegen (s.o.). Wenn Sie diesen in die Seite einsetzen wollen, ist es immer ratsam, die Variable zu verwenden, statt den Namen redundant auch noch in der HTML-Datei einzusetzen – was doppelten Änderungsaufwand im Falle einer späteren Umbenennung zur Folge hätte.

Auch die Aufgabennummer sollte nur per Variable (`$AufgabenNr`) referenziert werden, denn falls Sie die Aufgabe später noch verschieben sollten (die Reihenfolge der Aufgaben in einem Heft ändern oder Aufgaben in ein anderes Heft verschieben), wodurch sich die Nummer der Aufgabe ändert, wäre es lästig und fehleranfällig, danach auch die Aufgaben-HTML-Datei wieder bearbeiten und darin jedes Vorkommen der Nummer korrigieren zu müssen.

Falls Sie URLs angeben (z.B. Downloadlinks zu PDFs oder Referenzen auf Bilddateien aus den Kursressourcen), so kommen normalerweise ebenfalls Variablen zum Einsatz. Der einzige URL in obigem Quellcode ist der obligatorische URL, an welche die Formulareingaben zu senden sind, also das `action`-Attribut des `form`-Tags. Dieser besteht hier – abgesehen vom Servicenamen `Einsendung` und den Slashes (`/`) – praktisch ausschließlich aus Variablen. Das sollte immer so beibehalten (und für andere Links ebenso gehalten) werden, denn auf diese Weise ist sichergestellt, dass die Einsendung bzw. das Nachladen eines Bildes oder Links etc. immer funktionieren, auch wenn die Aufgabe verschoben wird oder der Kurs in einem späteren Semester erneut angeboten wird, ohne dass die HTML-Ressourcen angepasst werden müssen.

> *Setzen Sie allgemein nie konkrete Werte wie Kursnummer, Aufgabennummer, Semesterkennung oder Ähnliches ein, für die es auch Variablen gibt!*

Im Folgenden werden die wichtigsten Variablen vorgestellt, die im Rahmen von Aufgabenressourcen definiert sind. (*Die Groß-/Kleinschreibung muss genau eingehalten werden!*)

System

Variable	Bedeutung
`$WebAssignServer`	Derzeit ersetzt durch `https://online-uebungssystem.fernuni-hagen.de`, wird aber auch funktionieren, falls sich dieser Servername in Zukunft einmal ändern sollte.

Veranstalter (Lehrgebiet oder Einrichtung)

Variable	Bedeutung	Beispiel
`$Veranstaltername`	Kennung des Veranstalters, insb. in URLs verwendet	six
`$Lehrgebiet`	Anzeigename des Veranstalters (des Lehrgebiets oder der Einrichtung, die den Kurs anbietet)	Software Engineering
`$Universitaet`	Name der Universität des Veranstalters	FernUniversität in Hagen
`$Professor`	Name des Lehrgebietsinhabers o.ä.	Prof. Dr. Six

Kurs(version)

Variable	Bedeutung	Beispiel
`$KursNr`	Fünfstellige Kursnummer, auch in URLs verwendet	01612
`$Kursname`	Name des Kurses	Konzepte imperativer Programmierung
`$VersionsNr`	Kennung der Kursversion in Form des Semesters, in der diese angeboten wird. Insb. in URLs verwendet. Beginnt mit `WS` oder `SS` für Winter- bzw. Sommersemester, gefolgt von zweistelliger Jahreszahl.	WS09
`$Versionsname`	Name der Kursversion / des Semesters ausgeschrieben	Wintersemester 2009/2010
`$BetreuerEmail`	In den Kursparametern hinterlegte E-Mail-Adresse der Kursbetreuung	kurs1612@fernuni-hagen.de

Aufgabenheft

Variable	Bedeutung	Beispiel(e)
`$AufgabenheftNr`	Nummer des Aufgabenhefts	1
`$Aufgabenheftname`	Gibt den optionalen Namen des Aufgabenhefts aus, oder gar nichts (Variable wird ersatzlos entfernt), falls kein Name vergeben wurde.	Erster Aufgabenblock
`$Aufgabenheftname[prefix]`	Gibt den optionalen Namen des Aufgabenhefts aus und stellt ihm den Text `prefix` voran, sofern der Namen vergeben wurde, andernfalls wird gar nichts ausgegeben.	
`$Aufgabenheftname[prefix|postfix]`	Wie oben, stellt aber dem Namen – falls vorhanden – noch das `postfix` hinten an.	
`$AufgabenheftnameOderNr`	Gibt (nur) den Aufgabenheft-Namen aus, sofern einer festgelegt wurde, ansonsten den Text »Aufgabenheft i«, wobei *i* durch die Heft-Nummer ersetzt wird	Aufgabenheft 1
`$Bearbeitungsbeginn`	Datum und Uhrzeit, ab wann die Aufgaben des Hefts einsehbar sind und bearbeitet werden können.	01.04.2019, 00:00
`$Bearbeitungsende`	Datum und Uhrzeit, bis wann die Aufgaben des Hefts noch bearbeitet werden können (Einsendeschluss) und ab wann die Musterlösungen (im Regelfall) sichtbar werden. Falls das Bearbeitungsende um Mitternacht liegt, erfolgt die Ausgabe als 24 Uhr des letzten Einsendetags (statt 0 Uhr des Folgetags).	12.05.2019. 24:00

In der Regel werden Sie – zumindest innerhalb von Aufgabenformularen – nur die erste Variable `$AufgabenheftNr` benötigen, und zwar insbesondere im URL fürs `form`-Tag (s.u.).

Die Aufgabenheftname-Variablen dienen dazu, im Text das Aufgabenheft zu benennen. Dabei sind die ersten Varianten für eine Kombination mit einer Angabe der Heft-Nummer vorgesehen, die optional um den Namen ergänzt werden soll, z.B.:

```
Aufgabenheft $AufgabenheftNr$Aufgabenheftname[ (|)]
```

Obiges Beispiel gibt für unbenannte Aufgabenhefte nur Text der Form »Aufgabenheft 1« aus, für benannte dagegen »Aufgabenheft 1 (Name)«.

Falls bei einem benannten Aufgabenheft *nur* der Name ausgegeben werden soll, bei einem unbenannten Heft dagegen »Aufgabenheft 1«, dann verwenden Sie die Variable `$AufgabenheftnameOderNr`.

Auch die Bearbeitungstermine werden Sie in Aufgabenseiten normalerweise nicht anzeigen müssen: Studenten können diese Termine im Übungssystem bereits in der Aufgabenübersicht sehen, und falls Sie einzelne Aufgaben per LTI in eine Lernumgebung wie Moodle einbetten, enthält die eingebettete

Darstellung auch schon die Möglichkeit, die Termine einzublenden.

Aufgabe

Variable	Bedeutung	Beispiel(e)
`$AufgabenNr`	Nummer der Aufgabe (innerhalb eines Aufgabenhefts)	2
`$Aufgabenname`	Der von Ihnen vergebene Aufgabenname	»Aufgabe 2« oder »Kontrollstrukturen«
`$MaximalPunkte`	In der Aufgabe erreichbare Punktzahl	»50« oder »0«
`$MaximalPunkteUeberschrift`	Text mit erreichbarer Punktzahl, wie er einer Aufgabenüberschrift angehängt werden kann – leer, falls keine Punkte erreichbar sind	»(50 Punkte)« oder leere Ausgabe bei 0 erreichbaren Punkten

Webdesign / Embedding

Da Sie hier HTML-Dateien bearbeiten, ist es zwar möglich, die gesamte Webseite für die Aufgabe selbst zu gestalten, empfohlen ist jedoch, dies nicht pro Datei zu tun, sondern in den einzelnen Aufgabendateien im Wesentlichen die eigentlichen Aufgabeninhalte einzufügen. Diese Inhalte werden dann im Normalfall ins Standard-Webdesign des Online-Übungssystems eingebettet, Sie können aber auch abweichende Seitenvorlagen erstellen, siehe: Abweichendes Webdesign verwenden <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#webdesign> . In der Aufgaben-HTML-Datei werden diese Inhalte nur mit einem minimalen HTML-Gerüst versehen, das eine Fallback-Darstellung für den Fehlerfall bereitstellt.

Die eigentlichen Inhalte werden mit einem speziellen Variablenpaar markiert:

Variable	Bedeutung
`$EMBED`	Beginn des eigentlichen Inhalts, der in eine Seitenvorlage eingebettet werden soll. (Standardmodus)
`$EMBEDSupportsDarkMode`	Wie `$EMBED`, damit deklarieren Sie aber zusätzlich, dass alle Bilder in der Seite Dunkelmodus-kompatibel sind und nicht – wie im Standardmodus – zur Sicherheit grau hinterlegt werden sollen.
`$EMBEDDisableDarkMode`	Wie `$EMBED`, damit legen Sie aber zusätzlich fest, dass selbst bei vom Nutzer aktiviertem »Dark Mode« diese Seite immer in der hellen Ansicht geladen werden soll.
`$/EMBED`	Ende des einzubettenden Inhalts.

Beide Variablen (`$EMBED`, ggf. mit Suffix, sowie `$/EMBED`) müssen im `body` des HTMLs stehen.

Neben dem so markierten Body-Ausschnitt werden auch noch Teile des `head`s wie der `title` oder CSS-Links übernommen. Die Tags `<form ...>` und `</form>` dürfen auch außerhalb des EMBED-Blocks stehen, sofern es nicht mehrere von ihnen gibt. Details zu diesem Verfahren werden in einem gesonderten Handbuch <https://online-uebungssystem.fernuni-hagen.de/download/Designvorlagen2018/Designvorlagen2018OnlineUebungssystem.html> erläutert.

Was es mit dem »Dark Mode« und den entsprechenden beiden Markierungen

`$EMBEDSupportsDarkMode` und `$EMBEDDisableDarkMode` auf sich hat, wird im Dunkelmodus-Handbuch <https://online-uebungssystem.fernuni-hagen.de/download/DarkMode/DarkMode.html> behandelt.

Noch ein Hinweis: In den erzeugten Vorlageseiten, deren Quelltexte oben auch teilweise abgedruckt sind, folgt die `$EMBED`-Variable meist auf eine `h1`-Überschrift. Das hat folgenden Grund: Die Standard-Seitenlayout-Vorlagen des Online-Übungssystems legen bereits selbst eine Überschrift `h1` fest, unterhalb welcher der mit `$EMBED` markierte Inhalt eingebettet wird. Die `h1`-Überschrift der Kursressourcen dient lediglich der Fallback-Darstellung, falls das Embedding nicht funktioniert.

Spezielle Kontrollvariablen

Variable	Bedeutung
`$INCLUDE(...)`	SSI (Server-Side-Include): Sucht nach einer Kursressource mit Namen ... und fügt deren Inhalt an dieser Stelle ein. Statt des Includes von HTML-Fragmenten mit zentralen Webdesign-Inhalten wird aber normalerweise eher das Embedding (s.o.) empfohlen.
`$ESCAPE(...)`	dient zur Anzeige von HTML-Quellcode (nur in seltenen Ausnahmefällen sinnvoll, z.B. bei Aufgaben zur Sprache HTML): Falls der Text in den Klammern (...) – der selbst keine runden Klammern enthalten darf! – HTML-Tags oder -Entites enthält, werden diese automatisch escaped, so dass sie vom Browser nicht als HTML interpretiert, sondern unverändert angezeigt werden. Falls der String ... selbst wieder Variablen enthält (z.B. `$ESCAPE($FeldA1)`), werden die zuerst aufgelöst und danach erst auf das Endergebnis das „Escaping“ angewandt.
`$IfExistsA1(...)`	bewirkt die Anzeige des in Klammern gefassten *bedingten Textes* (...) im Fall, dass der Student in Eingabefeld `1` der Teilaufgabe `A` bereits etwas eingesendet hat. Hat der Student dagegen noch nie etwas in diesem Feld bzw. dieser Teilaufgabe eingesendet, wird die gesamte Variable einschließlich des bedingten Textes ausgeblendet. Der bedingte Text darf selbst keine runden Klammern enthalten, damit sein Ende korrekt feststellbar ist. Außerdem kann er selbst keine Variablen (`$...`) enthalten, sondern wird immer entweder unverändert oder gar nicht in der Ausgabe eingeblendet. (Hinweis: Auch Leereinsendungen, also das Einsenden einer Teilaufgabe, ohne irgendetwas in das Eingabefeld eingegeben zu haben, gelten als existierende Einsendungen.)
`$IfNotExistsA1(...)`	analog, nur genau umgekehrt: Bewirkt die Anzeige des bedingten Textes ... im Fall, dass der Student in dem Feld noch nie etwas eingesendet hat (auch keine Leereingabe).
`$IfNotEmptyA1(...)`	ähnlich wie `$IfExistsA1(...)`, auch hier ist in Klammern ein bedingter Text gegeben, der jedoch unter einer etwas anderen Bedingung angezeigt wird: Immer, falls der Student in Feld `1` der Teilaufgabe `A` eine *nicht-leere* Einsendung vorgenommen hat. Der Unterschied zu `IfExists...` liegt also darin, dass bei dieser Variablen *nicht nur* eine Einsendung existieren muss, sondern diese Einsendung obendrein nicht leer sein darf. Für Textfelder heißt „nicht leer“, dass dort mindestens ein Zeichen eingegeben worden sein muss. Für Checkboxen heißt „nicht leer“, dass mindestens eine davon angekreuzt sein muss. Bei Dateiupload-Feldern heißt „nicht leer“, dass dort eine Datei eingesendet worden sein muss. Ein Student kann z.B. nach erstmaligem Upload einer Datei diese wieder „zurückziehen“, indem er den Upload-Button nochmals betätigt, ohne eine neue Datei auszuwählen: Dann wird die letzte Dateieinsendung durch eine *leere* Einsendung überschrieben, für die `$IfExists...` immer noch den bedingten Text anzeigen

Variable	Bedeutung
	würde, `$IfNotEmtpy...` dagegen nicht mehr.
`$IfEmptyA1(...)`	analog zu `$IfNotEmptyA1(...)`: Bewirkt die Anzeige des bedingten Textes ... genau dann, wenn zu dem genannten Feld eine leere Einsendung vorliegt. Genau wie `$IfNotEmpty...` impliziert also auch diese Bedingung, dass eine Einsendung existieren muss, nur dass diese eben leer (statt nicht-leer) sein muss.
`$IfNotExistsOrEmptyA1(...)`	Disjunktion von `IfNotExists` und `IfEmpty`: Der Text in Klammern wird genau dann eingeblendet, wenn *entweder* noch gar keine Einsendung im Feld A1 vorliegt *oder* eine leere Einsendung vorliegt. Oder anders gesagt: Wenn *keine* nicht-leere Einsendung im Feld vorliegt, effektiv also das „echte" Gegenteil von `IfNotEmptyA1(...)`.
`$Resource(file1)orElse(file2)`	Prüft, ob in den Kursressourcen eine Datei namens `file1` existiert. Falls ja, wird die gesamte Variable durch `file1`, andernfalls durch `file2` ersetzt. Die Dateinamen `file1` und `file2` dürfen wiederum weitere Variablen enthalten. Falls `file1` z.B. die Variable `$MatrikelNr` enthält, können so individuelle Dateien für im Vorfeld angemeldete Studenten verlinkt werden, siehe Individuelle Dateien für bekannte Studenten verlinken.
`$JavaScriptRequired`	kann eingebunden werden, wenn der Browser bei abgeschaltetem JavaScript an der Stelle der Variablen eine Meldungsbox anzeigen soll, dass JavaScript für die volle Funktionalität benötigt wird. Bei aktivem JavaScript wird die Variable einfach ersatzlos ausgeblendet.
`$NOEXPAND`	sorgt dafür, dass nachfolgende mit `$` beginnende Tokens nicht als Variable erkannt werden, selbst wenn sie zufällig mit einer übereinstimmen.
`$/NOEXPAND`	hebt die Wirkung von `$NOEXPAND` für den nachfolgenden Text wieder auf.

Als Alternativen zu den obigen Variablen `$IfExistsA1(...)`, `$IfNotEmptyA1(...)` etc. mit Argumenten in Klammern (worin z.B: keine weiteren Klammern vorkommen dürfen), gibt es folgende *Variablen-Paare* aus je einer öffnenden und einer zugehörigen schließenden Kontrollvariablen:

Variable	Bedeutung
`$IfExistsA1`	analog zu `$IfExistsA1(...)` , aber auch nutzbar für *bedingten Text*, der selbst wiederum Variablen oder runde Klammern enthalten soll: Diese Variable leitet bedingten Text ein, der genau dann angezeigt werden soll, wenn der Student zu Feld `A1` schon einmal etwas eingesendet hat. Dieser bedingte Text muss *unbedingt* durch die folgende Variable wieder geschlossen werden:
`$/IfExistsA1`	beendet den mit `$IfExistsA1` eingeleiteten bedingten Text.
`$IfNotExistsA1`	analog zu `$IfNotExistsA1(...)` : Leitet einen nur bei Nicht-Existenz einer Einsendung des Studenten in Feld `A1` einzublendenden *bedingten Text* ein, der Klammern oder andere Variablen enthalten darf und durch `$/IfNotEmptyA1` abgeschlossen werden muss.
`$/IfNotExistsA1`	beendet den mit `$IfNotExistsA1` eingeleiteten bedingten Text.
`$IfNotEmptyA1`	analog zu `$IfNotEmptyA1(...)` : Leitet einen nur bei Existenz einer nicht-leeren Einsendung in Feld `A1` einzublendenden *bedingten Text* ein, der Klammern oder andere Variablen enthalten darf und durch `$/IfNotEmptyA1` abgeschlossen werden muss.
`$/IfNotEmptyA1`	beendet den mit `$IfNotEmptyA1` eingeleiteten bedingten Text.
`$IfEmptyA1`	analog zu `$IfEmptyA1(...)` : Leitet einen nur bei Existenz einer leeren Einsendung in Feld `A1` einzublendenden *bedingten Text* ein, der Klammern oder andere Variablen enthalten darf und durch `$/IfEmptyA1` abgeschlossen werden muss.
`$/IfEmptyA1`	beendet den mit `$IfEmptyA1` eingeleiteten bedingten Text.
`$IfNotExistsOrEmptyA1`	analog zu `$IfNotExistsOrEmptyA1(...)` : Beginnmarke eines bedingten Textes, der nur angezeigt werden soll, falls in Feld A1 keine nicht-leere Einsendung vorliegt, sondern vielmehr gar keine oder eine leere Einsendung.
`$/IfNotExistsOrEmptyA1`	beendet den mit `$IfNotExistsOrEmptyA1` eingeleiteten bedingten Text.

Hinweis: Die `$If(Not)Exists`-Variablen arbeiten komplett unabhängig von der weiter unten beschriebenen `data-ignore`-Einstellung. D.h. selbst wenn für ein Eingabefeld eingestellt wird, dass ein leer gelassenes Feld nicht implizieren darf, dass die Aufgabe bereits als bearbeitet gilt, so wird `$IfExists...` für ein leer gelassenes Feld nach Formular-Einsendung doch *immer* erfüllt sein.

Randomisierung

Initialisierung von Randomisierung

Variable	Bedeutung
`$Randomize(i)`	Wenn `$Random`-Variablen (s.u.) eingesetzt werden, um die Randomisierung einer Aufgabenseite zu konfigurieren, kann (irgendwo vor der ersten `$Random`-Variable) eine `$Randomize`-Variable eingefügt werden, welche Einstellungen für die Randomisierung festlegt. Wird diese Variable *nicht* angegeben, so erzeugen die Random-Variablen eine Randomisierung in *Grundeinstellung:* Genau *einer* der Random-Blöcke wird pro Teilnehmer zufällig gewählt und angezeigt, alle anderen ausgeblendet. Falls Sie *mehr* als einen Block auswählen möchten, legen Sie das über die Randomize-Variable fest, indem Sie die *Selektionsgröße i* (Anzahl der auszuwählenden Fragen) in die Klammern dahinter schreiben, z.B. `$Randomize(3)` für die Anzeige von 3 der N (≥3) folgenden Random-Blöcke. Falls Sie *blocklokale Randomisierung* verwenden (s.u.), ist innerhalb eines (Random- oder Fixed-)Blocks wieder eine entsprechende `$Randomize(...)`-Variable vor der ersten Unter-Random-Variable (wie `$Random.1`) zu notieren, sofern nicht die Grundeinstellung (Selektion genau eines der Random-Unterblöcke) verwendet

Variable	Bedeutung
	Grundeinstellung (Selektion genau eines der Random-Unterblöcke) verwendet werden soll. Anders als bei den Randomize-Varianten sind in der Randomize-Variable keine „Hierarchie-Punkte“ einzufügen.
`$Randomize(i, Shuffle)`	Falls die ausgewählten Blöcke/Fragen außerdem in einer *zufälligen Reihenfolge* präsentiert werden sollen, geben Sie als zweiten Parameter `"Shuffle"` hinter der Selektionsgröße an, getrennt durch Komma, also z.B.: `$Randomize(3, Shuffle)`. Falls Sie *keine* zufällige Selektion wünschen, sondern stets *alle* Random-Blöcke lediglich in zufälliger Reihenfolge anzeigen lassen möchten, erreichen Sie dies, indem Sie exakt die Anzahl der Random-Blöcke als Selektionsgröße in die Klammern schreiben.
`$Randomize(i, ShuffleFixed)`	Falls Sie auf der Hierarchieebene, auf die sich die Randomize-Variable bezieht, sowohl Random- als auch Fixed-Blöcke (s.u.) stehen haben, werden bei Angabe von `Shuffle` in der Randomize-Variable nur die Random-Blöcke untereinander in eine zufällige Reihenfolge gebracht, die Fixed-Blöcke bleiben an ihrem Ort fixiert und dienen nur zur Festlegung eines Blocks, innerhalb dessen lokal randomisiert werden kann. Durch Angabe von `ShuffleFixed` statt `Shuffle` bewirken Sie, dass in diesem Fall alle Blöcke, d.h. sowohl die Random- als auch die Fixed-Blöcke, vermischt werden / in eine Zufallsreihenfolge gebracht werden. (Eine zufällige *Selektion* von Blöcken, also Auswahl einer zufälligen Teilmenge, beschränkt sich dagegen weiterhin auf die Random-Blöcke, die Fixed-Blöcke sind in jedem Fall immer sichtbar.)
`$Randomize(i, j)`	Für Randomize-Variablen in Sub-Blöcken kann neben der Selektionsgröße *i* ein zweiter numerischer Parameter *j* angegeben werden, der genutzt werden kann, falls in mehreren Sub-Blöcken jeweils *identisch* randomisiert werden soll. Falls Sie z.B. die Spalten einer HTML-Tabelle randomisieren möchten, können Sie jede Tabellenzeile als Fixed-Block (oder Random-Block, falls die Zeilen selbst auch randomisiert werden sollen) notieren und *pro Zeile* dann die einzelnen Tabellenzellen in Sub-Randomblöcke einfassen. Nun würden allerdings in der Grundeinstellung für jede der Tabellenzeilen deren Zellen individuell randomisiert, d.h. der Spaltenzusammenhang ginge verloren. Um sicherzustellen, dass die Unter-Randomisierung der Zellen für alle Tabellenzeilen identisch erfolgt, die Spalten also zusammenhängend bleiben, verwenden Sie im Block jeder Tabellenzeile eine Randomize-Variable, in der Sie denselben Wert *j* als zweite Zahl angeben. Damit wird der Randomisierer für jede Tabellenzeile dieselbe Randomisierung vornehmen (gleiche Zellen-/Spaltenzahl vorausgesetzt). Siehe auch Beispiel 3 in Abschnitt Blocklokale Randomisierung.
`$Randomize(i, Shuffle, j)`	Kombination von `$Randomize(i, j)` und `$Randomize(i, Shuffle)`.
`$Randomize(i, ShuffleFixed, j)`	Kombination von `$Randomize(i, j)` und `$Randomize(i, ShuffleFixed)`.
`$RandomizeHeft(…)`	Wie `$Randomize(…)`, nur aktiviert der Zusatz »Heft« im Namen zusätzlich die eindeutige Fragenauswahl innerhalb von mehreren aus gleich vielen Fragen bestehenden Aufgaben desselben Aufgabenhefts, siehe Unterschiedliche Fragenauswahl innerhalb eines Hefts erzwingen. Angaben in Klammern können ansonsten analog zu `$Randomize(…)` gebildet werden. Die Klammern mit den Zusatzangaben können auch ganz entfallen, also nur `$RandomizeHeft` angegeben werden. Dann gelten die Grundeinstellungen (zufällige Selektion genau eines Random-Blocks), nur eben in Kombination mit dem Modus zur unterschiedlichen Auswahl in verschiedenen Aufgaben desselben Hefts.

Blöcke (Random- und Fixed-Blöcke)

Variable	Bedeutung
`$Random1`	beginnt einen „Random-Block“, d.h. eine Alternative für die Fragen-Randomisierung. Damit die Randomisierung Sinn ergibt, müssen mindestens zwei solche Random-Blöcke in der Aufgabenseite existieren, beide Random-Variablen müssen mit einer anderen Nummer enden (z.B. 1 und 2, aber die konkreten Werte und sogar die Reihenfolge in der Seite ist egal, die Nummern müssen lediglich eindeutig sein).
`$/Random1`	beendet den mit `$Random1` geöffneten Random-Block. Alles zwischen diesen beiden Variablen wird einem Studenten zufällig entweder angezeigt oder komplett aus der Aufgabenseite entfernt. Unten im Abschnitt Formular mit Fragen-Randomisierung wird noch genauer darauf eingegangen.
`$Fixed1`	beginnt einen „Fixed-Block“. Fixed-Blöcke werden im Regelfall nicht randomisiert, genauer: Eine *Zufallsauswahl* erfolgt *immer* nur aus Random-Blöcken, während eine *Zufallsreihenfolge (Shuffle) optional auch Fixed-Blöcke* mit Random-Blöcken vermischen kann (`$Randomize(i, ShuffleFixed)`). So kann also ein Anwendungsgebiet von Fixed-Blöcken darin bestehen, bei einer Kombination von Zufallsauswahl mit Zufallsreihenfolge das zufällige Ausblenden bestimmter Blöcke zu verhindern, die aber dennoch mit den anderen, ausblendbaren Randomblöcken vermischt werden sollen. Das zweite und häufigere Anwendungsgebiet von Fixed-Blöcken ist die Möglichkeit, eine blocklokale „Sub-Randomisierung“ umzusetzen, z.B. mehrere Multiple-Choice-Fragen in jeweils einen eigenen Fixed-Block einzuschließen, um dann lokal pro MC-Frage deren Antwortalternativen randomisieren zu können. Siehe Beispiele in Abschnitt Blocklokale Randomisierung.
`$/Fixed1`	beendet den mit `$Fixed1` begonnenen Fixed-Block.
`$Random.1`	beginnt einen „Sub-Random-Block“, also einen Random-Block innerhalb eines mit `$Random1` oder `$Fixed1` begonnenen Blocks (Ziffern nur beispielhaft). Es sind auch noch weitere Schachtelungen möglich, innerhalb eines `$Random.1`-Blocks können also weitere Random-Blöcke stehen, die dann mit `$Random..1` und `$/Random..1` einzuschließen sind, etc. Die Anzahl der Punkte entspricht also der Schachtelungstiefe: Blockvariablen auf oberster ebene tragen keinen Punkt, Unterblöcke einen Punkt, Unter-Unterblöcke 2 Punkte etc. (Die Punkt-Notation wurde in Anlehnung an hierarchische Kapitelnumerierungen gewählt, also der Schreibweise, nach der z.B. „1.2.1“ den ersten Unterabschnitt im zweiten Abschnitt von Kapitel 1 bezeichnet; Allerdings ist bei den Blöcken nur jeweils die letzte Nummer, die Blocknummer, von Interesse; die Nummern der übergeordneten Blöcke müssen und dürfen in der Variablen nicht mit genannt werden, weshalb vor und zwischen den Punkten keine Nummern stehen.)
`$/Random.1`	beendet den mit `$Random.1` begonnenen Sub-Block.
`$Fixed.1`	Analog zu Random-Blöcken können auch Fixed-Blöcke als Sub-Blöcke in übergeordneten Random- oder Fixed-Blöcken stehen. Auch hier können je nach Schachtelungstiefe mehrere Punkte zwischen `$Fixed` und der Blocknummer stehen.
`$/Fixed.1`	Beendet den mit `$Fixed.1` begonnenen Fixed-Unterblock.

Blockzähler (Counter)

In Verbindung mit Blöcken (`$Fixed` oder `$Random`) können Blockzähler-Variablen eingesetzt werden. Eine solche Countervariable wird in jedem Block, in dem Sie vorkommt, um 1 erhöht. Das ist natürlich in erster Linie für den Einsatz im Rahmen der Randomisierung (zufällige Auswahl/Selektion und/oder zufällige Reihenfolge) von Blöcken vorgesehen, denn dort wo Inhalte feststehen, könnten ja auch direkt die feststehenden Nummern direkt verwendet werden und besteht kein Bedarf für Zählervariablen.

Variable	Bedeutung
`$Counter1`	Ein Zähler mit Nummer 1, der innerhalb von `$Randomi`- oder `$Fixedi`-Blöcken (wobei `i` jeweils für eine Blocknummer steht, s.o.) verwendet werden kann und in jedem Block um 1 größer wird, während Mehrfachnennungen dieser Variablen innerhalb desselben Blocks denselben Wert behalten. Das gilt auch für Unterblöcke: In jedem Unterblock (wie z.B. `$Random..1`) desselben Blocks behält der Counter seinen Wert bei. Falls Sie mehrere solche Zähler brauchen sollten, verwenden Sie statt der `1` weitere Counter-Nummern.
`$SetCounter1(…)`	Vor dem ersten Auftreten von `$Counter1` kann optional mit der zugehörigen SetCounter-Variablen der Zähler initialisiert werden. Wenn es keine solche SetCounter-Variable zu einer Counter-Variable gibt, beginnt der Zählern mit dem Wert 1, d.h. im ersten Block wird dann `$Counter1` zur Ausgabe `1` führen, im zweiten Block zu `2` etc. Mit SetCounter können Sie den Startwert und das Ausgabeformat ändern. So führt `$SetCounter(0)` dazu, dass es sich nach wie vor um einen numerischen Zähler handelt, der jedoch mit Nummer Null beginnt, also im ersten Block als `0`, im zweiten als `1` ausgegeben wird. Alternativ zu einer Zahl kann auch ein Groß- oder Kleinbuchstabe angegeben werden. So führt z.B. `$SetCounter1(A)` dazu, dass der Counter im ersten Block zur Ausgabe von `A`, im zweiten zur Ausgabe von `B`, im dritten zu `C` führt etc.
`$Counter.1`	Analog zu der Hierarchieschreibweise für Random- und Fixed-Variablen (s.o.) können auch in Counter-Variablen Block-Schachtelungstiefen durch Punkte angegeben werden: `$Counter.1` ist eine Variable für einen *nur lokal innerhalb* eines (Top-Level-)Random- oder Fixed-Blocks wie `$Random1 … /$Random1` verwendeten Zähler, der seinen Wert in jedem Sub-Block (wie `$Random.1 … $/Random.1`) jeweils um 1 erhöht. Innerhalb eines `$Random.1 … /$Random.1`-Subblocks können wiederum ebenfalls lokale Zähler wie `$Counter..1` verwendet werden, die dann dort lokal alle „Sub-Subblöcke“ (mit zwei Punkten vor der Nummer) durchnummerieren.
`$SetCounter.1(…)`	Ist ebenfalls blocklokal innerhalb eines Random- oder Fixed-Blocks verwendbar, um dort vorhandene `$Counter.1`-Variablen zu initialisieren. Muss zwischen dem Blockbeginn (wie `$Random1` oder `$Fixed1`) und dem ersten Vorkommen der Zählervariablen (`$Counter.1`) stehen. Wird die Variable nicht angegeben, verwendet der blocklokale Zähler das Standardformat: Zahlen beginnend ab 1.

Beispiel für Counter-Variablen:

```
$SetCounter1(A)
$Randomize(2)

$Random1
<h2>Block $Counter1 (in Random1)</h2>
$/Random1

$Random2
<h2>Block $Counter1 (in Random2)</h2>
  $SetCounter.1(0)
  $SetCounter.2(a)
  $Randomize(3, Shuffle)
  $Random.1
    <h3>Block $Counter1.$Counter.1 ($Counter.2) (in Random 2.1)</h3>
  $/Random.1
  $Random.2
    <h3>Block $Counter1.$Counter.1 ($Counter.2) (in Random 2.2)</h3>
  $/Random.2
  $Random.3
    <h3>Block $Counter1.$Counter.1 ($Counter.2) (in Random 2.3)</h3>
  $/Random.3
$/Random2

$Random3
  <h2>Block $Counter1 (in Random3)</h2>
  $Randomize(3, Shuffle)
  $Random.1
    <h3>Block $Counter1.$Counter.1 (in Random 3.1)</h3>
  $/Random.1
  $Random.2
    <h3>Block $Counter1.$Counter.1 (in Random 3.2)</h3>
  $/Random.2
  $Random.3
    <h3>Block $Counter1.$Counter.1 (in Random 3.3)</h3>
  $/Random.3
$/Random3
```

- Obiges Beispiel definiert einen Counter `$Counter1`, der in jedem der drei Blöcke `$Random1` bis `$Random3` jeweils einen konstanten Wert hat. Durch die `$Randomize(2)`-Variable wird eingestellt, dass von diesen drei Blöcken nur zwei ausgewählt werden, und durch `$SetCounter1(A)` wird festgelegt, dass `$Counter1` im ersten der beiden ausgewählten Blöcke den Wert `A`, im zweiten ausgewählten Block den Wert `B` bekommt.
- In den beiden Blöcken `$Random2` und `$Random3` wiederum wird *jeweils* ein lokaler Unterblock-Zähler `$Random.1` verwendet. D.h. `$Random.1` bezeichnet innerhalb von `$Random2 ... $/Random2` einen anderen Zähler als innerhalb von `$Random3 ... $/Random3`. Im Random2-Block ist dieser Zähler `$Random.1` außerdem mittels `$SetCounter.1(0)` so eingestellt worden, dass er ab 0 zu zählen beginnt, während der gleichnamige Zähler des Random3-Blocks bei 1 beginnt.
- Im Random2-Block wird außerdem demonstriert, dass man mehrere blocklokale Zähler parallel betreiben kann, hier `$Counter.1` und `$Counter.2`, die beide mittels `$SetCounter.1` bzw. `$SetCounter.2` unterschiedlich eingerichtet wurden. (Zwei identisch eingerichtete lokale Zähler im selben Block ergäben auch keinen Sinn, da sie immer identische Werte erzeugen würden.)

- `$Counter1` demonstriert den Zweck eines Counters bei *Random-Selektion* (Zufallsauswahl): Hätte man statt `$Counter1` jeweils direkt `A`, `B` oder `C` ins HTML geschrieben, so wäre nach der Zufallsauswahl eben nicht sichergestellt, dass die verbliebenen zwei Blöcke mit `A` und `B` bezeichnet sind.
- Die `$Counter.1`-Variablen dagegen demonstrieren den Counter-Einsatz bei *Shuffle* (Zufallsreihenfolge).

Das (zufällige) Ergebnis des obigen HTML-Codes könnte dann beispielsweise wie folgt aussehen (bei zufälliger Auswahl der Blöcke Random2 und Random3):

> ***Block A (in Random2)***
> *Block A.0 (a) (in Random 2.1)*
> *Block A.1 (b) (in Random 2.3)*
> *Block A.2 (c) (in Random 2.2)*
> ***Block B (in Random3)***
> *Block B.1 (in Random 3.3)*
> *Block B.2 (in Random 3.1)*
> *Block B.3 (in Random 3.2)*

Einsendungen

Es gibt spezielle Variablen, um die Daten, die ein Student zuletzt zu einer Aufgabe eingesendet hat, in HTML-Seiten einzubinden. Diese werden in erster Linie in Quittungs- und Korrekturseiten benötigt, um die zu quittierenden bzw. zu korrigierenden Eingaben in einem Dokument zusammenzustellen. *Für eine Auflistung dieser Variablen sei daher auf die nachfolgenden Abschnitte Variablen für Quittungsseiten und Variablen für Korrekturseiten verwiesen.*

In Aufgabenseiten werden die letzten Eingaben im Normalfall automatisch wieder ins Formular eingefügt. *Einzige Ausnahme sind Dateiuploads:* Wenn ein Aufgabenformular ein Dateiupload-Element enthält, kann der Student darin eine Datei zum Upload auswählen. Angezeigt wird darin nur der lokale Dateiname. Bei der Einsendung wird dieser aufgelöst und der Inhalt der Datei eingesendet. Es ist unmöglich, beim erneuten Aufruf des Aufgabenformulars dann wieder die Referenz auf die Originaldatei auf der Festplatte des Studenten (sofern diese nicht ohnehin schon gelöscht oder nachbearbeitet wurde) ins Formularfeld einzutragen. Ebenso wenig lässt sich der zuletzt eingesendete *Dateiinhalt* in einem *Dateiauswahl*-Feld einblenden.

Falls der Student also im Aufgabenformular eine ggf. schon vorhandene letzte Datei-Einsendung angezeigt bekommen soll (empfohlen!), müssen diese Variablen ausnahmsweise auch in der Aufgabenseite eingesetzt werden, z.B. wie folgt:

```
$IfExistsA2(Ihre zuletzt hochgeladene Datei: )$FeldA2
```

Die (im Abschnitt Spezielle Kontrollvariablen genauer erklärte) `$IfExistsA1`-Kontrollvariable bewirkt, dass der in Klammern stehende *bedingte Text* „Ihre zuletzt hochgeladene Datei: “ noch nicht angezeigt wird, falls der Student zu Feld A2 noch niemals etwas eingesendet hatte. Die Variable `$FeldA2` selbst ist in dem Fall ohnehin unsichtbar. Diese Struktur stellt außerdem sicher, dass im Falle, dass der Student einmal etwas eingesendet hat, ohne eine Datei ausgewählt zu haben, in der Seite der Text „Ihre zuletzt hochgeladene Datei: [leer]“[5] angezeigt wird, die ihm mitteilt, dass zwei eine Einsendung vorliegt, diese jedoch keine Datei enthält[6].

Links

Prinzipiell können Sie mit den oben genannten Variablen bereits beliebige benötigte interne Links realisieren, z.B. `$WebAssignServer/$Veranstaltername/KursStartSeite/$KursNr/$Versionsnr`.

Es gibt aber auch Variablen, die komplette Links zur Navigation zwischen Aufgaben realisieren. Im Normalfall, wenn Sie Ihre Aufgaben in das Standard-Webdesign des Übungssystems einbetten (s.o.), werden Sie derartige Links nicht benötigen, denn die Standard-Seitenlayoutvorlagen weisen bereits Navigationsmenüs (derzeit am linken Seitenrand) auf, mit denen von einer Aufgabenseite zu allen anderen Aufgabenseiten desselben Aufgabenhefts direkt gewechselt oder zur Aufgabenübersicht zurückgesprungen werden kann.

Bei komplett manuell erstelltem Webdesign können diese Variablen aber ggf. nützlich sein.

Variable	Bedeutung
`$NaechsteAufgabeURL`	URL für die nächste Aufgabe im selben Aufgabenheft (es muss eine nächste geben, d.h. die Variable darf nicht in der Aufgabenseite der letzten Aufgabe verwendet werden)
`$NaechsteAufgabe[…](…)`	Kompletter Link auf die nächste Aufgabe. Falls keine nächste Aufgabe mehr existiert, die aktuell angezeigte Aufgabe also die letzte des Hefts ist, wird die Variable ersatzlos entfernt. Die Zusätze in eckigen oder runden Klammern sind optional. In den runden Klammern können Sie eine benutzerdefinierte Linkbeschriftung angeben, andernfalls lautet die Standardbeschriftung »nächste Aufgabe«. Falls Sie die eckigen Klammern angeben, wird deren Inhalt in den Attributstring des `a`-Tags mit aufgenommen, dort können Sie also z.B. ein `title`- oder `class`-Attribut unterbringen.
`$VorherigeAufgabeURL`	URL für die vorherige Aufgabe im selben Aufgabenheft (darf entsprechend nicht in einer Aufgabe Nr. 1 verwendet werden)
`$VorherigeAufgabe[…](…)`	Kompletter Link auf die vorherige Aufgabe, analog zu `$NaechsteAufgabe[…](…)`, Standardbeschriftung: »vorherige Aufgabe«
`$KursuebersichtURL`	URL der Kursstartseite
`$Kursuebersicht`	Link auf die Kursstartseite (mit dem Text »Kursstartseite«)
`$AufgabenuebersichtURL`	URL der Studentenstartseite
`$Aufgabenuebersicht`	Link auf die Studentenstartseite (mit dem Text »Aufgabenübersicht«)
`$AufgabeURL`	URL der Aufgabenseite selbst
`$Aufgabe[…](…)`	Link auf die Aufgabenseite der aktuellen Aufgabe, analog zu `$NaechsteAufgabe[…](…)`, Standardbeschriftung: »Aufgabentext«. In der Aufgabenseite selbst i.d.R. nicht sinnvoll anwendbar, aber z.B. in der Quittungsseite zum Zurückkehren zur Aufgabenseite.

Beispiele zu `$NaechsteAufgabe[…](…)`:

- Die Variable kann ganz ohne die Klammerpaare angegeben werden: `$NaechsteAufgabe`. Sie erzeugt dann einen Link der Art:

```
<a href="https://…">nächste Aufgabe</a>
```

- In runden Klammern kann eine abweichende Beschriftung gewählt werden, z.B. `$NaechsteAufgabe(Weiter zur nächsten Aufgabe »)`:

```
<a href="https://…">Weiter zur nächsten Aufgabe »</a>
```

- In eckigen Klammern können Attribute hinzugefügt werden, z.B. `$NaechsteAufgabe[class="mylink" title="Weiter zur nächsten Aufgabe"]`:

```
<a href="https://…" class="mylink" title="Weiter zur nächsten
Aufgabe">nächste Aufgabe</a>
```

- Beides lässt sich kombinieren, z.B. `$NaechsteAufgabe[class="mylink" title="Weiter zur nächsten Aufgabe"](Weiter…)`:

```
<a href="https://…" class="mylink" title="Weiter zur nächsten
Aufgabe">Weiter…</a>
```

Anmerkung: Die oben angegebenen Linkbeschriftungen zu Variablen wie `$Kursuebersicht` oder Standard-Linkbeschriftungen zu Variablen wie `$VorherigeAufgabe` sind nicht dauerhaft garantiert, könnten also in Zukunft ggf. modifiziert werden.

QR-Codes

Die folgenden Variablen dienen zum Einfügen eines QR-Codes, der ebenfalls eine Link-Adresse enthalten kann, aber eben nicht zum Anklicken im Browser, sondern zum Scannen / „Abfotografieren“ mit z.B. einem Smartphone oder Tablet, um die darin kodierte Adresse auf dem anderen Gerät zu öffnen.

Im Standard-Seitenlayout (»Design 2018«) bietet das Online-Übungssystem bereits eine Handoff-Funktion: Im Seitenfuß befindet sich (bei geeigenten Seiten) ein Symbol, über das ein solcher QR-Code mit der Adresse der gerade geöffneten Seite jederzeit eingeblendet werden kann. Diese Variablen können demgegenüber folgende Mehrwerte bieten: Die damit eingebetteten QR-Codes sind…

1. dauerhaft sichtbar und frei/prominent in Ihrer Seite (z.B. Kursstartseite oder Aufgabenseite) platzierbar,
2. auch in I-Frames (z.B. per LTI in Moodle eingebetteten Aufgaben) verwendbar oder in Seiten, die nicht das Übungssystem-Standard-Seitenlayout im aktuellen Design verwenden, und
3. optional ist auch ein URL frei wählbar, d.h. es kann optional auch zu anderen Seiten oder Ressourcen wie PDF-Dateien verlinkt werden, nicht nur zur gerade geöffneten Webseite.

Variable	Bedeutung
`$QR`	Fügt einen QR-Code mit dem aktuellen Seiten-URL ein („Handoff“: Dieselbe / gerade geöffnete Seite auf einem anderen Gerät laden).
`$QR(…)`	Fügt einen QR-Code mit dem in Klammern stehenden Inhalt ein. Das sollte in der Regel der Ziel-URL sein, zu dem Sie verlinken möchten (z.B. mit einem Downloadlink zu einem Dokument). Der URL in den Klammern darf auch wieder Variablen enthalten

Die Variablen werden jeweils durch ein `span`-Element (Inline-Element) mit dem QR-Code ersetzt –

aktiviertes JavaScript vorausgesetzt. Es bietet sich in der Regel an, diese Variablen in ein Blockelement wie z.B. einen Absatz einzufügen, ggf. auch beispielsweise mit Zentrierung des Textes, falls der QR-Code mittig/zentriert auf der Seite angezeigt werden soll (statt linksbündig), also z.B.:

```
<p>Diese Seite auf anderem Gerät öffnen:</p>
<p style="text-align: center;">$QR</p>
```

Ein Beispiel zum Verlinken einer Kursressource "XYZ.pdf" sowohl per lokalem Downloadlink als auch per QR-Code:

```
<p>Download von <a
href="$WebAssignServer/$Veranstaltername/KursStartSeite/$KursNr/$VersionsNr/XYZ
.pdf">XYZ.pdf</a> auf mobilem Gerät:</p>
<p style="text-align: center;">
$QR($WebAssignServer/$Veranstaltername/KursStartSeite/$KursNr/$VersionsNr/XYZ.p
df)
</p>
```

Heft-Schließen-Knopf

Vorab: Was ist Heft-Schließen?

Wie im Abschnitt Grundlagen/Struktur einer Kursumgebung <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#struktur> in Teil I beschrieben, ist jede Aufgabe immer einem Aufgabenheft untergeordnet, das wiederum einen Bearbeitungszeitraum für alle Aufgaben, aus denen es besteht, festlegt. Nach Ablauf dieses Bearbeitungszeitraums können dann alle Einsendungen der Studenten ausgewertet werden. Dazu findet (ggf. automatisch, sofern in den Kursparametern aktiviert, andernfalls erst auf manuelle Aktion des Kursbetreuers hin) ein sog. Heft-Schließen statt, in dessen Rahmen auch die Korrekturdokumente erzeugt werden (durch Einfügen der eingesendeten Daten und ggf. automatisch erzeugter (Vor-)Korrekturen und Auswertungen in die vom Aufgabenautor erstellten Korrekturschablonen wie `korrektur1.1.html`, siehe: Die Standard-Kursressourcen zu einer Aufgabe).

Bei automatisch korrigierten Aufgaben findet die Autokorrektur direkt im Rahmen des Heft-Schließens statt, und das Ergebnis jeder Autokorrektur wird direkt mit in das jeweilige Korrekturdokument mit eingebunden.

Im Falle manuell zu korrigierender Aufgaben enthalten diese Korrekturdokumente (kurz als „Korrekturen“ bezeichnet) in der Regel nur die Eingaben des jeweiligen Studenten, ggf. angereichert um eine (schon bei der Einsendung des Studente erzeugte) automatische Vorkorrektur. Diese Korrekturen können dann im Anschluss ans Heft-Schließen den Korrekturkräften zugeteilt werden. Die Korrektoren bearbeiten dann die Korrekturen, indem sie sie mit Kommentaren anreichern und eine Bewertung festlegen.

Kurz: Das Heft-Schließen ist also Voraussetzung für die Korrektur.

Weiterhin können Studenten nach dem Heft-Schließen, wie der Name schon andeutet, auch nichts mehr zu diesem Aufgabenheft einsenden. Wenn das Heft-Schließen erst nach dem Einsendeschluss stattfindet, hat das keine praktischen Auswirkungen mehr, da ja nach Einsendeschluss ohnehin schon nicht mehr eingesendet werden kann. Es es aber auch möglich, Hefte schon vor dem Einsendeschluss zu schließen. Das entspricht praktisch einer vorzeitigen Abgabe: Ist das Heft eines

Studenten schon vor Einsendeschluss geschlossen (das ist wohlgemerkt für jeden Studenten individuell möglich), kann er nichts mehr einsenden, dafür aber kann sofort, also ebenfalls schon vor Einsendeschluss, mit der Korrektur begonnen werden.

Ein solches vorzeitiges Heft-Schließen (vorzeitige Abgabe und somit Ermöglichung früherer Korrektur) sollte nur vom Studenten vorgenommen werden. Zu diesem Zweck kann den Studenten ein entsprechender „Heft-Schließen-Knopf" im Kurs eingeblendet werden:

Heft-Schließen-Knopf für Studenten…

Die Einblendung eines Heft-Schließen-Knopfes für Studenten ergibt in erster Linie dann Sinn, *wenn Sie auch tatsächlich vorzeitige Korrekturen vorsehen*:

- Bei handbewerteten Aufgaben bewirkt ein vorzeitiges Heft-Schließen, dass die Einsendungen früher korrigiert werden können und der Student somit früher sein Ergebnis bekommt. Falls Sie jedoch grundsätzlich erst nach Einsendeschluss die Korrekturen auf die Korrekturkräfte verteilen, gibt es keinen Sinn, den Studenten einen Heft-Schließen-Knopf anzubieten (und zu erklären).
- Bei automatisch bewerteten Aufgaben bringt ein vorzeitiges Heft-Schließen durch den Studenten nur dann etwas, wenn die dabei erzeugte Autokorrektur dem Studenten dann auch schon vorzeitig, also vor Einsendeschluss angezeigt würde. Das kann in den Optionen zum Aufgabenheft auch so eingestellt werden.
    - Eine solche Einstellung ergibt in erster Linie für Selbstkontrollarbeits-Hefte Sinn (siehe dazu auch den nachfolgenden Unterabschnitt Heft-Öffnen-Knopf für Selbstkontrollarbeiten).
    - Für Einsendearbeiten, Prüfungen etc. ist die vorzeitige Freigabe von Autokorrekturen dagegen normalerweise *nicht sinnvoll*, denn sie brigt die Gefahr, dass Informationen zur Korrektur und damit zu den korrekten Lösungen vor Einsendeschluss in Umlauf kommen und anderen Studenten, deren Hefte noch offen sind, die Lösung verraten wird. Sinnvoller dürfte daher die Einstellung sein, dass auch schon fertige Korrekturen bis zum Einsendeschluss zurückgehalten werden. Dann bringt aber vorzeitiges Heft-Schließen bei autokorrigierten Aufgaben keinen Vorteil.

Ein *alternatives Einsatzszenario* könnte darin bestehen, dass Sie explizit eine abschließende Heftabgabe verpflichtend machen und von Studenten nicht selbst (über einen Heft-Schließen-Knopf) abgegebene Aufgabeneinsendungen gar nicht korrigieren möchten. Im Normalfall wird dies nicht empfohlen, aber wenn Sie so etwas umsetzen möchten, können Sie das wie folgt tun:

- In den Kursparametern…
    - den Heft-Schließen-Knopf für Studenten aktivieren (s.u.),
    - das automatische Heftschließen nach Einsendeschluss abschalten (was ja alle Hefte, die von Studenten nicht selbst geschlossen wurden, nachträglich schließen würde)
    - und dann die Option deaktivieren, dass den Studenten nach Abgabeschluss angezeigt werden soll, die Heftabgabe sei erfolgt. (Standardmäßig wird ja davon ausgegangen, dass das Heftschließen durch Studenten selbst nur zur vorzeitigen Abgabe dient und mit erreichen der Abgabefrist alle noch offenen Hefte implizit als abgegeben gelten – auch wenn der eigentliche Heft-Schließen-Prozess, der dann nur noch die Korrekturen generiert, s.u., noch nicht ausgeführt wurde.)
- Dann natürlich nach Bearbeitungsende auch nicht als Betreuer selbst alle noch offenen Hefte schließen, sondern diese vielmehr geöffnet lassen, so dass dazu eben keine Korrekturen erzeugt wurden.

Wenn Sie also eines der beiden obigen Szenarien nutzen und dazu den Studenten einen Heft-Schließen-Button zur vorzeitigen oder verpflichtenden endgültigen Abgabe anbieten möchten, haben Sie folgende Möglichkeiten:

… in der Aufgabenübersicht

Der Standardweg, den Studenten einen Heft-Schließen-Knopf einzublenden, besteht in der Aktivierung der Option *»Heft-Schließen-Knopf in Aufgabenübersicht für Studenten«* in den Kursparametern. Wenn diese Funktion aktiv ist, erscheint in der Aufgabentabelle der Studenten-Startseite (oder auch der `StudentenHeftStartSeite`[7]) ein Heft-Schließen-Knopf pro Aufgabenheft (nur innerhalb des jeweiligen Bearbeitungszeitraums), mit dem der Student die Bearbeitung des Hefts und aller Aufgaben darin vorzeitig abschließen kann.

… in einer Aufgabenseite

Nur in Ausnahmefällen kann es auch wünschenswert sein, einen Heft-Schließen-Knopf zur endgültigen Abgabe direkt mit in eine Aufgabenseite einzubinden:

- Insbesondere für Aufgabenhefte, die genau eine einzige Aufgabe enthalten, kann das praktisch sein, denn dann bezieht sich der Button ja im Endeffekt auf die endgültige Einreichung *genau* der in der Aufgabenseite angezeigten Eingaben. Besteht ein Aufgabenheft dagegen aus mehreren Aufgaben, ist das eher kritisch zu sehen, denn dann ist ggf. nicht sofort für den Studenten ersichtlich, dass diese endgültige Abgabe nicht nur die Bearbeitung der angezeigten Aufgabe abschließt, sondern auch aller anderen Aufgaben desselben Aufgabenhefts!
- Falls Sie Ihre Aufgaben per LTI in eine andere Lernumgebung (z.B. Moodle) einbinden <https://online-uebungssystem.fernuni-hagen.de/download/LTI_Moodle/Uebungssystem_LTI_Moodle.html> , bekommen die Studenten (zumindest die, die von dieser Lernumgebung aus arbeiten) nur die Aufgabenseiten, nicht aber die Aufgabenübersichtsseite des Übungssystem-Kurses zu sehen. Ein nur dort eingeblendeter Heft-Schließen-Knopf bliebe den Studenten der Lernumgebung also verborgen. Möchten Sie diesen zur Ermöglichung vorzeitiger endgültiger Abgabe dennoch einblenden, bleibt nur die Einbindung in die Aufgabenseiten.

Sollten Sie sich dazu entscheiden, können Sie in einer Aufgabenseite (wie `aufgabe1.1.html`) den Heft-Schließen-Knopf über folgende Variablen einbinden:

Variable	Bedeutung
`$HeftSchliessenKnopf`	Ein Knopf mit Standardbeschriftung (derzeit »Heft schließen«)
`$HeftSchliessenKnopf(…)`	Ein Knopf mit selbst gewählter Beschriftung, die Sie zwischen den Klammern an Stelle der drei Punkte … eingeben. Die Beschriftung darf selbst keine runden Klammern enthalten.
`$HeftSchliessenKnopf{…}`	Gleichbedeutende Alternative, hier dürfen runde Klammern in der Beschriftung auftreten, dafür keine geschweiften.
`$HeftschliessenKnopf(1)(2)`	Ein Knopf mit zwei selbst gewählten Beschriftungen: An Stelle `1` legen Sie die normale Buttonbeschriftung fest, die angezeigt wird, so lange das Heft noch nicht geschlossen ist. An der Stelle `2` legen Sie eine zweite Beschriftung fest, die angezeigt wird, sobald das Heft geschlossen wurde. (Der Button ist in diesem Fall bereits inaktiv und nicht mehr anklickbar.)
`$HeftschliessenKnopf{1}{2}`	Gleichbedeutende Alternative, falls runde Klammern in den Beschriftungen (oder einer davon) benutzt werden sollen.
`$HeftSchliessenKnopf(1)(2)(OK)`	Durch Anfügen eines dritten Klammernpaars können Sie zusätzlich ein Icon festlegen, das bei geschlossenem Heft vor dem Text `2` auf dem (inaktiven) Button angezeigt werden soll (s.u.)
`$HeftSchliessenKnopf{1}{2}{OK}`	Gleichbedeutende Alternative, falls runde Klammern in den Beschriftungen (oder einer davon) benutzt werden sollen.
`$HeftSchliessenKnopf(1)Oeffnen(3)`	Ein Knopf, der neben einer Beschriftung `1` für die Heft-Schließen-Funktion noch eine Beschriftung `3` für die optionale Heft-Öffnen-Funktion (s.u.) festlegt.
`$HeftSchliessenKnopf{1}Oeffnen{3}`	entsprechend, s.o.
Hinweis	**Zu den folgenden Beispielen geben wir die Varianten mit geschweiften Klammern nicht mehr jeweils separat an**, sondern nennen nur noch beispielhaft die Schreibweise mit runden Klammern.
`$HeftSchliessenKnopf(1)(2)Oeffnen(3)`	Variante mit drei eigenen Beschriftungen: `1` für den aktiven Heft-Schließen-Knopf, `3` für den Heft-Öffnen-Knopf für geschlossene Hefte vor Bearbeitungsende und `2` für den Text auf dem deaktivierten Knopf nach Bearbeitungsende.
`$HeftSchliessenKnopf(1)(2)(INFO)Oeffnen(3)`	„Maximale“ Variante mit drei Beschriftungstexten und noch einem Icon, das nach Bearbeitungsende auf dem dann deaktivierten Button stehen soll.
`$HeftSchliessenKnopfGlobal`	Während `$HeftSchliessenKnopf` einen Button erzeugt, der erst dann aktiviert wird / anklickbar ist, sobald *die Aufgabe*, in deren Seite er steht, auch vom Benutzer bearbeitet wurde, wird *diese* „heft-globale“ Variante immer dann aktiviert, sobald *das Aufgabenheft* bearbeitet wurde, genauer: sobald mindestens eine Aufgabe des Hefts bearbeitet wurde, nicht jedoch zwangsläufig die Aufgabe mit diesem Button.
`$HeftSchliessenKnopfGlobal…`	Diese Variable kann ebenfalls in allen oben stehenden Varianten verwendet werden, also auch z.B. als `$HeftSchliessenKnopfGlobal{1}{2}{OK}`

Hinweise zum Einsatz dieser Variablen:

- Der Heft-Schließen-Knopf in der Aufgabe wird für einen Studenten erst aktiv (anklickbar), sobald eine Einsendung des Studenten zu dieser Aufgabe vorliegt.
  - Ausnahme: Bei Verwendung des Zusatzes `Global` im Variablennamen wird der Button für einen Studenten aktiviert, sobald der Student mindestens eine Aufgabe des Aufgabenhefts bearbeitet hat, zu dem die Aufgabe mit diesem Button gehört. Falls also z.B. Ihr Aufgabenheft aus n Aufgaben besteht, Sie in der letzten Aufgabenseite diesen `$HeftSchliessenKnopfGlobal` eingefügt haben, so können die Teilnehmer diesen dann (bei Aufruf der letzten Aufgabenseite) anklicken, selbst wenn sie diese letzte Aufgabe gar nicht bearbeitet haben – lediglich solange sie noch *gar keine* Aufgabe bearbeitet haben, bleibt der Button gesperrt/inaktiv.
- Die Variable *darf nicht innerhalb eines HTML-Formulars (`<form ...>...</form>`) stehen*, also insbesondere auch nicht innerhalb des Aufgabenformulars, denn sie wird nicht nur durch ein einfaches HTML-`button`-Element ersetzt, sondern durch ein eigenes `form`, das einen `button` enthält. In der Regel muss diese Variable also hinter dem `</form>`-Tag des Aufgabenformulars eingefügt werden.
- Es wird empfohlen, dieser Variable einen Text voranzustellen, der die Bedeutung des Knopfes beschreibt. Insbesondere bei Aufgabenheften, die mehrere Aufgaben enthalten, sollten Button-Beschriftung und einleitender Text unmissverständlich klarstellen, dass damit nicht nur diese eine Aufgabe, sondern das gesamte Heft endgültig eingereicht wird.

Mögliche Icons

Für die Varianten mit drei Klammerpaaren (zwei Beschriftungen und ein Icon) stehen folgende vier Icons zur Auswahl:

Icon-Name	Beschreibung	derzeitige Darstellung
`OK`	Grünes Check-Icon	
`INFO`	Blaues Info-Icon	
`WARNING`	Warndreieck	
`ERROR`	Rotes X-Icon	

- Die Groß-/Kleinschreibung des Icon-Namens ist dabei egal.
- Die Abbildungen in obiger Tabelle („derzeitige Darstellung“) zeigen die Darstellung Stand Februar 2019. Bei Änderungen am Übungssystem-Webdesign könnte sich die konkrete Darstellung dieser Icons ändern.
- Außerdem ist zu beachten, dass diese Variablen mit Iconnamen nur im regulären Übungssystem-Webdesign verwendet werden sollten, also in Aufgabenseiten, die den Embedding-Mechanismus verwenden. Andernfalls könnte das Layout der Buttons mit Icon unschön aussehen bzw. müsste mit eigenem CSS formatiert werden.

Das Icon wird erst nach der Heftabgabe angezeigt. Welches der Icons dafür passend ist, hängt stark vom zweiten Beschriftungstext ab. In der Regel, z.B. bei einem Text wie „Heftabgabe erfolgreich“, würde eher ein OK- oder Info-Icon passen. Lautet der Text dagegen „Keine weitere Aufgabenbearbeitung mehr möglich, da die Abgabe bereits erfolgt ist!“, könnte auch ein Warnungs- oder vielleicht sogar Fehler-Icon gewünscht sein, je nachdem was Sie ausdrücken möchten. (Da die Heftabgabe keinen Fehler darstellt, ist das Fehler-Icon im Normalfall nicht empfohlen – wurde aber der Vollständigkeit halber mit zur Verfügung gestellt.)

Heft-Öffnen-Knopf für Selbstkontrollarbeiten

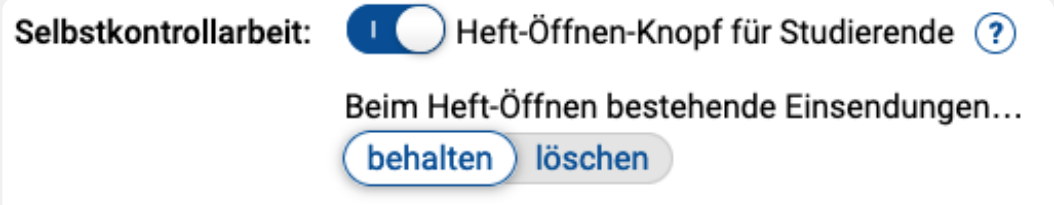
Normalerweise können Studierende ihr Heft höchstens schließen (sofern Sie ihnen überhaupt einen Heft-Schließen-Knopf anbieten). Das ist normalerweise final, wieder geöffnet werden kann ein Heft höchstens durch Betreuer.

Es gibt jedoch einen speziellen Sonderfall, in dem es wünschenswert sein kann, dass Studierende ihr bereits geschlossenes Heft auch selbst wieder öffnen können, nämlich *Aufgabenhefte, die eine Selbstkontrollarbeit umsetzen*:

Idealtypisch verstehen wir unter einem **Selbstkontrollarbeits-Heft** ein Aufgabenheft, das keinerlei manuell korrigierte Aufgaben enthält, sondern nur automatisch korrigierte Aufgaben, welche wiederum typischerweise nicht als „isolierte" Selbstkontrollaufgaben ihre Auswertung jeweils sofort bei Einsendung anzeigen (Sofort-Feedback), sondern sich bei offenem Heft zunächst wie die Aufgaben einer Einsendearbeit oder Prüfung (z.B. MC-Klausur) verhalten, d.h. lediglich die Eingaben quittieren, aber eben nicht sofort auswerten. Zu diesem Aufgabenheft werde aber ein Heft-Schließen-Knopf für Studierende angeboten, und die weiteren Hefteinstellungen seien so vorgenommen, dass die beim Heftschließen erzeugten Autokorrekturen und ggf. auch Musterlösungen dazu sofort sichtbar werden und nicht bis zum Bearbeitungsende zurückgehalten werden. (Das Bearbeitungsende wird bei solchen Heften typischerweise erst am Semesterende liegen.)

Das heißt: Wenn Studierende die Aufgaben eines solchen Hefts bearbeiten, unterscheidet sich das zunächst gar nicht von Einsendearbeiten, Klausuren etc.. Aber sobald sie das Heft selbst schließen, bekommen sie sofort das Ergebnis. Und wenn es sich bei den Aufgaben eben tatsächlich nur um Selbstkontrollaufgaben handelt, also um keine externe Leistungskontrolle wie z.B. Prüfungsleistungen, so dass die Auswertungsergebnisse nicht aufgehoben werden müssen, dann spricht nichts dagegen, dass die Teilnehmer ihren fertig ausgewerteten Versuch wieder zurückziehen und das Aufgabenheft im Anschluss erneut bearbeiten können. Falls z.B. die Auswertung des ersten Versuchs noch Wissenslücken aufgedeckt hat, könnten die Teilnehmer so zunächst den betroffenen Stoff vertiefen und im Anschluss die Selbstkontrollarbeit ein zweites Mal bearbeiten, um ihren Lernfortschritt zu überprüfen.

Genau dazu kann nun also (seit September 2021) in den Aufgabenhefteinstellungen eine Option aktiviert werden[8], dass den Studenten bei bereits geschlossenem Heft *vor Bearbeitungsende* an Stelle des Heft-Schließen-Knopfes nun ein Heft-Öffnen-Knopf angezeigt werden soll.
Dazu können Sie noch zusätzlich einstellen, ob dieser Heft-Öffnen-Knopf dann lediglich das Heft wieder öffnet, die letzte Einsendungen zu den Aufgaben aber erhalten bleiben und von den Teilnehmern/-innen anschließend einfach nachbearbeitet / verbessert werden können, *oder* ob beim Heft-Öffnen auch sämtliche Einsendungen des/der Teilnehmers/-in zu den Aufgaben des Hefts mit gelöscht, das Heft also komplett in einen unbearbeiteten Zustand zurückgesetzt werden soll.



Heft-Öffnen-Knopf-Optionen in der Aufgabenheftverwaltung

Falls Sie *randomisierte Aufgaben* in dem Heft einsetzen, werden nach dem Heft-Öffnen alle unbearbeiteten Aufgaben neu randomisiert. Gerade hier sollte dann also die Option zum Löschen vorheriger Einsendungen aktiviert werden, damit eben alle Aufgaben danach wieder unbearbeitet

sind und somit neu randomisiert werden. Auf diese Weise bekommen die Studierenden dann im zweiten Versuch nicht wieder exakt dieselben Fragen gestellt wie noch im ersten. Wenn Sie dagegen die Einsendungen beim Heft-Öffnen nicht löschen lassen, werden höchstens die Aufgaben neu randomisiert, die ein Teilnehmer vor dem ersten Heftschließen noch gar nicht bearbeitet hatte.

Wenn Sie die Heft-Öffnen-Knopf-Option für ein Aufgabenheft eingeschaltet haben, wird an jeder Stelle, wo normalerweise ein Heft-Schließen-Knopf für Studierende angezeigt wird, dieser durch einen Heft-Öffnen-Knopf ersetzt, sobald das Heft geschlossen wurde und so lange das Bearbeitungsende noch nicht verstrichen ist. Nach Ablauf des Bearbeitungsendes wird wieder ganz normal ein deaktivierter, nicht anklickbarer Button mit einer Beschriftung wie z.B. »Heftabgabe erfolgt« angezeigt, siehe oben.

Die *Standardbeschriftung* für den Heft-Öffnen-Knopf lautet derzeit »Heft öffnen«, falls die Einsendungen dabei erhalten bleiben, bzw. »Heft zurücksetzen«, falls die Einstellung gewählt wurde, dass dieser Button das Heft nicht nur wieder öffnet, sondern dabei auch alle Einsendungen löscht.

Falls Sie nun – wie oben beschrieben – eine `$HeftSchliessenKnopf`-Variable verwenden *und* die Heft-Öffnen-Knopf-Funktion aktiviert haben (wobei diese Variable dann in einer automatisch bewerteten Aufgabe oder einer unbewerteten Aufgabe stehen müsste, denn sobald handbewertete Aufgaben im Heft vorkommen, steht die Heft-Öffnen-Funktion ja wie gesagt nicht mehr zur Verfügung), und falls Sie *eine eigene Beschriftung* für den Heft-Öffnen-Knopf festlegen möchten, ergänzen Sie die Variable um einen Zusatz der Art `Oeffnen{Beschriftungstext}`.

Dieser Beschriftungstext wird immer dann auf dem Button sichtbar, wenn dieser das Heft wieder öffnen würde, wenn also die Heft-Öffnen-Knopf-Option fürs Aufgabenheft überhaupt eingeschaltet wurde und wenn der Student sein Heft geschlossen hat und das Bearbeitungsende noch nicht verstrichen ist.

Die Angabe ist wie gesagt optional, wenn Sie sie weglassen, wird »Heft öffnen« bzw. »Heft zurücksetzen« als Beschriftung für den Heft-Öffnen-Knopf verwendet, je nachdem, ob die Einsendungen dabei erhalten bleiben oder gelöscht werden. Wenn Sie wie hier beschrieben Ihre *eigene* Beschriftung angeben, so ist diese unabhängig von der Einstellung, was mit den Einsendungen beim Heft-Öffnen passieren soll.

Falls Sie neben der Beschriftung für den aktiven (anklickbaren) Schließen-Knopf und für den Öffnen-Knopf auch noch (im zweiten Klammernpaar) einen Beschriftungstext für geschlossene Hefte, ggf. auch noch mit Icon, angeben, so ist dieser dann nicht mehr sofort nach Heft-Schließen sichtbar, sondern erst nach Bearbeitungsende (denn vor Ende nach Schließen ist ja dann der Heft-Öffnen-Knopf statt dessen vorhanden).

Beispiel:

```
$HeftSchliessenKnopfGlobal{Heft auswerten}{Frist verstrichen}
{ERROR}Oeffnen{Heft neu beginnen}
```

Dieses Bespiel zeigt einen Heft-Schließen-Knopf, wie er in einer Aufgabenseite (einer automatisch bewerteten oder unbewerteten Aufgabe) vorkommen kann, der benutzbar ist, sobald zu irgendeiner Aufgabe des Hefts eine Einsendung vorliegt (nicht unbedingt zu der Aufgabe, in deren Seite er steht), und der bei noch offenem Heft die Beschriftung »Heft auswerten« trägt, nach Heft-Schließen (vor Bearbeitungsende) die Beschriftung »Heft neu beginnen« (sinnvoll z.B. bei der Option, dass der Button dann auch die Einsendungen löscht). Nach Ablauf der Bearbeitungszeit wird der Button dann ausgegraut sein und den Text »Frist verstrichen« mit einem roten Fehler-Icon (s.o.) anzeigen – egal ob das Heft da gerade offen oder geschlossen ist.

Verlinken von Dateien (Kursressourcen)

Ihre Aufgabenstellung kann prinzipiell vollständig in HTML gehalten bleiben. Sie können aber natürlich auch z.B. Bilder einbinden oder Dateien (wie komplexe PDF-Dokumente zur Aufgabenstellung) zum Download anbieten.

Die zu verlinkenden Dateien bzw. einzubindenden Grafiken sind zunächst als Kursressourcen <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#kr> im Übungssystem-Kurs zu hinterlegen. Auf deren allgemeine Einbindung wurde bereits im einleitenden Abschnitt Einbindung von Kursressourcen in der fortgeschrittenen Aufgabenerstellung <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#kr_fga> eingegangen. Für Dateien, die zur Aufgabenstellung gehören und nicht bereits vor Freischaltung von Aufgaben abrufbar sein sollen, bzw. für Dateien, die Teil einer Musterlösung sind und nicht bereits vor Freischaltung der Musterlösungen einzeln herunterladbar sein sollen, sollte außerdem die Taktung <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#taktung> berücksichtigt werden.

Ein ungetakteter Link hat z.B. in der Regel folgenden Aufbau:

```
<a
href="$WebAssignServer/$Veranstaltername/KursStartSeite/$KursNr/$VersionsNr/Dat
einame.xxx">Linktext</a>
```

Durch hinzufügen des Attributs `target="_blank"` kann auch eingestellt werden, dass die Datei in einem neuen Tab oder Fenster geöffnet werden und die Aufgabenseite offen bleiben soll. (Das ist ggf. sinnvoll, wenn der Link zu einer im Browser darstellbaren, also nicht grundsätzlich herunterzuladenden Datei führt.)

Getaktete Kursressourcen zu z.B. Aufgabe 2.3 haben einen Dateinamen, der mit `aufgabe2.3.` beginnt, und werden typischerweise wie folgt verlinkt:

```
<a
href="$WebAssignServer/$Veranstaltername/Aufgabentext/$KursNr/$VersionsNr/$Aufg
abenheftNr/$AufgabenNr/aufgabe$AufgabenheftNr.$AufgabenNr.…)">Linktext</a>
```

Ausführlicher geht die Online-Hilfeseite darauf ein, die Sie in Ihrem Kurs im Kursressourcen-Verzeichnis unter dem Hilfelink *»Weiterführende Informationen zu Standard- und Nicht-Standard-Kursressourcen, Zugriffsschutz, Taktung, Einbindung in HTML-Seiten etc.«* finden. Bei Auswahl einer Datei in Ihrer Kursressourcen-Liste wird Ihnen außerdem bereits der passende URL für einen Link (oder auch ein Image-Tag) generiert, so dass sie ihn in Ihr HTML-Dokument kopieren können.

Individuelle Dateien für bekannte Studenten verlinken

Es ist im Online-Übungssystem auch möglich, dass Sie (für eine teilnehmerbeschränkte Kursumgebung, deren Teilnehmer Ihnen im Vorfeld bekannt sind) individuelle Dateien erstellen und verlinken können. So kann z.B. jeder Teilnehmer an einer bestimmten Aufgabe eine eigene PDF-Datei mit individuell an ihn vergebener Aufgabenstellung zum Download angeboten bekommen. (Siehe Studenten-individuelle Ressourcen <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#krindiv> .)

Dieses Feature lässt sich wie folgt realisieren:

1. Erstellen Sie die benötigten Dateien, wobei der Dateinamen jeder individuell zugeschnittenen Datei die Matrikelnummer des Teilnehmers (oder den Loginnamen bei Mentoren- oder Teststudenten-Accounts) enthält.
    - Beachten Sie bei der Benennung der Datei auch die Hinweise zumZugriffsschutz <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#krindivzugriffsschutz> .
2. Erstellen Sie außerdem eine „Fallback-Datei“, die denjenigen Teilnehmern angezeigt wird, für die keine individuelle Datei in den Kursressourcen existiert. (Das kann z.B. eine allgemeine Aufgabenversion sein, oder – falls dieser Fall nicht vorkommen dürfte, sondern auf einen Fehler wie einen Tippfehler in einer Matrikelnummer im Dateinamen oder einen vergessenen Upload etc. zurückzuführen wäre – ein Dokument mit einer entsprechenden Fehlermeldung und Kontaktmöglichkeiten zur Klärung des Sachverhalts.)
3. Fügen Sie im HTML einen Downloadlink ein, wobei Sie auf dieKontrollvariable `$Resource(…$MatrikelNr…)orElse(…)` zurückgreifen.

Beispiel

Nehmen wir an, Sie möchten zu Aufgabe 1 in Aufgabenheft 1 individuelle PDF-Dateien in Ihrer Aufgabe verlinken.

Dann könnten Sie die PDF-Dateien wie folgt nennen:

- `aufgabe1.1.1234567.pdf` für eine individuelle Datei für den Studenten mit Matrikelnummer 1234567.
- `aufgabe1.1.fallback.pdf` für die Fallback-Datei, die angezeigt wird, falls eine individuelle Datei für den Studenten nicht gefunden wird.

Die Verlinkung im Aufgaben-HTML sähe dann z.B. wie folgt aus:

```
<a
href="$WebAssignServer/$Veranstaltername/Aufgabentext/$KursNr/$VersionsNr/$Aufg
abenheftNr/$AufgabenNr/$Resource(aufgabe$AufgabenheftNr.$AufgabenNr.$MatrikelNr
.pdf)orElse(aufgabe$AufgabenheftNr.$AufgabenNr.fallback.pdf)">Individuellen
Aufgabentext hier herunterladen</a>
```

Das HTML-Formular

Der zweite wesentliche Teil einer Aufgabenseite (neben der Aufgabenstellung) ist das Webformular, über das die Studenten ihre Lösungen einsenden können.

Zu diesem Zweck wird *im Normalfall genau ein*[9] `form`-HTML-Element in die Seite eingebaut.

`form`-Tag

Damit über das Formular eine Einsendung möglich ist und alle Eingaben gespeichert werden können, muss das öffnende `form`-Tag bestimmte Regeln einhalten:

- Das Attribut `method` *muss* den Wert `POST` haben (andere Methoden sind zur Einsendung nicht möglich).
- Für einfache Formulare ohne Datei-Uploads:

    - Das Attribut `action` muss folgenden URL enthalten: `$WebAssignServer/$Veranstaltername/Einsendung/$KursNr/$VersionsNr/$AufgabenheftNr/$AufgabenNr/`
- Für Multipart-Formulare (insb. alle Formulare, die mindestens einen Fileupload enthalten):
    - Das Attribut `action` muss folgenden URL enthalen: `$WebAssignServer/$Veranstaltername/EinsendungMultipart/$KursNr/$VersionsNr/$AufgabenheftNr/$AufgabenNr/`
    - Das Attribut `enctype` muss angegeben werden und den Wert `multipart/form-data` enthalten.
- **Wichtig:** Fügen Sie *kein* `accept-charset`-Attribut zum `form`-Tag hinzu! Das Übungssystem fügt automatisch ein solches Attribut ein und deklariert darin genau das Charset, in dem die Formulareingaben vom Übungssystem auch dekodiert werden (derzeit UTF-8).
- Hinweis: Der abschließende Slash hinter `$AufgabenNr` ist optional. Früher hat das Übungssystem diesen nicht hinzugefügt, inzwischen wird er aber empfohlen. Dieser spielt jedoch nur dann eine Rolle, wenn Sie in der Quittungsseite eine getaktete Aufgabenressource mit relativem URL einbinden möchten, also z.B. `<img src="aufgabe$AufgabenheftNr.$AufgabenNr.abb1.png">` oder ähnlich. Solche Ressourcen werden in der Quittung nur korrekt angezeigt, wenn der Browser sie unter einem URL nachlädt, der auch auf Aufgabennummer vor dem Dateinamen enthält. Und wenn Sie den Action-URL nicht mit einem Slash abschließen, würde der Browser diesen Dateinamen nicht hinter der Aufgabennummer anhängen, sondern vielmehr die Aufgabennummer im Action-URL durch den Dateinamen ersetzen. (Ungetaktete Kursressourcen lassen sich auch in letzterem Fall fehlerfrei abrufen.)

D.h. im Allgemeinen sieht ein Form-Tag für „normale“ Aufgabenformulare (ohne Dateiuploads) wie folgt aus:

```
<form method="POST"
action="$WebAssignServer/$Veranstaltername/Einsendung/$KursNr/$VersionsNr/$AufgabenheftNr/$AufgabenNr/">
```

Dagegen sieht ein Form-Tag für ein Multipart-Formular (z.B. mit Dateiuploads) i.d.R. wie folgt aus:

```
<form method="POST"
action="$WebAssignServer/$Veranstaltername/EinsendungMultipart/$KursNr/$VersionsNr/$AufgabenheftNr/$AufgabenNr/"
enctype="multipart/form-data">
```

Submit-Button(s)

Damit ein Student etwas einsenden kann, muss – neben den Eingabefeldern, auf die der nachfolgende Abschnitt eingeht – ein Submit-Button zum Absenden der Eingaben (innerhalb des `form`-Elements) vorgesehen werden. Falls Sie Ihre Aufgabe in mehrere „technische“ Teilaufgaben <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#ta> zerlegen, deren Eingaben getrennt voneinander einzusenden sind, wird ein Submit-Button pro Teilaufgabe benötigt.

Wie genau der Button aussieht, beschriftet ist oder wo im Formular er angeordnet wird, liegt voll

und ganz in Ihrem Gestaltungsspielraum. Lediglich der *Name* des Buttons muss einer Konvention folgen, damit das Übungssystem auswerten kann, *welcher* Submit-Button gedrückt wurde.

Das `name`-Attribut des Button-Tags muss folgenden Aufbau haben:

```
einsenden<Teilaufgabe>
```

Dabei ist `<Teilaufgabe>` durch einen Großbuchstaben zu ersetzen: `A` für die erste (ggf. einzige) Teilaufgabe, `B` für die zweite Teilaufgabe etc.

Obiges Beispiellisting aus Abschnitt Generieren und Bearbeiten einer Seitenvorlage enthält z.B. nur eine Teilaufgabe und damit genau einen Submit-Button:

```
<input type="submit" name="einsendenA" value="Beschriftungstext">
```

oder in der neueren HTML5-Syntax (die nicht nur einfachen Text als Beschriftung erlaubt):

```
<button type="submit" name="einsendenA">Beschriftung</button>
```

Spezielle Submit-Buttons

Die folgenden beiden Submit-Buttons können *ausschließlich in Multipart-Formularen* (Servicename `EinsendungMultipart` und `enctype="multipart/form-data"`, siehe oben) verwendet werden!

Außerdem sollte die Teilaufgabe ausschließlich Dateiupload-Felder (`<input type="file"…`) umfassen, diese Buttons sind nicht für den Einsatz mit anderen Formularfeldern wie Text, Checkboxen etc. vorgesehen.

Dateiupload-Felder werden zu „Multi-Uploadfeldern“, indem der zu ihrer Teilaufgabe (also zum Upload der Dateien verwendete) Submit-Button ein `name`-Attribut des folgenden Aufbaus bekommt:

```
einsendenPlus<Teilaufgabe>
```

Das bewirkt, dass jede Dateieinsendung zur Teilaufgabe nicht – wie normal – die vorherige überschreibt, sondern mit dieser zusammengefasst wird, sie also ergänzt. Das wird weiter unten im Abschnitt Multi-Dateiuploads noch genauer beschrieben.

Insbesondere im Kontext solcher Multi-Dateiuploads sollte, um dennoch frühere Einsendungen wieder zurückziehen zu können, zusätzlich ein Button des folgenden Namens angeboten werden:

```
einsendenLeer<Teilaufgabe>
```

Ein solcher Button bewirkt zunächst, wie der „normale“ `einsenden<Teilaufgabe>`-Button auch, eine Einsendung zu jedem Formularfeld der Teilaufgabe, wobei jede frühere Einsendung in diesen Feldern überschrieben wird. Im Unterschied zur normalen Einsenden-Button werden jedoch die Eingaben in den Formularfeldern ignoriert, es werden zu allen Formularfeldern grundsätzlich leere Einsendungen erzeugt, so als ob die Formularfelder nicht ausgefüllt wären. Damit kann dieser Button

explizit zum Löschen vorheriger Einsendungen genutzt werden.

Eingabefelder

Damit ein Submit-Button überhaupt Daten einsenden kann, muss das Formular natürlich weiterhin Eingabefelder (`input`, `textarea`) definieren. Wie beim Submit-Button ist lediglich die *Benennung der Felder* vorgegeben. Diese hat nach folgendem Schema zu erfolgen:

```
Feld<Teilaufgabe><FeldNr>
```

Alle Felder, deren Inhalte mit dem Button `einsendenA` eingesendet werden sollen, müssen diesem durch Einsetzen desselben Teilaufgaben-Buchstabens `A` zugeordnet werden (analog für alle weiteren Teilaufgaben bzw. deren Submit-Buttons). Zur Komplettierung der Feld-ID werden alle Eingabefelder derselben Teilaufgabe durchnummeriert, und zwar jeweils lückenlos beginnend bei `1`. Das erste Eingabefeld zu Teilaufgabe `A` heißt demnach `FeldA1`, das erste Feld zur zweiten Teilaufgabe (`B`) heißt `FeldB1`, das zwölfte Eingabefeld zur dritten Teilaufgabe heißt `FeldC12` etc.

Wie bei den Variablen ist auch hier die *Groß-/Kleinschreibung zu beachten!*

Ein Beispiel für eine Textarea finden Sie bereits im obigen Listing (Abschnitt Generieren und Bearbeiten einer Seitenvorlage). Dieses generierte Aufgabenformular enthält eine Plaintext-Textarea. Falls die Studenten (HTML-)formatierten Text eingeben können sollen, können Sie die Aufgabendatei um eine entsprechende Konfiguration erweitern, und das Übungssystem wird entsprechende JavaScripte für einen WYSIWYG-Editor automatisch einbinden. Details dazu finden Sie im Handbuch zur Integration von WYSIWYG-Editoren <https://online-uebungssystem.fernuni-hagen.de/download/WYSIWYG_IBK/WYSIWYG_IBK.html> .

Nehmen wir aber als **begleitendes Fallbeispiel** an, Sie streben ein ganz anderes Aufgabenformular an und fügen das nachfolgende Beispielformular in die generierte HTML-Datei ein. Hierbei handelt es sich um ein Mulitpart-Formular mit zwei Teilaufgaben (eine für einen Dateiupload und eine für alle weiteren Eingaben) und ein paar im Folgenden noch zu erklärenden Besonderheiten:

```
<form method="POST"

action="$WebAssignServer/$Veranstaltername/EinsendungMultipart/$KursNr/$Version
sNr/$AufgabenheftNr/$AufgabenNr/"
      enctype="multipart/form-data"
      class="validation-styles">

<label for="t1">Wie lautet die Antwort auf die ultimative Frage nach dem Leben,
dem Universum und dem ganzen Rest?</label>
<input type="number" inputmode="numeric" min="0" max="999999" id="t1"
name="FeldA1" data-if-empty="Keine Angabe" data-ignore="empty">
<br>
<input type="checkbox" name="FeldA2" value="hello" id="cb1" data-if-empty="---"
data-ignore="never"> <label for="cb1">Hallo!</label><br>
<input type="checkbox" name="FeldA2" value="world" id="cb2"> <label
for="cb2">Welt</label>

<p><input type="submit" name="einsendenA" value="Eingaben einsenden"></p>

<p>PDF hochladen: <input type="file" name="FeldB1"
accept=".pdf,application/pdf,application/x-pdf" data-if-empty="Keine Datei
eingesendet!" data-ignore="empty">
<input type="submit" name="einsendenB" value="PDF einsenden"><br>
$IfExistsB1(Ihre zuletzt hochgeladene Datei: )$FeldB1HREF</p>

</form>
```

Zunächst demonstriert das Beispiel das Auftreten zweier Teilaufgaben `A` und `B` und die freie Positionierung und Beschriftung der Submit-Buttons.

Die `id`-Attribute der verschiedenen Eingabefelder spielen fürs Übungssystem selbst keine Rolle, sondern werden nur verwendet, um Labels zu den Feldern zuzuordnen (so dass z.B. beim Klick auf das Label »Hallo« die Checkbox links daneben ihren Zustand verändert, also der Nutzer nicht exakt auf die kleine Box klicken muss, oder dass beim Klick auf das Label »Wie lautet die Antwort […]« direkt der Cursor in die zugehörige Eingabezeile gesetzt wird).

Die Klasse `validation-styles` im Formular sowie das Attribute `type="number"` mit den weiteren Attribuen `min` und `max` im Input-Element konfigurieren Funktionen zur *Eingabevalidierung*. In diesem Fall wird damit ausgedrückt, dass die Eingabezeile nur bis zu 6 Ziffern aufnehmen darf. Ein dazugehöriges JavaScript, das ins. für eine unterschiedliche Darstellung fehlerhaft ausgefüllter Felder vor bzw. nach einem ersten Einsendeversuch sorgt, wird vom Übungssystem dynamisch in die Aufgabenseite injiziert, Sie müssen es also nicht selbst einbinden. Details finden sich in einer gesonderten Dokumentation zur Formular-Eingabevalidierung <https://online-uebungssystem.fernuni-hagen.de/download/JSFormValidation/FormularValidierung.html> .

Das Attribut `data-if-empty` in einem `input`- oder `textarea`-Tag gibt einen Text vor, der in Quittungs- und Korrekturseite angezeigt werden soll, falls der Student das Feld leer lässt. Ohne eine solche Angabe erfolgt eine Standard-Anzeige für leere Einsendungen, derzeit `[leer]`. Im Falle von mehreren Feldern desselben Namens (typisch für Checkboxen) muss das `data-if-empty`-Attribut, falls überhaupt angegeben, im ersten der gleichnamigen Eingabefelder stehen.

Das zweite Data-Attribut in obigem Beispiel (`data-ignore`) wird im Anschluss in einem eigenen Unterabschnitt genauer erläutert.

Die beiden Checkboxen in obigem Beispiel sollen den gerade schon erwähnten Sonderfall demonstrieren, dass mehrere Eingabefelder mit demselben Namen existieren dürfen (sofern jedes davon einen String einsendet, also keines ein Dateiupload-Feld ist). Deren Werte werden dann durch Komma getrennt aufgezählt, d.h. der Wert der Eingabe in `FeldA2` in obigem Beispiel ist entweder…

- `---`[10], wenn keine Checkbox angekreuzt ist,
- `hello`, wenn nur die Checkbox »Hallo!« angekreuzt ist,
- `world`, wenn nur die Checkbox »Welt« angekreuzt ist oder
- `hello,world`, wenn beide Checkboxen markiert sind.

Der letzte Absatz in obigem Beispiel demonstriert eine *typische Integration eines Dateiupload-Feldes*. Zunächst wird eine explizite Anzeige der letzten Einsendung (falls vorhanden) eingebunden, vgl. dazu den obigen Abschnitt zu Variablen für Einsendungen in Aufgabenformularen. Weiterhin wird hier das optionale `accept`-Attribut zur Beschränkung des Uploads auf bestimmte Dateitypen demonstriert, vgl. gesondertes Handbuch zu Dateityp-Einschränkungen <https://online-uebungssystem.fernuni-hagen.de/download/FileUploadAccept/FileUploadAccept.html> . Auf Dateiupload-Felder wird in nachfolgenden Abschnitten, beginnend ab Einfache Dateiuploads, genauer eingegangen.

Zahleneingabefelder

Eingabefelder mit `type="text"` lassen beliebige Texteingaben zu und können somit auch zur Eingabe von Zahlen genutzt werden.

Wenn Sie aber in einem solchen Feld explizit nach Zahlen Fragen und Eingaben anderen Textes gar nicht zulassen möchten, haben Sie verschiedene Möglichkeiten:

1. Die ältere Möglichkeit, die spezifisch vom Online-Übungssystem geboten wird, ist ein Textfeld mit zusätzlicher Javascript-basierter Eingabevalidierung (siehe separates Handbuch <https://online-uebungssystem.fernuni-hagen.de/download/JSFormValidation/FormularValidierung.html>).
   Diese Library ist aber inzwischen als abgekündigt/deprecated eingestuft. Sie wird aus Bestandsschutzgründen weiterhin unterstützt, aber für die Erstellung neuer Aufgabenseiten nicht mehr empfohlen.
2. In HTML 5 gibt es nun eine genormte Art, Input-Felder als Zahlenfelder auszuzeichnen, wozu die Input-Art zunächst über das Attribut `type="number"` definiert wird. Weitere Attribute legen Constraints wie Minimum (`min`) oder Maximum (`max`) fest, sowie eine Schrittweite (`step`, Voreinstellung ist 1, so dass genau alle ganzen Zahlen eingegeben werden können, während eine Schrittweite 5 z.B. nur Fünferschritte ermöglicht – also 0, 5, 10, 15, 20, … bzw. in Kombination mit einem Minimum wie 2 z.B. auch Abfolgen wie 2, 7, 12, 17, … –, und Schrittweiten kleiner als 1 erlauben auch Fließkommazahlen – bei Schrittweite 0.01 z.B. mit maximal zwei Nachkommastellen, die spezielle Schrittweite `step="any"` erlaubt beliebig viele Nachkommastellen). Außerdem kann über das `inputmode`-Attribut z.B. eine bestimmte Bildschirmtastatur auf Touchscreen wie insb. Smartphones angefordert werden.
   - Auch auf diese Möglichkeiten sowie die im Online-Übungssystem speziell für diese HTML-5-Number-Inputs geschaffenen weiteren Möglichkeiten geht ebenfalls das Handbuch zur Eingabevalidierung <https://online-uebungssystem.fernuni-hagen.de/download/JSFormValidation/FormularValidierung.html> genauer ein.
   - Eine ausführliche Dokumentation speziell zum Number-Imput findet sich z.B. bei MDN <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Input/number> .

Im begleitenden Fallbeispiel oben wurde z.B. für die Frage nach der Antwort auf die ultimative Frage

ein Number-Input eingefügt, das die Eingabe von genau sechsstelligen natürlichen Zahlen (alle ganzen Zahlen zwischen 0 und 999999) ermöglicht. Im auf Mausbedienung ausgelegten Desktop-Browser wird dieses typischerweise zwei „Spinner-Buttons", also kleine Pfeilsymbole auf/ab zum Erhöhen bzw. Erniedrigen des gerade eingegebenen Wertes in Einerschritten einblenden. Der `inputmode="numeric"` aktiviert z.B. auf Smartphones eine Bildschirmtastatur ähnlich der eines Telefons oder Taschenrechners: Ein Ziffernblock, typischerweise ganz ohne weitere Tasten wie Komma, Minus etc. – auch wenn das vom jeweiligen Endgerät bzw. Betriebssystem oder Browser abhängt: Android z.B. bietet gewöhnlich mehr Tasten zusätzlich zu Zifferntasten an als iOS es tut.

Weiterhin wurde das Formular im Fallbeispiel mit der Klasse `validation-styles` ausgezeichnet, was die oben erwähnten Übungssystem-spezifischen Stile und Funktionen zur Eingabevalidierung aktiviert. Das bewirkt z.B., dass bei einer Fehleingabe beim ersten Ausfüllen des Formulars die Formatierung noch recht dezent auf den Fehler hinweist (derzeit durch eine rote Umrandung), aber nach einem ersten (aufgrund fehlerhafter Eingaben) fehlgeschlagenen Einsendeversuch die nicht korrekt ausgefüllten Felder (hier das „Antwort"-Zahlenfeld) auffälliger (derzeit durch Nachstellen eines roten Warndreieck-Icons) hervorgehoben werden.

Textareas

Neben `input`-Elementen können Sie natürlich auch `textarea`-Elemente zur Eingabe längerer Texte in Ihre Aufgabenseite aufnehmen. Im Normalfall dienen diese zur Eingabe von mehrzeiligem, unformatiertem **Plaintext**.

Im einfachsten Fall sieht eine solche Einbindung wie folgt aus:

```
<textarea name="FeldA1" rows="10" cols="100"></textarea>
```

Eventuell zwischen den Tags stehender Inhalt wird bei Studierenden, die die Aufgabe noch nie bearbeitet haben, als Default in die Textarea eingetragen und kann von diesen modifiziert werden. Liegen bereits Einsendungen vor, wird aber statt dessen stets die letzte Einsendung dort eingetragen. (Analog zu `value`-Attributen von Input-Tags.)

Die Texteingaben in Textareas werden im Regelfall nicht als HTML-, sondern als Plaintext interpretiert, allerdings können (und müssen) Sie das im Endeffekt selbst entscheiden: Die Einbindung der Einsendung in die Quittungs- und Korrekturseitenvorlagen entscheidet darüber, ob die Einsendung unverändert in die ausgegebene HTML-Seite eingefügt wird (als als HTML interpretiert wird) oder ob darin vorkommende Zeichen wie `<` und `>` durch HTML-Entities wie `<` bzw. `>` zu ersetzen und damit 1:1 den Studierenden und Korrektoren anzuzeigen sind, die Einsendungen also als Plain Text zu interpretieren sind. In letzterem Fall verwenden Sie dazu `$Feld`-Variablen mit `P`-Suffix (wie „Plain"). Siehe dazu: Feldvariablen. (Dasselbe gilt streng genommen nicht nur für Texteingaben in Textareas, sondern auch für Eingaben in einfache Text-Inputs.)

Im Regelfall ist weiterhin davon auszugehen, dass die Eingaben nicht nur Plain Text sein sollen, sondern als solcher auch erstens in einer Proportionalschrift angezeigt und bei Überlänge auch mit automatischem Zeilenumbruch an die Seitenbreite angepasst werden sollen. Auch hier gilt: Für die Darstellung in Quittungs- oder Korrekturseiten ist das gesondert zu regeln. An dieser Stelle soll aber auf die Möglichkeiten eingegangen werden, wie konkret das Aussehen der Textareas selbst an diese Anforderungen angepasst werden kann.

Das Online-Übungssystem bietet dazu – zumindest sofern Sie Ihre Webseiten im Standarddesign des Übungssystems anbieten, typischerweise durch Nutzung des Embedding-Mechanismus, siehe Webdesign / Embedding – folgende CSS-Klassen an, die Sie Ihren Textareas zuordnen können:

class	Beschreibung
`fullwidth`	Die Textarea wird sich in der Breite an den verfügbaren Platz in der Seite anpassen, die Angabe aus dem `cols`-Attribut wirkt dann nur noch als Fallback, wenn das CSS nicht geladen wurde.
`hardwrap`	Wenn diese Klasse *nicht* angegeben wird, werden Texteingaben in der Textarea mit Überbreite (Zeilen, die länger als die Breite der Textarea sind) automatisch umgebrochen. Mit dieser Klasse können Sie das verhindern. Insbesondere für Textareas zur Quellcodeeingabe o.ä. ist das sinnvoll, und insbesondere dann, wenn wirklich nur dort eine neue Zeile begonnen werden soll, wo der Einsender/die Einsenderin die Return-Taste gedrückt hat.
`proportional`	Erzwingt die Verwendung einer Proportionalschrift wie in anderen Eingabefeldern auch.

Neben Plaintext können Sie, wie bereits angedeutet, natürlich auch HTML-Einsendungen entgegennehmen. Und so lange Sie nicht erwarten, dass die Studierenden selbst HTML-Code schreiben und einsenden sollen, sondern diese einfach eine Möglichkeit bekommen sollen, ihre Texteingaben in einem **WYSIWYG-Editor für formatierten Text** vornehmen können sollen, der daraus dann HTML-Code generiert und einsendet, so fügen Sie ebenfalls einfach nur Textareas (optional mit `class="fullwidth"`) ein und aktivieren Sie über einen Kommentar im HTML-Kopf die WYSIWYG-Editor-Integration, die alle oder bestimmte Textareas beim Laden der Seite automatisch durch solche Editoren ersetzt. Das ist in einem separaten Handbuch <https://online-uebungssystem.fernuni-hagen.de/download/WYSIWYG_IBK/WYSIWYG_IBK.html> ausführlich beschrieben.

Formeleditor

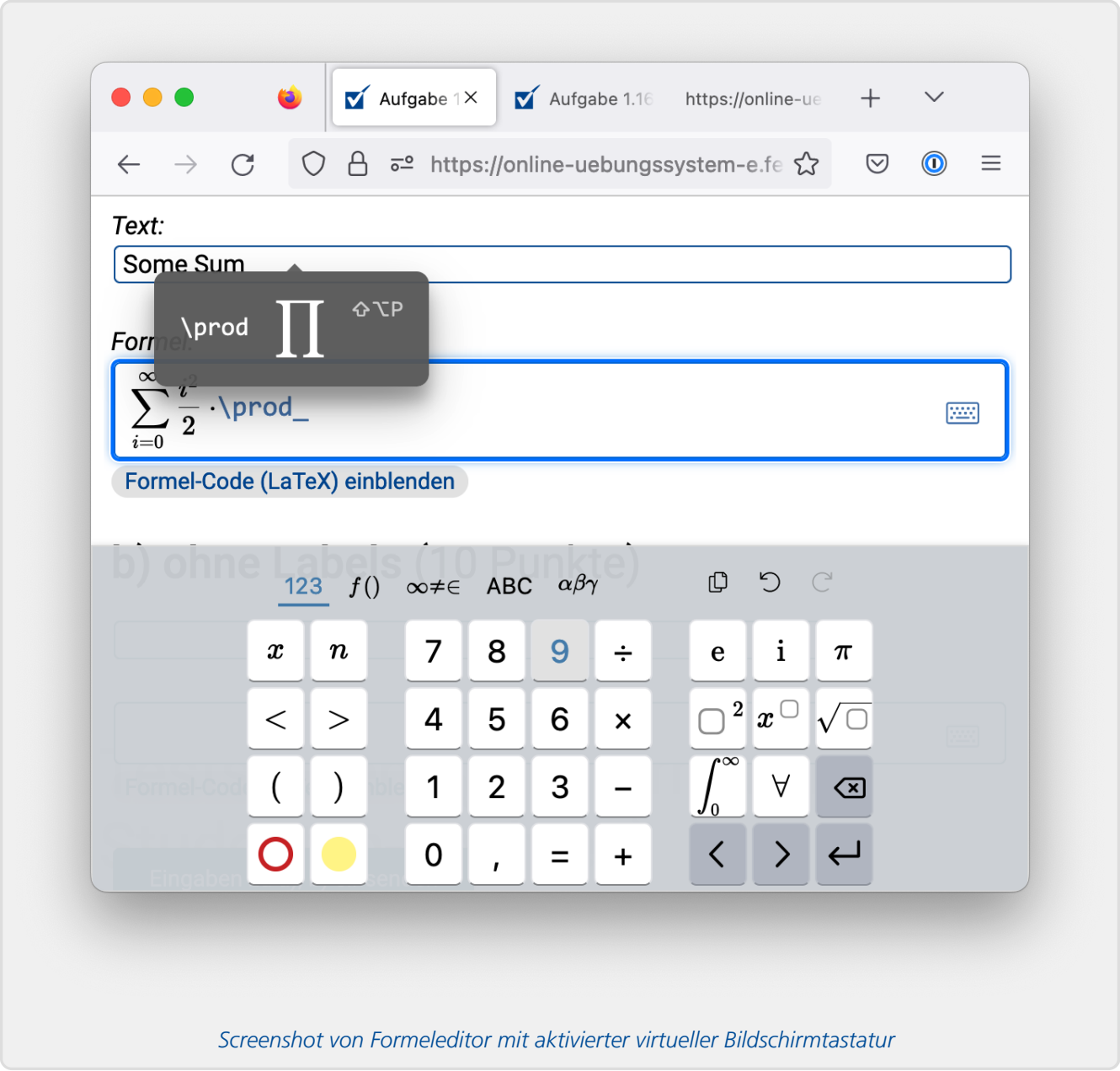
Seit Januar 2023 bietet das Online-Übungssystem nicht mehr nur eine Möglichkeit zur *Anzeige* (Rendering) von in LaTeX notierten Formeln (vgl. TeX-Integrationshandbuch <https://online-uebungssystem.fernuni-hagen.de/download/TeXIntegration/TeXIntegration.html>), sondern auch einen Formeleditor zur assistierten *Eingabe* solcher Formeln.

Diese Eingabemöglichkeiten stehen prinzipiell auch für Studierende zur Verfügung. Falls Sie eine Textarea mit WYSIWYG-Editor für die Eingabe längerer (HTML-)formatierter Texteingaben in Ihre Aufgabenseite eingebunden haben (siehe den vorangegangenen Abschnitt), so können diese dort bereits wahlweise LaTeX-Formeln selbst in den Text eingeben oder dazu auch den Formeleditor innerhalb des WYSIWYG-Editors aufrufen (sofern Sie den Editor nicht in der einfachsten, „abgespeckten“ Variante eingebunden haben).

Wenn Sie aber direkt nur nach der Eingabe einer bestimmten Formel fragen möchten und dazu ein Eingabefeld (`input`-Element) in die Aufgabenseite einfügen möchten, das aber nicht nur die Plaintext-LaTeX-Eingabe ermöglicht, sondern eben mit einem Formeleditor ausgestattet ist, so geht das ab sofort wie folgt: Tragen Sie ins `type`-Attribut des Input-Elements den Übungssystem-spezifischen Typ `latex` ein, z.B.:

```
<input type="latex" name="FeldA1">
```

Das Online-Übungssystem wird dann dieses Input-Feld durch einen Formeleditor ersetzen (für LaTeX-erfahrene Nutzer auch mit der Option, das Originalfeld zur Anzeige oder Eingabe von LaTeX-Quellcode zusätzlich einzublenden). Das kann dann (Stand Januar 2023) z.B. wie folgt aussehen:



Screenshot von Formeleditor mit aktivierter virtueller Bildschirmtastatur

Mehrere gleichnamige Eingabefelder, `data-separator`-Attribut

Im vorangegangenen Abschnitt wurde bereits am Beispiel von Checkboxen demonstriert, dass mehrere Input-Elemente zum selben Eingabefeldnamen (hier `FeldA2`) existieren dürfen. Checkboxen und Radiobuttons sind der gebräuchlichste Anwendungsfall hierfür.

Aber auch für Texteingabefelder (`<input type="text">` oder `<input type="number">` etc.) ist das prinzipiell möglich. In dem Fall sollten Sie jedoch *unbedingt* das im Folgenden genauer erläuterte `data-separator`-Attribut verwenden.

Nehmen wir z.B. an, Sie würden Text-Eingabefelder wie folgt erzeugen:

```
Teil 1: <input type="text" name="FeldA3"><br>
Teil 2: <input type="text" name="FeldA3">
```

Dann würden tatsächlich – genau wie bei den Checkboxen aus obigem Beispiel – beide Teileingaben zu FeldA3 per Komma getrennt zu einem eingesendeten String zusammengefügt, d.h. gibt jemand unter »Teil 1« das Wort `Hallo` ein und unter »Teil 2« das Wort `Welt`, dann lautet die in Feld A3

gespeicherte Einsendung am Ende `Hallo,Welt`.

Das Problem ist aber: Wenn der Student anschließend das Aufgabenformular erneut aufruft und ihm dabei seine letzten Einsendungen wieder im Formular eingeblendet werden, dann erscheint in *beiden* Eingabefeldern zu FeldA3 der zuletzt eingesendete Feldwert `Hallo,Welt`. Drückt der Student dann erneut auf Einsenden, ohne die Eingaben zu ändern, lautet die Einsendung bereits `Hallo,Welt,Hallo,Welt`, nach der dritten Einsendung dann schon `Hallo,Welt,Hallo,Welt,Hallo,Welt,Hallo,Welt` etc. Die Länge der Einsendung verdoppelt sich also mit jeder erneuten Einsendung und wächst damit exponentiell!

Nur beim Ausfüllen von Checkboxen erkennt das Online-Übungssystem automatisch, dass es die vorliegende Einsendung wieder „an den Kommas“ in Teile aufsplitten und diese Teile mit den `value`-Attributen der Checkboxen abgleichen muss, um eben wieder genau diejenigen Checkboxen anzukreuzen, deren Values in der Einsendung vorkommen. (Bei Radiobuttons liegt ohnehin nur der Value eines der Radiobuttons als Einsendung vor, und der Radiobutton mit dem zuletzt eingesendeten Wert wird dann wieder aktiviert.)

Die oben demonstrierte Vervielfachung der Texteingaben bei mehreren gleichnamigen Textinputs können Sie verhindern, indem Sie das `data-separator`-Attribut am ersten dieser Text-Inputs angeben. Dabei können Sie obendrein gleich das ansonsten implizit verwendete Komma durch ein Zeichen oder eine Zeichenkette eigener Wahl ersetzen.

Modifizieren wir z.B. obiges Beispiel zu:

```
Teil 1: <input type="text" name="FeldA3" data-separator=" und "><br>
Teil 2: <input type="text" name="FeldA3">
```

Hinweise:

- Das `data-separator`-Attribut muss nur einmalig angegeben werden, und zwar im *ersten Input* des jeweiligen Namens. Die späteren gleichnamigen Inputs benötigen dieses Attribut nicht mehr. (Sollte es dort dennoch angegeben werden, wird es ignoriert. Sollte in obigem Beispiel also das `data-separator`-Attribut nur im zweiten, nicht im ersten Input stehen, ist es also komplett wirkungslos! Und sollte in *beiden* Inputs ein solches Attribut angegeben werden, so zählt nur das aus dem ersten Input.)
- Wird das `data-separator`-Attribut an einem Input-Element angegeben, das das *einzige* Input desselben Namens ist (auf das also keine weiteren gleichnamigen Inputs mehr folgen), so wird es ebenfalls ignoriert.

Gibt nun ein Student in obiges Teilformular wieder die Worte `Hallo` und `Welt` in die beiden Textinputs ein, so wird als Einsendung im Feld A3 der String `Hallo und Welt` gespeichert. Und bei Wiederaufruf der Aufgabenseite wird die letzte Einsendung `Hallo und Welt` wieder in die zwei Teile `Hallo` sowie `Welt` zerlegt und diese wieder in die Eingabefelder eingetragen.

Ein Konfliktfall kann sich natürlich ergeben, wenn dieser Trennstring auch in der Eingabe vorkommt. Gibt ein Student nun z.B. ins erste Feld `Pünktchen und Anton` ein und ins zweite Feld `Erich Kästner`, dann lautet die Gesamt-Einsendung zu Feld A3 `Pünktchen und Anton und Erich Kästner` und besteht somit aus *drei* per „ und “ aufgezählten Teilen. Beim Wiederaufruf der Aufgabenseite werden diese *drei* Teile dann auf die *zwei* Inputs wie folgt verteilt: Je ein Teil pro Input, die überschüssigen Teile landen mit im letzten Input. D.h. im ersten Feld würde `Pünktchen` und im zweiten Input `Anton und Erich Kästner` eingetragen. Das entspricht zwar offensichtlich nicht wieder exakt der zuvor vorgenommenen Eingabe, ist aber zumindest insofern verlustfrei, als dass bei

einer erneuten Einsendung wieder dieselbe zusammengesetzte Einsendung `Pünktchen und Anton und Erich Kästner` entsteht, Neueinsendungen ohne Änderungen also nichts an der Einsendung ändern.

Wenn Sie aber im `data-separator`-Attribut nur *genau ein Zeichen* (also keine längere Zeichenkette) eingeben, dann sind die Eingaben sogar immer exakt wieder in die Originaleinsendung zerlegbar. Zu diesem Zweck wird dann nämlich eine Maskierung von ggf. in der Eingabe vorkommenden Trennzeichen (Separatoren) durch einen sog. *Escape Character* vorgenommen. Als Escape-Character wird immer ein Backslash (`"\"`) verwendet, außer im Fall, dass `data-separator="\"` definiert wird, dann wird ein Slash `"/"` als Escape-Character verwendet.

Ändern wir dazu das Beispiel noch einmal wie folgt ab:

```
Teil 1: <input type="text" name="FeldA3" data-separator=","><br>
Teil 2: <input type="text" name="FeldA3">
```

Jetzt wird eine Einsendung wie `Anton` und `Berta` wieder (wie bei Verzicht auf `data-separator`) zu `Anton,Berta` zusammengesetzt, aber bei Wiederaufruf des Aufgabenformulars auch wieder korrekt auf die beiden Eingabefelder verteilt.

Und wenn ein Nutzer das Trennzeichen Komma in der Einsendung verwendet, wird jedem Komma ein Backslash vorangestellt. So würde z.B. aus den Eingaben `Kästner, Erich` und `Pünktchen und Anton` die Einsendung `Kästner\, Erich,Pünktchen und Anton`, die bei erneuten Formularaufruf auch wieder korrekt in die Originaleingaben zerlegt und wieder in die Eingabefelder eingefügt würde.

Sollte der Einsender auch einen Backslash eingeben, wird dieser zur Sicherheit verdoppelt. Beispiel: Im ersten Feld gebe der Student `C:\,D:\` ein, im dritten Feld `\\Server\test`. Dann lautet die in Feld A3 gespeicherte Einsendung: `C:\\\,D:\\,\\\\Server\\test`, und das Übungssystem ist in der Lage, diesen String wieder eindeutig in die zwei Teileingaben zurück zu zerlegen, indem der String an jeder Stelle aufgetrennt wird, an der ein Komma steht, dem kein Backslash oder eine gerade Anzahl von Backslashes vorangestellt ist und anschließend jeder Backslash, der vor einem anderen Backslash oder einem Komma steht, wieder gelöscht wird. So entstehen wieder die Teileingaben `C:\,D:\` und `\\Server\test`.

- Diese Zerlegung geschieht insbesondere beim Wiederaufruf des Aufgabenformulars: Die Teileingaben erscheinen wieder genauso in den Eingabefeldern, wie sie erstmalig eingegeben wurden.
- Auch automatische Korrekturmodule können solche mehrteiligen Eingaben wieder zerlegen.
  - Beim internen Korrekturmodul wird dies für die Begriffsbewerter (Typen `Begriffe`, `BegriffeIC`, `BegriffeSmart` und `BegriffeSmartIC` sowie Typen `Begriffsfolge`, `BegriffsfolgeIC`, `BegriffsfolgeSmart` und `BegriffsfolgeSmartIC`) unterstützt. Diese Begriffsbewerter erlauben damit das Eingaben von Teilbegriffen zu einem Begriff: Es wird ein „logisches Eingabefeld“ (eine Eingabefeld-ID wie hier „FeldA3“) pro gefragtem Gesamtbegriff verwendet und mehrere Eingabefelder mit diesem Namen (bei Verwendung des `data-separator`-Attributs mit genau einem Trennzeichen darin) für die jeweiligen Teilbegriffe.
  - Auch selbst erstellte *externe* Soap-Korrekturmodule können solche zusammengesetzten Einsendungen zu einem Feld wieder in die Teileingaben aus mehreren Inputs zerlegen, wenn das `data-separator`-Feld verwendet wurde. Der verwendete Separator-String (also der Inhalt des `data-separator`-Attributs) und im Falle eines genau ein Zeichen langen Separators der vom Übungssystem implizit gewählte Escape-Character werden

dem Korrekturmodul dazu mit übermittelt.

- Bei der Erzeugung von Antworthäufigkeitstatistiken ist ebenfalls eine Konfiguration möglich, dass solche zusammengesetzten Eingaben wieder zerlegt und die Teile gezählt werden. Siehe dazu Kapitel Antworthäufigkeits-Statistik zu einer Aufgabe aktivieren/konfigurieren.

Das hier beschriebene „Escaping" mit Backslash (oder Slash) erfolgt explizit ausschließlich für Felder mit einem `data-separator`-Attribut mit Wert der Länge 1 am ersten Vorkommen von mehreren gleichnamigen Inputs. Im Defaultfall von Checkboxen ohne `data-separator`-Attribut (siehe oben) erfolgt also kein Escaping, d.h. Checkboxen funktionieren nur, wenn in den Value-Attributen kein Komma vorkommt. Dies gilt in erster Linie aus Abwärtskompatibilitätsgründen. Aber als Aufgabenautor können Sie bei Checkboxen ja auch selbst sicherstellen, dass keine Kommas in den möglichen Values vorkommen können – anders als bei Texteingabefeldern, bei denen Sie nie ausschließen können, dass Teilnehmer/-innen genau das von Ihnen definierte Trennzeichen in ihrer Eingabe mit eintippen – weshalb eben genau für diesen Fall das Escaping eingeführt wurde.

Hinweis: Wenn Sie diese Eingaben (insbesondere bei Wahl genau eines Trennzeichens, so dass ggf. auch ein Escaping stattfindet) nicht nur an ein (Vor-)Korrekturmodul übermitteln, sondern auch direkt quittieren oder in der Korrekturseite anzeigen möchten, dann verwenden Sie dazu am besten das Variablensuffix `SPLIT` (also eine Variable der Form `$FeldA3SPLIT`), siehe Feldvariablen. Diese Schreibweise sorgt dafür, dass dort nicht der gespeicherte (und eindeutig immer wieder in seine Bestandteile zurückzerlegbare) String mit Trennzeichen und ggf. Escape-Zeichen angezeigt wird, sondern eine besser menschenlesbarer (nicht immer eindeutig zurück-zerlegbare) Darstellung mit wahlweise », « oder » / « als Aufzählungszeichen.

Bearbeitungsstatus einer Aufgabe: `data-ignore`-Attribut

Seit Februar 2019 unterstützt das Online-Übungssystem einen neuen Mechanismus zur Festlegung, in welchen Fällen eine Aufgabe überhaupt als bearbeitet gelten soll.

Ohne Nutzung der i.F. beschriebenen neu eingeführten `data-ignore`-Attribute zu Input-Tags gilt – wie bisher – eine Aufgabe grundsätzlich als von einem Studenten bearbeitet, sobald dieser zu mindestens einer Teilaufgabe dieser Aufgabe einmal den Einsenden-Knopf gedrückt hat. Selbst wenn er alle Eingabefelder leer gelassen hat, also einfach nur bei unausgefülltem Formular den Einsenden-Button gedrückt hat, gilt also – trotz des Fehlens sinnvoller Eingaben – die Aufgabe als von ihm bearbeitet. Dies äußert sich in verschiedenen Punkten:

1. In der Aufgabenübersicht oder einer Aufgabenliste im Seitenmenü sieht der Student ein Sternchen als Anzeige dafür, dass er die Aufgabe bearbeitet hat.
2. Auch in der Ergebnisübersicht wird dem Studenten ab sofort angezeigt, dass Eingaben von ihm vorliegen (und deren Bewertungsstatus).
3. Auch ein Betreuer sieht in der Einsendungsübersicht die Anzahl „offener Hefte" (Anzahl von Studenten, die schon mindestens eine Aufgabe eines Aufgabenhefts bearbeitet haben). Dort wird dieser Student ab der ersten Einsendung mitgezählt.
4. Beim Heft-Schließen wird zu jeder bearbeiteten Aufgabe eine Korrektur erstellt. Im Falle einer handbewerteten Aufgabe ist das ein Dokument mit allen Eingaben des Studenten, das einem Korrektor zur Korrektur/Bewertung zugeteilt wird. Hatte der Student das Formular leer eingeschickt, also ohne irgendetwas einzugeben, wird demnach dennoch immer eine leere Korrektur erstellt, die von einem Korrektor (typischerweise mit null Punkten) zu bewerten ist.

Insbesondere bei Aufgaben, in denen jeder Teilnehmer seine Lösung in Form einer einzelnen Datei hochladen soll, ist dieser Zustand unschön, denn die Aufgabe gilt (nach dem bisherigen Verfahren) immer als bearbeitet, selbst wenn der Student gar keine Datei hochgeladen hat, sondern einfach den

Upload-Button gedrückt hat, ohne eine Datei auszuwählen. Auch die im Folgenden (siehe Einfache Dateiuploads) beschriebene Möglichkeit, zumindest bei der ersten Aufgabenbearbeitung per Pflichtfeld-Mechanismus eine Dateieinsendung zu erzwingen (also eine Leereinsendung ohne Dateiupload zu vermeiden), birgt immer noch das Problem, dass der Student anschließend den Dateiupload wieder zurückziehen kann: In diesem Zustand liegt dann doch wieder eine leere Einsendung vor, und die Aufgabe wird dem Studenten dann bei Standardeinstellung dennoch immer als bearbeitet angezeigt – was unglücklich ist, da bei derartigen Aufgaben viele Studenten die „Aufgabe wurde bearbeitet"-Anzeige intuitiv als „Dateieinsendung liegt vor" interpretieren.

Es kann daher wünschenswert sein, eine Aufgabe als (noch oder wieder) von einem Studenten unbearbeitet zu markieren, wenn von diesem Studenten nur eine leere Formulareinsendung vorliegt. Das gilt aber nicht global für *alle* Aufgaben! Bei Multiple-Choice-Fragen (X aus N) z.B. kann auch das Einsenden eines leeren Formulars, in dem keine einzige Antwortalternative angekreuzt wurde, als sinnvolle Antwort gelten – nämlich als die Antwort, dass alle Alternativen falsch sind. *Solche* Aufgaben müssen dann natürlich weiterhin auch bei Leereinsendungen als bearbeitet gelten und bewertet werden. Daher wurde das Verhalten des Online-Übungssystems diesbezüglich nicht einfach komplett geändert, sondern vielmehr eine neue Konfigurationsmöglichkeit auf Formularfeld-Ebene geschaffen:

Zu jedem `<input>`-Element Ihres Aufgabenformulars können Sie das Attribut `data-ignore` hinzufügen, das jeweils einen der folgenden drei Werte bekommen muss (Groß-/Kleinschreibung des Attributwerts ist dabei egal):

data-ignore	Bedeutung
`never` (Default)	Jede Eingabe (auch eine leere) in dieses Feld zählt als „sinnvolle Antwort".
`empty`	Nur nicht-leere Eingaben in dieses Feld zählen als Antworten.
`always`	Dieses Feld trägt gar nicht zum Bearbeitungszustand der Aufgabe bei, Eingabe gelten nie als inhaltliche Bearbeitung / Antwort auf eine Aufgabenstellung.

Falls mehrere Input-Elemente mit demselben Feldnamen existieren (typisch für Checkboxen und Radiobuttons, die eine Gruppe bilden), muss das `data-ignore`-Attribut im *ersten* dieser gleichnamigen Input-Elemente angegeben werden (vgl. Checkboxen zu `FeldA2` in obigem begleitendem Fallbeispiel).

Wird das `data-ignore`-Attribut für ein Feld gar nicht angegeben, gilt aus Abwärtskompatibilitätsgründen das bisherige Verhalten als Default. *Keine* Angabe ist also gleichbedeutend mit der Angabe `data-ignore="never"`.

Sendet ein Student nun z.B. ein komplett leeres Formular ein, und ist mindestens eines der Formularfelder mit `data-ignore="never"` versehen (oder ganz ohne `data-ignore`-Attribut), so gilt dieses Eingabefeld und damit die gesamte Aufgabe als bearbeitet, trotz der leeren Einsendung.

Sendet ein Student ein komplett leeres Formular ein, und sind alle Formularfelder entweder mit `data-ignore="empty"` oder `data-ignore="always"` markiert, also keines mit `never` oder ohne Angabe, dann gilt die Aufgabe als unbearbeitet. Erst bei Ausfüllen mindestens eines der mit `data-ignore="empty"` markierten Felder nimmt die Aufgabe (für den Studenten) den Zustand „bearbeitet" an.

Für die meisten Formularfelder, insbesondere für Dateiuploads oder Text-Eingabefelder, ist `empty` die empfohlene Einstellung. Das gilt auch z.B. für Radio-Buttons von Einfachauswahl-Aufgaben, sogar

auch für Checkboxen einer Multiple-Choice-Frage, *sofern* für diese gelten soll, dass immer mindestens eine Antwort korrekt ist und dass die Frage erst als bearbeitet gelten soll und bewertet werden soll, wenn mindestens eine Checkbox angekreuzt wurde. Der Wert `never` (oder gar keine Angabe des Attributs) ist insbesondere wichtig für Multiple-Choice-Fragen, bei denen alle Alternativen falsch sein können, oder auch, falls zwar mindestens eine Alternative korrekt ist, für Studenten, die gar nichts markiert haben, die Antwort aber dennoch (auf Teilübereinstimmung hin) bewertet werden soll.

Im obigen begleitenden Fallbeispiel wurde für den Dateiupload und die Zahleneingabe jeweils der empfohlene `empty`-Wert angegeben, für die beiden Checkboxen dagegen `never`. Die Angabe `empty` für den Zahlen-Input hat somit in diesem konkreten Fall streng genommen keinen Effekt[11]: Da die Teilaufgabe A ein Eingabefeld mit `never`-Einstellung (die Checkboxen) enthält, gilt die gesamte Aufgabe spätestens dann als bearbeitet, wenn ein Student zur Teilaufgabe A etwas einsendet, also den `einsendenA`-Button betätigt – auch bei komplett leerem Formular.

Ebenso gilt die Aufgabe des Fallbeispiels als bearbeitet, wenn in Teilaufgabe B eine Datei hochgeladen wurde. Nimmt ein Student dagegen für Teilaufgabe B eine Leereinsendung vor (`einsendenB` drücken, ohne in `FeldB1` eine Datei ausgewählt zu haben) und hat er `einsendenA` noch nie gedrückt, so gilt die Aufgabe als von ihm unbearbeitet. Für den Studenten äußert sich das zunächst so, dass die Aufgabe in den Übersichten nicht (mehr) als bearbeitet markiert wird. Ruft er das Aufgabenformular erneut auf, wird ihm dennoch – anders als vor der ersten Bearbeitung – sein aktueller Einsendestand für Teilaufgabe B angezeigt, in Form des Textes »Ihre zuletzt hochgeladene Datei: Keine Datei eingesendet!«. Die Variable `$IfExistsB1(…)` wertet also eine Leereinsendung *immer* als Einsendung, unabhängig von der `ignore`-Einstellung. Das ist auch Absicht: So kann der Student hier jederzeit unterscheiden (bei Aufruf des Aufgabenformulars), ob die Aufgabe *deshalb* als unbearbeitet gilt, weil er noch nie etwas eingesendet hat, oder weil er eine leere Einsendung vorgenommen hat. Daher stehen auch die Attribute `data-ignore="empty"` und `data-if-emtpy="…"` in keinem Konflikt zueinander: Auch wenn Leereingaben bei der Frage nach dem Bearbeitungsstatus ignoriert werden, wird dennoch der If-Empty-Ersatztext in Quittung und Korrektur oder – wie hier bei Dateiuploads – auch dem Aufgabenformular angezeigt.

Weiterhin wird zu einer als unbearbeitet geltenden Aufgabe bei der Heftabgabe auch keine Korrektur erzeugt: In obigem Beispielfall (Leereinsendung zu Teilaufgabe B, keine Einsendung zu Teilaufgabe A) wird also insbesondere – anders als bei Default-Einstellung – keinem Korrektor eine Korrekturseite vorgelegt, in der er lediglich den Text »Keine Datei eingesendet.« bewerten soll, und der Student wird auch in der Ergebnisübersicht nur sehen, dass er die Aufgabe nicht bearbeitet hat und dort nie eine mit 0 Punkten bewertete „leere Korrektur“ vorfinden.

Die dritte Einstellmöglichkeit `always` ist für Formularelemente vorgesehen, die selbst keine Aufgabenbestandteile sind. Typisches Beispiel: Eine Eigenständigkeitserklärung per Pflicht-Checkbox der Art »Ich erkläre, dass ich diese Arbeit eigenständig bearbeitet habe.«:

```
<form method="POST"

action="$WebAssignServer/$Veranstaltername/EinsendungMultipart/$KursNr/$VersionsNr/$AufgabenheftNr/$AufgabenNr/"
      enctype="multipart/form-data"
      class="validation-styles">
<p>
<input type="file" name="FeldA1" data-if-empty="Keine Datei eingesendet!" data-ignore="empty">
<button type="submit" name="einsendenA">Datei einsenden</button>
<br>
$IfExistsA1(Ihre zuletzt hochgeladene Datei: )$FeldA1</p>
<p>
<input type="checkbox" name="FeldA2" value="Ja" data-if-empty="Nein" required data-ignore="always" id="assert">
<label for="assert">Ich erkläre, dass ich diese Arbeit eigenständig bearbeitet habe.</label>
</p>
</form>
```

Die Eingabevalidierung wird hier eine Einsendung nur erlauben, wenn die Checkbox angekreuzt ist, da die Checkbox als `required` markiert ist. Eine hundertprozentige Sicherheit, dass ein Einsenden ohne Ankreuzen dieser Checkbox nicht möglich ist, besteht allerdings nicht. Gegenüber der alten JavaScript-basierten Lösung ist die Umsetzung mit HTML5-Required-Attribut insofern sicherer, als dass ein Browser selbst bei abgeschaltetem JavaScript nun eine Einsendung verhindern sollte, wenn die nicht markiert wurde. Aber wenn z.B. ein sehr alter Browser verwendet wird, der das required-Attribut noch nicht unterstützt, könnte eine Einsendung z.B. auch ohne diese Markierung gelingen. Deshalb wurde per `data-is-empty` auch für diesen Fall der Nicht-Markierung eine entsprechende Darstellung (`Nein`) für Quittungs- und Korrekturseiten festgelegt.

Wählt ein Student nun keine Datei aus, aber markiert diese Checkbox und sendet ein, so sollte wegen der fehlenden Dateieinsendung die Aufgabe am besten als unbearbeitet gewertet werden – und das, obwohl in `FeldA2` (der Checkbox) eine nicht-leere Eingabe vorliegt. Diese Checkbox allein stellt eben keine sinnvolle Aufgabenbearbeitung dar, und daher soll der Zustand der Checkbox nicht in den Bearbeitungsstatus der Aufgabe einfließen, sondern immer ignoriert werden. Genau das wird per `data-ignore="always"` festgelegt. Somit wird lediglich das verbleibende Eingabefeld `FeldA1` (Dateiupload) berücksichtigt: Ist dieses leer, gilt die Aufgabe als unbearbeitet (wegen `data-ignore="empty"`), bei vorliegender Datei dagegen als bearbeitet.

Vorgefüllte Felder

Es ist möglich, in die Eingabefelder bereits im Formular bestimmte Werte einzutragen (durch ein `value`-Attribut in `input`-Elementen für Texteingaben, Text-Inhalte in einem `textarea`-Element oder `checked`- oder `selected`-Attribute in Checkboxen, Radiobuttons oder Select-Options). Diese werden Studenten dann beim ersten Aufruf der Aufgabe angezeigt / vorgeschlagen. Sobald ein Student aber zu einer (Teil-)Aufgabe einmalig etwas eingesendet hat und das Aufgabenformular anschließend erneut lädt, wird die letzte Einsendung des Studenten in das jeweilige Eingabefeld eingetragen und überschreibt damit eine ggf. im Formular stehende Vorgabebelegung des Felds.

Einsendungs-Wiederherstellung verhindern

Im Normalfall gilt für jedes Eingabefeld (also jedes `input`-Element mit einem Namen `FeldXy`, wobei `X` ein Buchstabe zur Teilaufgabenkennung und `y` eine Zahl (Feldnummer) ist), dass diese Felder zwar beim ersten Aufruf der Aufgabenseite leer (oder mit Vorgabewert vorgefüllt, s.o.) sind, aber bei jedem wiederholten Aufruf der Aufgabenseite, *nachdem* zur Teilaufgabe `X` eine Einsendung vorgenommen wurde, mit dem zuletzt eingesendeten Wert gefüllt werden, also die letzte Eingabe zur Nachbearbeitung wiedergeben.

In seltenen Ausnahmefällen kann es sinnvoll sein, dieses Ausfüllen der vorliegenden letzten Einsendung zu einem Eingabefeld zu unterdrücken, so dass dieses Feld auch bei wiederholtem Aufruf der Aufgabenseite immer leer ist bzw. immer den vorgefüllten Wert wie beim ersten Aufgabenstart hat. (Insbesondere für Hidden-Fields kann das sinnvoll sein, oder auch z.B. für Checkboxen, die vor jeder Einsendung – auch im Falle wiederholter Einsendungen – aktiv angekreuzt werden müssen).

Um dieses Wiederherstellen der letzten Einsendung in einem Input-Element zu unterdrücken, ist im `class`-Attribut des Inputs die Klasse `no-restore` einzutragen, z.B.:

```
<input type="hidden" name="FeldA12" value="A" class="no-restore">
```

Falls es mehrere Inputs mit demselben Feldnamen geben sollte (wie z.B. bei Checkboxen oder Radiobuttons), ist diese Klasse i.d.R. an allen davon anzubringen, denn die Klasse wirkt sich immer nur auf das konkrete Input-Element aus, an dem sie steht, weitere Inputs mit demselben Feldnamen „erben“ diese Eigenschaft also nicht.

Beispiel:
Betrachten wir folgende zwei Radiobuttons, von denen nur einer mit Klasse `no-restore` versehen ist:

```
<input type="radio" name="FeldA1" value="A" class="no-restore">
<input type="radio" name="FeldA1" value="B">
```

In diesem Fall würde, falls zuletzt der Wert `"A"` (erster Radiobutton) eingesendet wurde, beim Neu-Laden der Seite diese Auswahl nicht wiederhergestellt, es wären also wieder beide Radiobuttons unmarkiert. Anders dagegen nach Einsendung von `"B"` (zweiter Radiobutton): In dem Fall würde beim erneuten Aufruf der Aufgabenseite der zweite Radiobutton automatisch wieder selektiert (`checked`), weil für diesen Radiobutton die Einsendungs-Wiederherstellung eben *nicht* unterdrückt ist.

Einfache Dateiuploads

Soll ein Student eine Datei einsenden, genügt in der Regel ein einfaches Dateiupload-Feld, d.h. ein Formularfeld, in dem der Student genau eine lokale Datei auswählen und einsenden kann. Jede spätere, erneute Einsendung einer Datei überschreibt die vorherige Einsendung. Zum Zurückziehen einer versehentlich vorgenommenen Einsendung kann ein Student also entweder eine andere Datei einsenden, die die erste ersetzt, oder auch eine leere Einsendung (ohne Auswahl einer neuen Datei) vornehmen, um die zuvor eingesandte Datei ersatzlos zu löschen.

Die Einbindung wird in obigem Fallbeispiel bereits demonstriert. Die folgenden Punkte sollen noch einmal die wesentlichen dabei zu beachtenden Aspekte zentral zusammenstellen:

1. Für das Eingabefeld wird ein einfaches Input-Element mit `type="file"` verwendet, z.B.: `<input type="file" name="FeldB1" …>`
2. Für jeden Dateiupload sollte eine eigene „technische" Teilaufgabe <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#ta> vorgesehen werden, in obigem Beispiel Teilaufgabe `B`, d.h. es gibt keine weiteren Eingabefelder mit einer Kennung `FeldB*`.
    - Ausnahme: Falls z.B. beim Hochladen der Datei der Student zwangsweise eine Checkbox markiert haben soll, z.B. um eine Erklärung abzugeben wie „Ich versichere, die Arbeit eigenständig bearbeitet zu haben" oder „…die Hinweise gelesen zu haben" etc., dann kann eine entsprechende Checkbox derselben Teilaufgabe zugeordnet werden: `<input type="checkbox" name="FeldB2" required value="Ja">`. Das `required`-Attribut markiert die Checkbox als Pflichtfeld, siehe auch Handbuch zur Formularvalidierung <https://online-uebungssystem.fernuni-hagen.de/download/JSFormValidation/FormularValidierung.html> .
3. Für diese Teilaufgabe `B` ist dann ein eigener Submit-Button vorzusehen, typischerweise mit `"Upload"` oder `"Hochladen"` o.ä. benannt: `<button type="submit" name="einsendenB">Datei hochladen</button>`
4. Das Formular muss für Multipart-Submit ausgelegt sein (siehe `form`-Tag).
5. Da ein File-Input keine bereits vorliegende Dateieinsendung anzeigen kann, sollte unterhalb des Inputs und Upload-Buttons ein Text `"Ihre letzte Einsendung"` o.ä. mit der Variablen `$FeldB1` oder `$FeldB1HREF` o.ä. folgen. Und damit dieser erst angezeigt wird, wenn wirklich schon eine Einsendung vorliegt, sollte die Kontrollvariable `$IfExistsB1` verwendet werden, wie in obigem Fallbeispiel bereits demonstriert.
6. Damit an der Stelle `$FeldB1` bei leerer Einsendung nicht ein so nichtssagender Text wie „Ihre letzte Einsendung: [leer]" steht, sollte in jedem Fall das Input-Element noch um ein `data-if-empty="Keine Datei hochgeladen"`-Attribut (ggf. mit abweichendem Text) ergänzt werden, s.o.
7. Per `data-ignore="empty"` kann festgelegt werden, dass Einsendungen ohne ausgewählte Datei nicht als zu bewertende Antworten gelten sollen, sondern die Aufgabe als unbeantwortet gelten soll, wenn der Student zwar den Upload-Button gedrückt hatte, aber ohne eine Datei auszuwählen (und die Teilaufgabe `A` nicht bearbeitet wurde), siehe Bearbeitungsstatus einer Aufgabe.
8. Sollen nicht beliebige Dateitypen akzeptiert werden, so kann ein Dateiupload-Feld auf bestimmte Dateitypen (Dateiendungen und/oder MIME-Types) eingeschränkt werden, siehe: Handbuch zu Dateityp-Einschränkungen <https://online-uebungssystem.fernuni-hagen.de/download/FileUploadAccept/FileUploadAccept.html>

Dateiuploadfeld als Pflichtfeld?

Möchten Sie verhindern, dass ein Student einfach nur den Upload-Button drücken kann, ohne eine Datei ausgewählt zu haben, könnten Sie das Upload-Feld zu einem *Pflichtfeld* machen, indem Sie das Attribut `required` hinzufügen. Dabei sollte auch das `form`-Tag vorzugsweise mit `class="validation-styles"` ausgezeichnet werden, was (Stand Mai 2023) folgendes bewirkt:

- Die Dateiauswahl wird mit den validation-styles *nicht* mehr direkt beim Laden der Seite rot umrandet (und damit als „noch fehlerhaft ausgefüllt" markiert), sondern zunächst nur mit einem angehängten Sternchen als Pflichtfeld markiert.
- Erst nach einem Einsendeversuch ohne Dateiauswahl wird nachträglich eine rote Umrandung

und ein rotes Warndreieck-Icon hinzugefügt, um den Grund für die fehlgeschlagene Einsendung stärker hervorzuheben. (Außerdem erscheint i.d.R. auch ein Popup-Hinweis, aber das ist unabhängig von der validation-styles-Klasse.)

Allerdings gibt es bei dieser Datei-Pflichtfeld-Konstruktion zwei wichtige Aspekte zu beachten:

Zum Ersten gilt diese Pflichtangabe immer für das gesamte HTML-Formular, nicht nur für eine Teilaufgabe. Falls Ihre Aufgabe also *nicht nur* aus einem Dateiupload-Feld besteht, sondern auch noch weitere Teilaufgaben mit eigenen Upload-Buttons umfasst, ist dieser Mechanismus höchstens nutzbar, wenn auch für jede Teilaufgabe im Aufgaben-HTML jeweils ein eigenes `<form>`-Element angelegt wird! Andernfalls ließen sich Eingaben auch zu *anderen* Teilaufgaben nicht einsenden, ohne im Pflicht-Dateifeld eine Datei ausgewählt zu haben, und das, obwohl diese ausgewählte Datei ja beim Einsenden jeder *anderen* Teilaufgabe nicht als Einsendung gespeichert würde!

*Allgemein empfehlen wir, ein Pflichtfeld wirklich nur dann einzusetzen, wenn die gesamte Aufgabe **allein** aus diesem Dateiupload-Feld (als `FeldA1`) besteht und keine weiteren Teilaufgaben existieren!*

Zum Zweiten wäre mit einer generellen Pflichtfeld-Regelung auch das Zurückziehen einer versehentlichen Einsendung durch Überschreiben per Leereinsendung nicht mehr möglich!

Daher empfehlen wir, das Feld nur so lange zum Pflichtfeld zu machen, wie der Student die Teilaufgabe noch nicht bearbeitet hat. D.h. ruft er die Aufgabenseite auf und hat noch nie eine Datei hochgeladen, ist das Upload-Feld ein Pflichtfeld. Hat er aber über dieses Pflichtfeld schon einmal eine Datei eingesendet und ruft die Aufgabe erneut auf, handelt es sich *nicht mehr* um ein Pflichtfeld, so dass ein Zurückziehen der letzten Einsendung per neuer Leereinsendung möglich ist. Das lässt sich mit Hilfe der oben eingeführten Steuer-Variablen `$IfNotExistsA1` wie folgt realisieren:

```
<p>
<input type="file" name="FeldA1" $IfNotExistsA1(required) data-if-empty="Keine
Datei eingesendet!" data-ignore="empty">
<button type="submit" name="einsendenA">Datei einsenden</button>
<br>
$IfExistsA1(Ihre zuletzt hochgeladene Datei: )$FeldA1</p>
```

Der Aufgabenerstellungsassistent für handbewertete Aufgaben geht derzeit übrigens genauso vor wie hier beschrieben (nur für Aufgaben, die wirklich *nur* aus einem Dateiupload bestehen, höchstens noch mit einer Bestätigungs-Checkbox kombiniert).

Hinweis: Auch für die Quittungsseite empfehlen sich besondere Maßnahmen, worauf im entsprechenden Kapitel noch gesondert eingegangen wird.

Spezialfälle: Dateiuploads für mehrere Dateien

Ein „einfacher“ Dateiupload, also die Verwendung eines Eingabefelds `<input type="file" name="FeldB1" …>` mit einem Submitbutton der Art `<input type="submit" name="einsendenB" …>` (wie in obigem begleidenten Fallbeispiel), bedingt, dass ein Student zum im Formular verwendeten Eingabefeld genau eine Datei einsenden kann. Jede spätere, erneute Einsendung einer Datei überschreibt die vorherige Einsendung.

Soll ein Student mehr als eine Datei einsenden (können), sind folgende Fälle zu unterscheiden:

n Einzel-Uploads

Ist die Anzahl *n* der genau oder maximal einzusendenden Dateien bekannt (z.B. jeder Student soll bis zu 4 oder immer genau 3 Dateien einsenden), so können natürlich mehrere solcher „normalen" Dateiuploads wie oben beschrieben verwendet werden.

Dabei sollte beachtet werden, dass für jedes Dateiupload-Feld jeweils eine eigene Teilaufgabe angelegt wird, die genau ein File-Input-Feld und einen eigenen Einsenden-Button (Upload-Button) enthält, z.B.:

```
<form method="POST"

action="$WebAssignServer/$Veranstaltername/EinsendungMultipart/$KursNr/$Version
sNr/$AufgabenheftNr/$AufgabenNr/"
      enctype="multipart/form-data">

<p>Datei 1: <input type="file" name="FeldA1" data-if-empty="Keine Datei
eingesendet!" data-ignore="empty">
<button type="submit" name="einsendenA">hochladen</button><br>
$IfExistsA1(Ihre zuletzt hochgeladene Datei: )$FeldA1HREF</p>

<p>Datei 2: <input type="file" name="FeldB1" data-if-empty="Keine Datei
eingesendet!" data-ignore="empty">
<button type="submit" name="einsendenB">hochladen</button><br>
$IfExistsB1(Ihre zuletzt hochgeladene Datei: )$FeldB1HREF</p>

</form>
```

Der Grund, warum jedes Upload-Feld eine eigene Teilaufgabe bilden sollte, ist, dass so für jedes der Upload-Felder auch nachträglich jede Datei einzeln wieder aktualisiert werden kann.

Würden Sie dagegen mehrere „normale" Dateiupload-Felder zu einer Teilaufgabe zusammengefassen, so würden immer alle gemeinsam eingesendet. Wählte ein Student, der schon einmal Dateien hochgeladen hat, dann nur in einem der Upload-Felder eine neue Datei aus, ließe die restlichen Felder jedoch leer, so würde er vermutlich erwarten, dass er damit nur diese eine frühere Einsendung überschreibt, tatsächlich würden jedoch alle früher zu der Teilaufgabe eingesendeten Dateien überschrieben, da für diejenigen Upload-Felder, zu denen er keine neue Datei ausgewählt hat, eine leere Einsendung erzeugt würde.

Multi-Dateiupload (mehrstufiger Upload)

Das Übungssystem bietet inzwischen die Möglichkeit, ein einziges File-Upload-Formularfeld zu verwenden und darin den Upload einer oder mehrerer (beliebig vieler) Dateien zu erlauben. Die Uploads können dann nacheinander erfolgen, d.h. ein Student kann eine erste Datei einsenden, später eine zweite, dritte u.s.w. Datei durch weitere Uploads hinzufügen.

Zu diesem Zweck sehen Sie wieder (wie beim Einzel-Upload auch) eine Teilaufgabe vor, die aus einem Datei-Upload-Feld und einem Submit-Button besteht, aber verwenden Sie für den Submit-Button statt `einsendenX` den Namen `einsendenPlusX` (wobei `X` der Buchstabe ist, der die Teilaufgabe bezeichnet).

Der `Plus`-Zusatz bedeutet, dass jede mit diesem Button zu einem Feld der genannten Teilaufgabe eingesandte Datei *nicht* die vorherige Dateieinsendung *überschreiben*, sondern vielmehr *ergänzen* soll:

- Wurde noch keine Datei in dem Feld eingesendet, passiert dasselbe wie bei gewöhnlichem `einsendenX`-Button: Die Datei wird als neue Einsendung gespeichert.
- Wurde zuvor eine (einzige) Datei eingesendet und nun per `einsendenPlusX` eine zweite, so wird als Einsendung eine neue ZIP-Datei gespeichert, die beide Dateien (erste und zweite Einsendung) umfasst.
- Liegt (z.B. nach zwei Einzeluploads) bereits eine ZIP-Datei als Einsendung vor und wird eine weitere Datei eingesendet, so wird diese zur bestehenden ZIP-Datei hinzugefügt.

Falls in einem solchen File-Upload-Feld auch ZIP-Dateien hochladbar sind (Sie also kein `accept`-Attribut, s.o., hinzugefügt haben, das erlaubte Dateitypen festlegt, ohne ZIP-Dateien mit zu erlauben), kann ein Student seinerseits auch mehrere Dateien gezippt in einem Schritt hochladen. In dem Fall gilt im Allgemeinen:

- Liegt noch keine Dateieinsendung vor, wird die hochgeladene ZIP-Datei als Einsendung gespeichert.
- Liegt bereits eine einzelne eingesendete Datei vor, wird sie zur hochgeladenen ZIP-Datei hinzugefügt.
- Liegt bereits eine ZIP-Datei mit früheren Einsendungen vor, werden beide ZIP-Dateien (die mit allen früheren Einsendungen und die hochgeladene) zu einer verschmolzen.

Sie können das File-Input-Element außerdem auch mit dem Attribut `multiple` ausstatten: Dann erlaubt der Browser die Auswahl von mehr als einer Datei zum Upload, so dass der Nutzer auch in einem einzigen Schritt gleich *mehrere* Dateien zu seiner Einsendung hinzufügen kann (ohne diese als ZIP-Datei einsenden zu müssen).

Namenskollisionen und Duplikate: Haben eine schon vorliegende und eine später hinzuzufügende Datei denselben Namen, wird die später eingesandte i.d.R. automatisch umbenannt (durch Anhängen von `_1` oder bei Bedarf einer größeren Nummer), so dass beide Dateien nebeneinander im resultierenden ZIP vorhanden sind. Ausnahme: Wenn zwei gleichnamige Dateien identisch sind, wird in der Regel kein Duplikat gespeichert, sondern nur eine der beiden gleichen Dateien aufgehoben.

Dateityp-Einschränkungen: Falls Sie für ein File-Upload-Feld ein `accept`-Attribut angegeben haben, das neben bestimmten Dateitypen (wie PDF) auch ZIP-Dateien erlaubt, ändert sich obige Semantik des ZIP-Uploads wie folgt: Aus der hochgeladenen ZIP-Datei werden nur diejenigen Dateien zur bestehenden Einsendung hinzugefügt, deren Dateiendung in der `accept`-Liste aufgeführt ist (ohne `.zip`). (Mime-Type-Angaben im `accept`-Attribut werden beim ZIP-Upload ignoriert, da die Mime-Typen von Dateien im ZIP nicht direkt zur Verfügung stehen.) Beispiel: Falls Sie `accept=".jpg,.jpeg,.gif,.png,image/*",.zip,application/zip` im Dateiupload-Feld annotiert haben, so wird durch Mime-Typ `image/*` zwar der Browser angewiesen, bei Einzel-Dateiuploads beliebige Bilddateien zu akzeptieren, aus hochgeladenen ZIP-Dateien dagegen werden nur die Dateien mit den explizit aufgezählten Dateiendungen `.jpg`, `.jpeg`, `.gif` und `.png` (ohne Beachtung der Groß-/Kleinschreibung) entpackt und der Einsendung hinzugefügt.

Zurücksetzen/Löschen: Da durch den `einsendenPlusX`-Button nur neue Dateieinsendungen hinzugefügt werden können, jedoch keine früheren Einsendungen mehr überschrieben werden, sollte den Studenten zusätzlich ein Löschen-Button zum Zurücksetzen der bisherigen Einsendungen angeboten werden, damit sie wenigstens im Zweifel alles bisher Hochgeladene wieder löschen/zurückziehen und neu beginnen können.

Zu diesem Zweck kann ein zweiter Submit-Button mit Namen `einsendenLeerX` zur selben Teilaufgabe hinzugefügt werden. Sein allgemeines Verhalten wurde schon unter Spezielle Submit-

Buttons erwähnt. In diesem Kontext führt er dazu, dass die bestehende (Multi-)Dateieinsendung durch eine leere Einsendung überschrieben wird, also alle bisher eingesendeten Dateien effektiv wieder gelöscht werden.

Es ist sinnvoll, einen solchen Löschen-Button nur dann anzuzeigen, wenn überhaupt eine nicht-leere Einsendung in diesem Feld vorliegt, was über die Kontrollvariable `$IfNotEmptyXY` (mit `X` = Teilaufgabenkennung (Großbuchstabe) und `Y` = Feldnummer) gesteuert werden kann.

Beispiel für eine Teilaufgabe (hier mit Kennung `D`, also als vierte Teilaufgabe eines Gesamt-Aufgabenformulars) zum Multi-Dateiupload mit Anzeige der vorliegenden Einsendung (auch, falls sie leer ist) und Löschen-Button (nur falls eine nicht-leere Einsendung vorliegt):

```
<p>Dateien hochladen:
<input type="file" name="FeldD1" multiple data-ignore="empty" data-if-
empty="Keine Dateien hochgeladen oder alle wieder gelöscht!">
<button type="submit" name="einsendenPlusD">Datei(en) hinzufügen</button><br>
$IfExistsD1
  Ihre bisher hochgeladenen Dateien: $FeldD1
  $IfNotEmptyD1
    <button type="submit" name="einsendenLeerD"
            onclick="return confirm('Wirklich alle bisher hochgeladenen Dateien
wieder zurückziehen? (Sie können anschließend weiterhin neue Dateien
hochladen.)');">
            Löschen</button>
  $/IfNotEmptyD1
$/IfExistsD1
```

Auch hier haben wir per `data-ignore="empty"` festgelegt, dass die Aufgabe nach Drücken des Löschen-Buttons wieder als (von diesem Studenten) unbearbeitet gelten soll (sofern nicht zu anderen Teilaufgaben zu bewertende Eingaben enthalten).

n Dateien als Sammel-Upload (Multi-File-Submit)

Der Vollständigkeit halber sei noch eine dritte Möglichkeit erwähnt. Wie im ersten Fall geht diese von einer festen Anzahl von Dateiupload-Formularfeldern aus, über die also maximal eine bestimmte Anzahl *n* von Dateien eingesendet werden kann. Im Unterschied zu beiden vorhergehenden Varianten bilden alle *n* File-Inputs aber eine einzige Teilaufgabe, die Dateien werden also immer zusammen „als Einheit“ eingesendet. Wenn Sie nun auch noch allen File-Inputs denselben Namen geben, diese also aus Übungssystem-Sicht zu einem einzigen Eingabefeld zusammenfassen (analog zu Checkbox- oder Radiobutton-Inputs), so wird beim Betätigen des `einsendenX`-Buttons genau eine Dateieinsendung erzeugt, die alle in diesen Feldern hochgeladenen Einzeldateien umfasst. Ähnlich wie beim oben beschriebenen Multi-Dateiupload werden dabei alle hochgeladenen Dateien (sofern es mindestens zwei sind) zu einer ZIP-Datei zusammengefasst.

Beispiel für bis zu *n* = 3 hochzuladende Dateien:

```
<form method="POST"

action="$WebAssignServer/$Veranstaltername/EinsendungMultipart/$KursNr/$Version
sNr/$AufgabenheftNr/$AufgabenNr/"
      enctype="multipart/form-data">
    <p>Wählen Sie bis zu drei Dateien aus, die Sie einsenden wollen
       (eine frühere Einsendung wird komplett überschrieben):</p>
    <p>
        <input type="file" name="FeldA1" data-ignore="empty" data-if-
empty="Keine Datei hochgeladen"><br>
        <input type="file" name="FeldA1"><br>
        <input type="file" name="FeldA1"><br>
        <input type="submit" name="einsendenA"  value="einsenden"><br>
        $IfExistsA1(Ihre letzte Einsendung: )$FeldA1
    </p>
</form>
```

Auch hier haben wir wieder per `data-ignore="empty"` festgelegt, dass die Aufgabe als unbearbeitet interpretiert werden soll, falls keine einzige Datei hochgeladen wurde. Es sei nochmals daran erinnert, dass im Fall von mehreren Input-Tags zum selben Feldnamen (wie hier) die Data-Attribute nur an einem dieser Tags (einmal pro Feldname) anzugeben sind.

Beliebig viele Dateien als Sammel-Upload (Multi-File-Submit)

Eine Variante des letzten Abschnitts ist die Möglichkeit, nicht eine feste Anzahl von mehreren File-Input-Feldern in die Aufgabenseite einzubauen, sondern ein einziges Input mit `multiple`-Attribut: In einem solchen Input kann der Nutzer mehr als eine Datei auswählen. Tut er das und lädt in einem Schritt mehrere Dateien hoch, werden diese vom Online-Übungssystem ebenfalls zu einer ZIP-Datei zusammengefasst.

Beispiel:

```
<form method="POST"
action="$WebAssignServer/$Veranstaltername/EinsendungMultipart/$KursNr/$Version
sNr/$AufgabenheftNr/$AufgabenNr/" enctype="multipart/form-data">
<p>Wählen eine oder mehrere Dateien aus, die Sie einsenden wollen (eine frühere
Einsendung wird komplett überschrieben):</p>
<p>
  <input type="file" name="FeldA1" multiple data-ignore="empty" data-if-
empty="Keine Dateien hochgeladen!"><br>
  <input type="submit" name="einsendenA"  value="einsenden"><br>

  $IfExistsA1(Ihre letzte Einsendung: )$FeldA1
</p>
</form>
```

Dieses Beispiel verhält sich praktisch wie ein einfacher Dateiupload, nur dass eben mehrere Dateien auf einmal hochgeladen werden können. Jeder neue Upload ersetzt den vorherigen (also alle zuvor hochgeladenen) Dateien komplett.

Im Normalfall würden wir statt dessen aber eher den oben beschriebenen mehrstufigen Multi-Dateiupload vorziehen, der auch das Hochladen in mehreren Schritten, also das spätere Hinzufügen von Dateien erlaubt.

Formular mit Randomisierung

Random-Blöcke zur Fragen-Randomisierung

Falls Sie die Fragen-Randomisierung <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#fragenrandom> nutzen möchten, also Ihr Aufgabenformular in mehrere Fragen zerlegen möchten, von denen jedem Teilnehmer eine (oder eine echte Teilmenge) ausgewählt und angezeigt werden soll, oder die für jeden Teilnehmer in eine individuelle Reihenfolge gebracht („gemischt“) werden sollen, dann gehen Sie wie folgt vor:

Leiten Sie jede dieser Fragen mit einer Variablen des Typs `$Random1` ein und beenden Sie sie mit `$/Random1`. Einen solchen mit diesen zwei Variablen eingeschlossenen Teilbereich des Formulars bezeichnen wir im Folgenden als „Random-Block“ oder kurz „Block“.
Die Nummer (hier: `1`) muss in beiden Variablen eines Blocks identisch sein, sich aber von Block zu Block unterscheiden, d.h. sie identifiziert den jeweiligen Block. Naheliegend (aber nicht notwendig) ist eine fortlaufende Nummerierung der Blöcke, was dann z.B. wie folgt aussehen könnte:

```
<h1>$Aufgabenname $MaximalPunkteUeberschrift</h1>
$EMBED
<form method="POST"
action="$WebAssignServer/$Veranstaltername/Einsendung/$KursNr/$VersionsNr/$Aufg
abenheftNr/$AufgabenNr/">

 <p>Einleitender Text für alle Teilnehmer</p>
 $Random1
 <h2>Überschrift erste Frage</h2>
 <p>Aufgabentext erste Frage</p>
 <textarea name="FeldA1" data-ignore="empty"  class="fullwidth" rows="25"
cols="100"></textarea>
 <button type="submit" name="einsendenA">Eingaben einsenden</button>
 $/Random1

 $Random2
 <h2>Überschrift zweite Frage</h2>
 <p>Aufgabentext zweite Frage</p>
 <textarea name="FeldB1" data-ignore="empty"  class="fullwidth" rows="25"
cols="100"></textarea>
 <button type="submit" name="einsendenB">Eingaben einsenden</button>
 $/Random2

 … ggf. weitere Fragen analog
</form>
$/EMBED
```

(Die Nummerierung der Random-Blöcke ist prinzipiell beliebig, muss insbesondere weder lückenlos noch ab 1 beginnend noch aufsteigend sortiert sein. Der erste Block könnte z.B. auch mit `$Random42`…`$/Random42` umschlossen sein.)

Sofern Sie, wie in obigem Beispiel, „nur“ Random-Blöcke ohne zusätzliche weitere (i.F. beschriebene) Konfigurationsmaßnahmen erzeugen, wird jedem Teilnehmer *genau ein* individuell für ihn zufällig ausgewählter Block präsentiert und alle weiteren Random-Blöcke werden ausgeblendet.

Falls Sie *mehr als einen* Block pro Teilnehmer auswählen lassen möchten, z.B. zwei zufällig ausgewählte Blöcke aus vieren, dann fügen Sie im Aufgabenformular *vor dem ersten Random-*

Block[12] die folgende Variable ein: `$Randomize(2)`. Die Zahl in Klammern gibt an, wieviele Fragen/Blöcke pro Teilnehmer ausgewählt und präsentiert werden sollen.

Falls Sie möchten, dass auch die *Fragenreihenfolge* der ausgewählten/angezeigten Fragen nicht immer der Reihenfolge in Ihrer Aufgabenseite entspricht, sondern für jeden Teilnehmer eine individuelle Reihenfolge „gewürfelt" wird, ergänzen Sie die Randomize-Deklaration um ein Argument „Shuffle" wie folgt: `$Randomize(2, Shuffle)`.

Falls *ausschließlich* die Fragenreihenfolge randomisieren wollen, also zwar jedem Teilnehmer stets *alle* Fragen stellen möchten, aber diese in zufälliger Reihenfolge, gehen Sie genauso vor, d.h. fassen Sie alle Fragen in Random-Blöcke ein und stellen Sie dem ersten Random-Block eine solche Randomize-Variable voran, in der Sie die Fragenzahl aufs Maximum setzen und das Shuffle-Argument hinzufügen. Enthält Ihre Seite z.B. genau 5 Fragen, so wird `$Randomize(5, Shuffle)` „zufällig 5 aus den 5 Fragen auswählen", also effektiv immer alle Fragen anzeigen, aber eben in zufälliger Reihenfolge.

Zwischen zwei Random-Blöcken (in obigem Beispiel also zwischen `$/Random1` und `$Random2`) steht im Regenfall kein Inhalt. Allgemein gilt, dass jeglicher Inhalt, der *nicht* innerhalb eines Random-Blocks steht, allen Teilnehmern immer angezeigt wird. Das gilt auch für Text zwischen Random-Blöcken: Enthielte also obiges Beispiel einen Text zwischen `$/Random1` und `$Random2`, so würde dieser Text den Teilnehmern, denen Frage 1 gestellt wird, *hinter* dieser ausgewählten Frage angezeigt, während er denjenigen Teilnehmern, denen Frage 2 gestellt wird, *vor* der Frage angezeigt würde.

Falls Sie den in Abschnitt »Unterschiedliche Fragenauswahl innerhalb eines Hefts erzwingen« <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#randomizeheft> beschriebenen Modus aktivieren wollen, der verhindern soll, dass einem Teilnehmer nicht in zwei oder mehr Aufgaben desselben Aufgabenhefts (die gleich viele Fragen enthalten und gleiche Randomisierungseinstellungen verwenden) zufällig genau dieselbe(n) Frage(n) (mit der selben Nummer hinter der `$Random…`-Variable) ausgewählt werden, dann ergänzen Sie die Randomize-Variable jeweils um den Namenszusatz „Heft", d.h. fügen Sie im Formular noch vor der ersten `$Random1`-Variable die Variable `$RandomizeHeft` statt `$Randomize` ein. Die Angaben in Klammern dahinter sind identisch zur Randomize-Variable, also z.B. `$RandomizeHeft(2, Shuffle)` aktiviert die zufällige Auswahl von 2 Fragen in Zufallsreihenfolge mit der Besonderheit, dass in mehreren Aufgaben desselben Hefts, die exakt gleich viele Random-Blöcke enthalten und ebenfalls diese Variable steht, also ebenfalls zwei Fragen gewählt werden sollen, versucht wird, jeweils eine andere Auswahl zu treffen (sofern möglich[13]). Die `$RandomizeHeft`-Variable kann auch ganz ohne weitere Parameter in Klammern angegeben werden. Dann werden wieder die Grundeinstellungen (zufällige Auswahl genau eines der Random-Blöcke) verwendet, nur eben mit der Besonderheit, dass bei mehreren Aufgaben mit gleicher Randomblock-Anzahl und Verwendung dieser Variablen eben versucht wird, in jeder Aufgabe einen anderen Random-Block zu wählen[13].

Teilaufgaben

In obigem Beispiel wurde für jede Frage eine eigene Teilaufgabe gewählt (Teilaufgabe `A` für das Eingabefeld und den Submit-Button der ersten Frage, Teilaufgabe `B` entsprechend für die zweite Frage). Das ist nicht unbedingt nötig, man könnte auch alle Felder einer einzigen Teilaufgabe unterordnen und einen einzigen Submit-Button dazu am Ende (hinter dem letzten `$/Random`) platzieren.
Aber *die Unterteilung in mindestens eine Teilaufgabe pro Random-Block, keine Verteilung von Feldern einer Teilaufgabe über mehrere Random-Blöcke) wird dringend empfohlen!*
Denn bei einer Teilaufgabe über mehrere/alle Random-Blöcke würden immer für jede Einsendung zu dieser Teilaufgabe auch wirklich Einsendungen für jedes Feld der Teilaufgabe in der Datenbank gespeichert. Für jedes nicht sichtbare (weil in einem ausgeblendeten Random-Block liegende) Eingabefeld würde also dennoch eine leere Einsendung erzeugt – und auch in der Quittungsseite quittiert. Getrennte Teilaufgaben ermöglichen dagegen Einsendungen *ausschließlich* derjenigen Felder, die dem Studenten auch angezeigt / ausgewählt wurden, und entsprechend auch Quittierung der Eingabe genau für die zuvor gezeigten Eingabefelder (und keine Bestätigung von Leereingaben in Feldern, die dem Studenten nie gezeigt wurden).

Auch bei Verwendung von Autokorrektur ist die Unterteilung in Teilaufgaben in der Regel sehr wichtig, damit für jede Frage das Korrekturmodul immer nur die Eingaben zu dieser Frage übermittelt bekommt und nicht zusätzliche „Leereingaben“ zu den ausgeblendeten Fragen. Natürlich hängt es im Zweifel vom Korrekturmodul ab, ob das einen Unterschied macht. Der *interne Aufgabenbewerter* z.B. würde hier – wenn alle Fragen zu einer Teilaufgaben zusammengefasst wären – eine Autokorrektur erzeugen, die so aussieht, als seien dem Teilnehmer alle Fragen angezeigt worden und er habe nur zu den nicht ausgewählten Fragen keine Eingaben gemacht. Er bekäme daher eine Korrektur für jede Frage, auch für die ihm gar nicht gestellten Fragen, und dabei könnte er – zumindest bei Multiple-Choice-Fragen – ggf. sogar noch Teilpunkte für diese Fragen bekommen!

Fazit: Erstellen Sie am besten *immer* genau eine Teilaufgabe für alle Nicht-Dateiupload-Felder einer Frage (sowie eine pro Dateiupload).

Normalerweise kann allerdings immer nur *eine* Teilaufgabe gleichzeitig eingesendet werden. Bei einer **Randomisierung mit Auswahl *genau einer* Frage pro Aufgabenseite** ist das auch gar kein Problem: Erstellen Sie eine Teilaufgabe pro Frage, dem Studenten wird genau eine Frage und damit genau eine Teilaufgabe angezeigt, er sendet zu dieser einen Teilaufgabe seine Eingaben ein (und weiß gar nicht, dass es in der Aufgabeneinrichtung noch weitere Teilaufgaben gäbe, da sie ja vor ihm versteckt sind), erhält eine Quittung zur bearbeiteten Teilaufgabe etc.

Natürlich muss dazu jeder Random-Block neben den Eingabefeldern der Teilaufgabe auch einen entsprechenden Einsendebutton für diese Teilaufgabe enthalten, genau wie im Beispiellisting oben: Dort enthält z.B. der `$Random2`-Block ein Eingabefeld `FeldB1` und einen Submitbutton `einsendenB` dazu.

Bei einer **Randomisierung mit Auswahl *mehrerer* Fragen pro Aufgabenseite** sieht die Sache dagegen anders aus. Auch hier wird wieder empfohlen, eine (oder bei Bedarf mehrere) Teilaufgaben pro Frage zu bilden, aber die Frage ist, wie diese einzusenden sind.

Es gibt da prinzipiell zwei Möglichkeiten.

Die *erste* ist simpel: Es könnte nach wie vor jeder Random-Block neben den Eingabefeldern zu seiner Teilaufgabe eine eigenen Submitbutton enthalten, z.B.:

```
<h1>$Aufgabenname $MaximalPunkteUeberschrift</h1>
$EMBED
<form method="POST"
action="$WebAssignServer/$Veranstaltername/Einsendung/$KursNr/$VersionsNr/$Aufg
abenheftNr/$AufgabenNr/">

 <p>ACHTUNG, Negativbeispiel! Nicht zur Nachahmung empfohlen!</p>

 $Randomize(2)
 $Random1
 <h2>Überschrift erste Frage</h2>
 <p>Aufgabentext erste Frage</p>
 <input type="text" name="FeldA1" data-ignore="empty"  size="100">
 <button type="submit" name="einsendenA">Antwort 1 einsenden</button>
 $/Random1

 $Random2
 <h2>Überschrift zweite Frage</h2>
 <p>Aufgabentext zweite Frage</p>
 <input type="text" name="FeldB1" data-ignore="empty"  size="100">
 <button type="submit" name="einsendenB">Antwort 2 einsenden</button>
 $/Random2

 $Random3
 <h2>Überschrift dritte Frage</h2>
 <p>Aufgabentext dritte Frage</p>
 <input type="text" name="FeldC1" data-ignore="empty"  size="100">
 <button type="submit" name="einsendenC">Antwort 3 einsenden</button>
 $/Random3

 …
</form>
$/EMBED
```

Wenn die Inputs oben File-Inputs wären und jeder Submit-Button dazu der zugehörigen Upload-Button, entspräche das durchaus dem normalen und empfohlenen Vorgehen, da jedes Dateiupload-Feld immer seinen eigenen Submit-Button haben sollte (s.o.).

Aber für sonstige Eingabefelder, insbesondere für Textfelder wie oben wird diese Lösung explizit **nicht** empfohlen! Sie hat nämlich zwei große Nachteile in der Usability für die Studenten:

1. Jeder Student müsste jede Frage einzeln einsenden, d.h., falls ihm z.B. die erste und die dritte Frage ausgewählt wurden, erst die Antwort auf die erste Frage eingeben, mit dem Button »Antwort 1 einsenden« abschicken, von der Quittung zur Aufgabenseite zurückkehren, dann die Antwort auf Frage 3 eingeben und per »Antwort 3 einsenden« dann abschicken (oder umgekehrt).
2. Hinzu kommt das Problem, dass von mehreren Submitbuttons im selben Aufgabenformular immer der erste zum *Default Button* wird. In Textareas spielt das keine Rolle, aber falls Ihr Formular wie das obige Beispiel einfache Eingabefelder (`input`) enthält, gilt: Wann immer der Student bei fokussiertem Input die Enter-Taste drückt, wird der erste Submitbutton ausgelöst. In obigem Beispiel hieße dies: Gibt der Student im Eingabefeld von z.B. Frage 3 einen Text ein und drückt Enter, wird Teilaufgabe 1 eingesendet und seine Eingabe zu Frage 3 nicht gespeichert! Das ist offensichtlich nicht sinnvoll und sollte so daher nicht angeboten werden.

Daher wurde speziell für diese Randomisierung mit Auswahl *mehrerer* Fragen, die sich über *mehrere Teilaufgaben* erstrecken, eine neue Einsendemöglichkeit geschaffen, die eine (vom Randomisierer vorgenommene) *Auswahl/Selektion* von Teilaufgaben in einem einzigen Schritt einsenden kann.

Selektionssubmit über mehrere Teilaufgaben

Der Selektionssubmit hat zum Ziel, dass Sie Ihre Aufgabenseite ähnlich aufbauen können wie für unrandomisierte Aufgaben aus genau einer Teilaufgabe: Sie fügen Random-Blöcke mit mehreren Eingabefeldern hinzu, die zu mehreren Teilaufgaben zugeordnet werden, am Ende der Seite aber nur *einen* Einsendebutton (nicht Teil eines Random-Blocks), der zum Einsenden aller Eingaben in den sichtbaren Eingabefelder bzw. derer Teilaufgaben dient.

Innerhalb der Random-Blöcke fügen Sie an Stelle von eigenen Submit-Buttons spezielle Hidden-Inputs ein mit dem Schlüssel `Selektion` und der Teilaufgabenbezeichnung (Buchstabe) als Wert. Der Schlüssel des Submit-Buttons beginnt wie gewohnt mit `einsenden`, allerdings wird an Stelle einer Teilaufgabenbezeichnung das Suffix `Selektion` angehängt:

```
<h1>$Aufgabenname $MaximalPunkteUeberschrift</h1>
$EMBED
<form method="POST"
action="$WebAssignServer/$Veranstaltername/Einsendung/$KursNr/$VersionsNr/$Aufg
abenheftNr/$AufgabenNr/">

 <p>Einleitender Text für alle Teilnehmer</p>

 $Randomize(2)
 $Random1
 <h2>Überschrift erste Frage</h2>
 <p>Aufgabentext erste Frage</p>
 <input type="text" name="FeldA1" data-ignore="empty"  size="100">
 <input type="hidden" name="Selektion" value="A">
 $/Random1

 $Random2
 <h2>Überschrift zweite Frage</h2>
 <p>Aufgabentext zweite Frage</p>
 <input type="text" name="FeldB1" data-ignore="empty"  size="100">
 <input type="hidden" name="Selektion" value="B">
 $/Random2

 $Random3
 <h2>Überschrift dritte Frage</h2>
 <p>Aufgabentext dritte Frage</p>
 <input type="text" name="FeldC1" data-ignore="empty"  size="100">
 <input type="hidden" name="Selektion" value="C">
 $/Random3

 <p>
 <button type="submit" class="large" name="einsendenSelektion">Eingaben
einsenden</button>
 </p>
</form>
$/EMBED
```

Die Funktionsweise ist zunächst relativ einfach: Wenn der `einsendenSelektion`-Button gedrückt wird, werden insbesondere auch die `Selektion`-Felder zu allen Random-Blöcken eingesendet. Hat in obigem Beispiel die Randomisierung einem Studenten z.B. die Blöcke 1 und 3 (also die Teilaufgaben A und C) präsentiert, so wird unter dem Schlüssel `Selektion` dem Übungssystem die Information `A,C` übermittelt. Und das Übungssystem wird daraufhin in einem einzigen Schritt Einsendungen zu allen Eingabefeldern der Teilaufgaben A und C erstellen, aber zur (dem Studenten nicht angezeigten) Teilaufgabe B keine Leereinsendungen erzeugen.

Kombination von Selektionssubmit und File-Submits:

Wie bereits mehrfach betont, sollten Dateiuploads immer eine eigene Teilaufgabe bilden, mit einem eigenen `input type="file"` und einem eigenen Submit-Button (Upload-Button) dazu. Diese Teilaufgaben gehören dann natürlich nicht mit zu den „selektierteren" Teilaufgaben, sollen also *nicht* mit dem `einsendenSelektion`-Button erfasst werden.

Passen wir obiges Beispiel abschließend nochmal ein wenig an, und zwar so, dass es aus vier Fragen besteht, wobei…

- Fragen 1 und 2 jeweils nur aus einem Dateiuploadfeld bestehen (jeweils 1 Teilaufgabe),
- Frage 3 ein Dateiupload- und ein Textfeld umfasst (2 Teilaufgaben) und
- Frage 4 zwei Textfelder umfasst (1 Teilaufgabe):

```
<h1>$Aufgabenname $MaximalPunkteUeberschrift</h1>
$EMBED
<form method="POST"
action="$WebAssignServer/$Veranstaltername/Einsendung/$KursNr/$VersionsNr/$Aufg
abenheftNr/$AufgabenNr/">

 <p>Einleitender Text für alle Teilnehmer</p>

 $Randomize(2)
 $Random1
 <h2>Erste Frage</h2>
 <p>Datei: <input type="file" name="FeldA1" data-if-empty="Keine Datei
eingesendet!" data-ignore="empty">
 <button type="submit" name="einsendenA">hochladen</button><br>
 $IfExistsA1(Ihre zuletzt hochgeladene Datei: )$FeldA1HREF</p>
 $/Random1

 $Random2
 <h2>Zweite Frage</h2>
 <p>Datei: <input type="file" name="FeldB1" data-if-empty="Keine Datei
eingesendet!" data-ignore="empty">
 <button type="submit" name="einsendenB">hochladen</button><br>
 $IfExistsB1(Ihre zuletzt hochgeladene Datei: )$FeldB1HREF</p>
 $/Random2

 $Random3
 <h2>Dritte Frage</h2>
 <p>Datei: <input type="file" name="FeldC1" data-if-empty="Keine Datei
eingesendet!" data-ignore="empty">
 <button type="submit" name="einsendenC">hochladen</button><br>
 $IfExistsC1(Ihre zuletzt hochgeladene Datei: )$FeldC1HREF</p>
 <p>Und eine Texteingabe:
  <input type="text" name="FeldD1" data-ignore="empty"  size="100">
 </p>
 <input type="hidden" name="Selektion" value="D">
 $/Random3

 $Random4
 <h2>Vierte Frage</h2>
 <p>Eingabe 1:
  <input type="text" name="FeldE1" data-ignore="empty"  size="100">
 </p>
 <p>Eingabe 2:
  <input type="text" name="FeldE2" data-ignore="empty"  size="100">
 </p>
 <input type="hidden" name="Selektion" value="E">
 $/Random4

 <p>
 <button type="submit" class="large" name="einsendenSelektion">Eingaben
einsenden</button>
 </p>
</form>
$/EMBED
```

Nun kann es zu verschiedenen Konstellationen kommen, wenn einem Teilnehmer zwei dieser vier

Fragen angezeigt werden:

1. Die Auswahl besteht aus den Random-Blöcken 3 und 4. Dann sieht der Student zwei Submit-Buttons, einen zum »hochladen« der Datei zur dritten Frage (Teilaufgabe C) und den »Eingaben einsenden«-Button am Ende zum Einsenden der Texteingaben zu beiden Fragen 3 und 4 (Selektion = Teilaufgaben D und E).
2. Die Auswahl enthält Block 3 *oder* 4, sowie einen der beiden Blöcke 1 *oder* 2. In dem Fall kann der Student zu jedem Datei-Auswahlfeld jeweils den separaten Upload-Button betätigen und der abschließende »Eingaben einsenden«-Button sendet den Inhalt des Textfeldes aus Block 3 bzw. 4 ein (Selektion = Teilaufgabe D oder E).
3. Die Auswahl besteht aus den Random-Blöcken 1 und 2. Dies ist ein besonderer Fall, denn in diesem Fall „ist die Selektion leer", sämtliche Hidden-Fields mit `Selektion`-Schlüssel wurden vom Randomizer entfernt. In diesem Fall entfernt der Randomizer automatisch auch den (in diesem Fall sonst wirkungslosen) `einsendenSelektion`-Button, d.h. der Student sieht in diesem Fall wirklich nur die Dateiupload-Felder der Blöcke 1 und 2 mit jeweils einem Upload-Button dazu.

Selektionssubmit mit Speicherung der Selektion

Im Normalfall, so wie im vorigen Abschnitt eingeführt, ist die Information aus den `Selektion`-Feldern flüchtig, d.h. bei Einsendung verwendet das Online-Übungssystem diese Aufzählung der eingesendeten (von der Randomisierung ausgewählten) Teilaufgaben nur, um die Einsendung auszuwerten und die Eingaben nur zu genau diesen Teilaufgaben zu speichern, also keine Leereinsendungen zu den ausgefilterten Teilaufgaben zu erzeugen.

Nach dieser Speicherung der Einsendungen zu den Teilaufgaben ist diese „Selektion" dann nicht mehr rekonstruierbar, insbesondere wenn eine Randomisierung mit „Shuffle", also Zufallsreihenfolge verwendet wird, wird die Information über diese für einen Studierenden „gewürfelte" individuelle Reihenfolge nicht gesichert. Sie steht den Korrekturmodulen nicht zur Verfügung.

Bei den mit dem internen (Vor-)Korrekturmodul ausgewerteten / automatisch korrigierten Aufgaben hatte das bei Shuffle-Aufgaben z.B. folgende Auswirkungen:

- Bei Vorkorrektur, die ja direkt bei der Einsendung als Sofortfeedback ausgeführt und in die Quittung eingebunden wird, weiß zwar der als Vorkorrekturmodul eingesetzte »Teilaufgabenbewerter« auch selbst nichts über die Reihenfolge, aber er wird ja ohnehin für jede Teilaufgabe einzeln ausgeführt, und es ist möglich, die einzelnen Vorkorrekturen zu den jeweiligen Teilaufgaben in der Quittungsseite wieder mit Hilfe von Random-Variablen in dieselbe Reihenfolge zu bringen, siehe Quittungen bei Randomisierung / Selektionssubmit.
- Die normale, erst beim Heftschließen (durchs Korrekturmodul »Aufgabenbewerter«) ausgeführte Autokorrektur dagegen bewertet immer alle vorliegenden Einsendungen zu allen Teilaufgaben, hatte aber bei dieser Technik bislang keine Kenntnis über die Reihenfolge, in der die Teilaufgaben jeweils in der Aufgabenseite eines konkreten Studierenden angeordnet waren. Daher war bis Juli 2023 jede Autokorrektur immer in der festen Reihenfolge, in der die Fragen in der Aufgaben-HTML-Seite stehen (bzw. im Aufgabeneditor angeordnet sind).

Um den Nachteil des letzten genannten Punktes zu beheben, es also dem internen Korrekturmodul (»Aufgabenbewerter«) – aber prinzipiell auch jedem selbst entwickelten externen Korrekturmodul – zu ermöglichen, seine Ausgaben an die Shuffle-Reihenfolge der Aufgabenseite anzupassen, wurde der Selektionssubmit um die Möglichkeit erweitert, die eingesendete Selektion (also Aufzählung von Teilaufgabenbuchstaben der von der Randomisierung ausgewählten Teilaufgaben, ggf. in der von der Randomisierung gewürfelten Zufallereihenfolge) zusammen mit den Einsendungen in der Datenbank

zu speichern, damit sie dann zum Zeitpunkt des Heftschließens (und damit der Autokorrektur) zusammen mit den eigentlichen Einsendungen ans Korrekturmodul übermittelt werden kann.

Um diese Speicherung der Selektion anzufordern, ist in der Aufgabenseite nur eine einzige Änderung notwendig: *Der Name des Selektions-Submitbuttons ist dazu um den Zusatz „Speichern" zu ergänzen*, d.h. statt `einsendenSelektion` ist einfach `einsendenSelektionSpeichern` einzutragen, z.B.

```
 <button type="submit" class="large"
name="einsendenSelektionSpeichern">Eingaben einsenden</button>
 </p>
```

Die Auswirkung ist, dass dann nicht nur zu jeder selektierten Teilaufgabe eine Einsendung mit den jeweiligen studentischen Eingaben gesichert wird, sondern noch eine Einsendung zur „Pseudo-Teilaufgabe" `$`[14] und Feldnummer 1 erzeugt (also so, als sei diese Selektion in ein Eingabefeld namens `Feld$1` eingegeben worden).

Ein Korrekturmodul (kein Vorkorrekturmodul, das ja nur pro Teilaufgabe ausgeführt wird) bekommt dann neben allen Einsendungen zu allen echten Teilaufgaben eben auch diese Selektions-Information als Einsendung zu Pseudo-Teilaufgabe `$`, Feld `1` übermittelt und kann diese wahlweise ignorieren oder eben entsprechend als Information über die Randomisierung der Aufgabenseite interpretieren. Der interne »Aufgabenbewerter« tut genau das: Wenn er diese Selektionseinsendung vorfindet, gibt er zunächst alle Autokorrekturen zu genau den darin genannten Teilaufgaben in der entsprechenden Selektionsreihenfolge aus. (Sollten darüber hinaus noch weitere Einsendungen zu Teilaufgaben existieren, die in der Selektion *nicht* genannt sind, werden diese im Anschluss auch noch ausgegeben. Im Normalfall sollte das nicht vorkommen, ist aber prinzipiell z.B. denkbar, wenn die Aufgabe nachbearbeitet wurde, nachdem schon Einsendungen vorlagen.)

Blocklokale Randomisierung

Bisher wurde nur der Fall betrachtet, dass die gesamte Aufgabenseite in Random-Blöcke unterteilt wird, die in ihrer Reihenfolge gemischt werden und/oder von denen eine zufällige Teilauswahl getroffen werden kann. Das Anwendungsgebiet dafür ist typischerweise die Untergliederung von Aufgaben in Fragen <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#fragen> .

Ab Sommer 2021 bestehen aber auch weitergehende Randomisierungsmöglichkeiten, indem insbesondere *innerhalb* von solchen Fragen (Random-Blöcken) wieder Subblöcke / Unterbereiche gebildet und ihrerseits randomisiert werden können. Typisches Anwendungsgebiet dafür ist z.B. die Randomisierung einzelner Antwortoptionen innerhalb einer Multiple-Choice-Frage, d.h. Anzeigen der Antwortoptionen in einer zufälligen / studentenindividuellen Reihenfolge und/oder Auswahl einer Teilmenge von in der Aufgabe definierten Antwortoptionen.

Das war bisher nicht möglich, denn dazu muss ein Random-Block seinerseits nochmals Random-Blöcke enthalten können. Nun ist also eine Schachtelung möglich, sogar beliebiger Tiefe (d.h. Unter-Random-Blöcke dürfen ihrerseits nochmals Unter-Unter-Randomblöcke enthalten etc.).

Für den Fall, dass Sie zwar fragenlokale Randomisierung (z.B. der Antwortoptionen von Multiple-Choice-Fragen) nutzen möchten, die eigentlichen Fragenblöcke aber selbst *nicht* randomisiert werden sollen, ist das auch kein Problem: Zu diesem Zweck wurden neben Random- auch neue, sog. Fixed-Blöcke eingeführt, die selbst nicht randomisiert werden, aber eben einen Block bilden, innerhalb dessen eine lokale Randomisierung stattfinden kann.

Beispiel 1: Multiple-Choice-Fragen randomisieren

Nehmen Sie an, eine Aufgabenseite soll aus zwei Multiple-Choice-Fragen „x aus 5“ in fester Reihenfolge (erste und zweite Frage) bestehen, aber die Antwortalternativen *pro Frage* sollen jeweils untereinander gemischt werden. Dann können Sie nicht einfach jede der insgesamt 10 Antwortoptionen in einen „normalen“ Random-Block wie eingangs beschrieben packen, denn dann würden ja alle 10 Optionen beliebig untereinander vermischt! Vielmehr soll für jede der beiden Fragen eine *separate* Randomisierung stattfinden, wozu eben jede Frage in einen Fixed-Block eingefasst wird (Fixed deshalb, da die Fragen selbst ja laut der Annahme eingangs nicht randomisiert werden sollen) und innerhalb jedes der beiden Blöcke, also für jede Frage separat, eine blocklokale Randomisierung (in diesem Fall eine Shuffle-Only-Randomisierung) angewandt wird.

Das eben besagte Beispiel könnte in etwa wie folgt aussehen. (Dabei halten wir den HTML-Code absichtlich simpel, der Aufgabenerstellungsassistent generiert – auch im Hinblick auf Barrierefreiheit / Screenreader-Tauglichkeit – deutlich komplexeres HTML-Markup. Aber hier soll ja im Wesentlichen der Randomisierungs-Code im Vordergrund stehen und nicht optimales HTML-Markup für eine Multiple-Choice-Frage.)

```
$Fixed1
  <h2>Erste Frage</h2>
  $Randomize(5, Shuffle)
  $SetCounter.1(A)
  $Random.1
    <label for="mcA1A">$Counter.1</label>
    <input  id="mcA1A" type="checkbox" name="FeldA1" value="_A" data-
ignore="empty">
    <input type="hidden" name="FeldA2" value="_A/$Counter.1" class="no-restore"
data-ignore="always">
    …Antworttext zur ersten Antwortalternative…
  $/Random.1

  $Random.2
    <label for="mcA1B">$Counter.1</label>
    <input  id="mcA1B" type="checkbox" name="FeldA1" value="_B">
    <input type="hidden" name="FeldA2" value="_B/$Counter.1" class="no-
restore">
    …Antworttext zur zweiten Antwortalternative…
  $/Random.2

  …

  $Random.5
    <label for="mcA1E">$Counter.1</label>
    <input  id="mcA1E" type="checkbox" name="FeldA1" value="_E">
    <input type="hidden" name="FeldA2" value="_E/$Counter.1" class="no-
restore">
    …Antworttext zur fünften Antwortalternative…
  $/Random.5
$/Fixed1

$Fixed2
  <h2>Zweite Frage</h2>
  $Randomize(5, Shuffle)
  $SetCounter.1(A)
  $Random.1
    <label for="mcA3A">$Counter.1</label>
    <input  id="mcA3A" type="checkbox" name="FeldA3" value="_A" data-
ignore="empty">
    <input type="hidden" name="FeldA4" value="_A/$Counter.1" class="no-restore"
data-ignore="always">
    …Antworttext zur ersten Antwortalternative…
  $/Random.1

  …

  $Random.5
    <label for="mcA3E">$Counter.1</label>
    <input  id="mcA3E" type="checkbox" name="FeldA3" value="_E">
    <input type="hidden" name="FeldA4" value="_E/$Counter.1" class="no-
restore">
    …Antworttext zur fünften Antwortalternative…
  $/Random.5
$/Fixed2
```

Falls die beiden Fragen auch randomisiert werden sollen, wären einfach die Variablen `$Fixed*` durch `$Random*` zu ersetzen und ggf. noch eine `$Randomize…`-Variable der `$Random1` voranzustellen.

Die verschiedenen verwendeten Variablen wie `$Fixed1`, `$Random.1`, `$Counter.1`, `$SetCounter.1(A)` etc. wurden bereits oben in der Variablenreferenz eingeführt, werden hier aber erstmals im Rahmen eines Beispiels vorgeführt.

Das Beispiel demonstriert insbesondere folgende Punkte:

1. **Syntax für Fixed-Blöcke**: Analog zu Random-Blöcken, nur eben mit dem Schlüsselwort `Fixed` statt `Random` gebildet, also geöffnet durch `$Fixed`, ergänzt durch eine eindeutige *Blocknummer* und geschlossen durch `$/Fixed`, ergänzt um dieselbe Blocknummer.
2. **Syntax für Unterblöcke / blocklokale Blöcke**: Random-Blöcke innerhalb eines „Top-Level-" Fixed- oder Random-Blocks werden analog notiert, allerdings *mit einem Punkt vor der Blocknummer*, z.B. `$Random.1 … $/Random.1`.
   - Auch auf dieser Hierarchietiefe sind analog Fixed-Blöcke möglich, die dann entsprechend z.B. mit `$Fixed.1 … $/Fixed.1` eingefasst werden.
   - Wie bereits angekündigt, dürfen auch solche Unterblöcke wiederum „Unter-Unterblöcke" enthalten, dann angezeigt durch entsprechend *zwei* Punkte zwischen Schlüsselword (`Random` oder `Fixed`) und der Blocknummer. Die Anzahl der Punkte gibt also die Schachtelungstiefe an. (Diese Syntax orientiert sich an typischer hierarchischer Kapitelnummerierung, nur dass hier die lokale Blocknummer allein von Interesse ist, und es weder nötig noch erlaubt ist, die Blocknummern der Vorfahrblöcke vor / zwischen den jeweiligen Punkten mit anzugeben.)
3. **Blocklokales Randomize**: Auch für blocklokale Randomisierung gelten dieselben Regeln zur Initialisierung wie bei der globalen (Fragen-)Randomisierung: Im Standardfall (ohne `$Randomize`-Variable) wird pro Teilnehmer/-in genau einer der (Sub-)Blöcke ausgewählt und angezeigt. Für die Auswahl einer größeren Teilmenge und/oder eine Zufallsreihenfolge (Shuffle), ist eine `$Randomize`-Variable zu verwenden. Diese muss *innerhalb* des jeweiligen Blocks stehen, dessen blocklokale Randomisierung sie konfigurieren soll, dort jedoch *vor* der ersten Random-Variablen (hier also jeweils zwischen `$Fixed1` bzw. `$Fixed2` und der darauf folgenden `$Random.1`-Variablen).
   Beachten Sie, dass – anders als bei Fixed-, Random- oder Countervariablen – in der Randomize-Variablen *kein Punkt* für die Schachtelungstiefe angegeben wird. Das ist in diesem Fall auch nicht nötig, da der Block, auf den sich eine Randomize-Variable bezieht, immer eindeutig ist.
4. **Zählervariablen (Counter)**:
   - Multiple-Choice-Fragen im Online-Übungssystem verwenden in der Regel Kennbuchstaben (A, B, C, …) für die einzelnen Antwortoptionen. Das ist zwar keine technische Notwendigkeit, hat aber insbesondere den folgenden Vorteil: Die eigentlichen, nur im Aufgabenformular angezeigten Beschreibungen dürfen beliebig langer und komplexer HTML-Code sein, auch aus Bildern bestehen oder Bilder neben Text umfassen etc., und die gespeicherte Einsendung besteht dennoch jeweils nur aus einem Buchstaben. Die Beschreibungstexte können dann insbesondere auch noch während der Bearbeitungsphase nachbearbeitet (z.B. Tippfehler korrigiert) werden, was nicht möglich wäre, wenn die Beschreibungstexte selbst direkt eingesendet würden und eine Tippfehlerkorrektur gleich die Menge an Antwortoptionen verändern würde und bereits eingesendete Antworten damit ungültig würden.
   - Wenn diese Buchstaben aber nun den Optionen fest zugeordnet wären, würde eine Randomisierung (sowohl Mischen als auch Auswahl) nur bedingten Nutzen erfüllen,

denn nach wie vor könnten sich die Studierenden ja untereinander austauschen, ob sie „die Antwort A" für richtig halten oder nicht: Die Reihenfolge, in der sie die Antworten sehen, spielt dafür ja keine Rolle, wenn die Zuordnung der Kennbuchstaben für alle Teilnehmer konstant bleibt.

- Daher wurde im Beispiel (als Inhalt des der jeweiligen Checkbox vorangestellten Labels) jeweils eine spezielle `$Counter`-Variable verwendet. Counter-Variablen werden durch konkrete Werte ersetzt, *nachdem* die Randomisierung angewandt wurde, also in diesem Beispielfall (Shuffle-Only ohne Teilmengenauswahl) nachdem die Optionen in eine teilnehmerindividuelle Reihenfolge gebracht wurden. Damit werden die Antwortoptionen für jeden Teilnehmer immer (in *seiner* Reihenfolge) von oben nach unten mit A, B, C, D und E bezeichnet, d.h für verschiedene Studierende ist die ihnen jeweils *angezeigte* Kennung meist eine andere, die „Antwort A von Teilnehmer 1" ist nicht zwangsläufig identisch mit der „Antwort A von Teilnehmerin 2".
- Beachten Sie, dass auch hier ein *Punkt in der Countervariablen* steht. Eine Variable wie `$Counter1` wäre ein seitenglobaler Zähler, der in jedem Haupt-/Fragenblock um 1 inkrementiert würde. Eine Variable mit *einem* Punkt wie hier (`$Counter.1`) definiert einen blocklokalen Counter (also lokal in `$Fixed1 … $/Fixed1` bzw. `$Fixed2 … $/Fixed2`), der in jedem Unterblock (wie `$Random.1 … $/Random.1`) jeweils um 1 inkrementiert wird, hier also für jede Antwortoption einen anderen Wert annimmt, innerhalb einer solchen Option aber mehrfach genannt werden kann: Im Beispiel kommt die Countervariable in jedem Random-Subblock genau zweimal vor und wird beide Male durch denselben Wert, die Kennung für diese Antwortoption, ersetzt.
- Es gibt verschiedene *Zählerformate*, die `$SetCounter`-Variable im Vorfeld legt das Format fest. In diesem Beispiel wird `$SetCounter.1(A)` verwendet, was jeweils eine blocklokale Zählervariable `$Counter.1` definiert und dazu festlegt, dass dieser Zähler mit dem Buchstaben A beginnen soll, im nächsten Unterblock dann den Wert B annehmen soll, im dritten C u.s.w.
  Die `$SetCounter`-Variable ist optional: Entfällt sie, wird ein numerischer Zähler erzeugt, der mit 1 beginnt. Für Multiple-Choice-Fragen, zumindest bei Verwendung eines „3A-Bewertungsmodus" (3 Antwortalternativen: Student/-in hält die Antwortoption für wahr, falsch oder trifft gar keine Aussage) müssen allerdings Buchstaben als Kennungen verwendet werden, weil dort (in Quittung und Autokorrektur) Großbuchstaben für „Antwort als richtig eingestuft" und Kleinbuchstaben für „Antwort als falsch eingestuft" verwendet werden, und das wäre mit Ziffern als Antwortkennung nicht möglich.

5. **Interne und angezeigte MC-Antwortkennungen**:
   - Wie oben beschrieben, soll den Teilnehmern/-innen jeweils eine individuelle Kennung (A, B, C…) vor jeder Antwortoption angezeigt werden, eben in der (individuellen) Reihenfolge der Nennung der Optionen.
   - Die in der Datenbank gespeicherten Kennungen für die gegebenen Antworten sollen dagegen für alle Teilnehmer identisch sein. Zum Ersten ist das sicherer, weil die gespeicherten Einsendungen so unabhängig von der Randomisierung bleiben. Zum Zweiten ist das auch z.B. eine Voraussetzung für die Antworthäufigkeitsstatistiken: Wenn die Statistik z.B. aussagt, dass 95% aller Teilnehmer/-innen „die Antwort A" gegeben haben, dann wäre das ohne Aussagekraft, wenn dieses „A" für jede/-n Teilnehmer/-in effektiv eine andere Antwort bezeichnete.
     Daher wird also eine für alle Teilnehmer identische, zentrale Kennung für die Antwortoptionen verwendet, die den Teilnehmern jedoch nicht angezeigt wird. Hierfür könnten wieder einfach Buchstaben (hier A bis E) verwendet werden. Um aber Missverständnisse durch Verwechslungen der teilnehmerindividuellen Kennungen und

der zentralen/internen Kennungen zu vermeiden, verwendet das Beispiel (genau wie der Aufgabenerstellungsassistent) jeweils einen Buchstaben *mit vorangestelltem Unterstrich* als interne Kennung (vgl. jeweils das `value`-Attribut der Checkboxen).

- Wenn ein Student nun z.B. die erste und die letzte ihm angezeigte Antwortalternative ankreuzt, für ihn (per Counter) mit A und E bezeichnet, hinter A aber (nach Randomisierung) die Checkbox aus Block `$Random.3` mit der internen Kennung `_C` und hinter E diejenige aus `$Random.1` mit der internen Kennung `_A` stünde, so würde also die Einsendung lauten: `_C,_A`. Genau so würde sie auch in der Datenbank gespeichert. Dieser „interne" Wert der Einsendung soll aber natürlich dem Teilnehmer nicht angezeigt werden:
  - Für den Fall einer Einsendearbeit ohne Sofortkorrektur, in dem nur eine Quittung über die eingegangene Einsendung ausgegeben wird, soll statt `_C, _A` wirklich `A, E` in der Quittung stehen.
  - Und auch in der Autokorrektur soll unter „Ihre Einsendung" entsprechend `A, E`, also eine Wiedergabe seiner *individuellen* Antwortkennungen stehen.
  - Weiterhin soll auch die Anzeige der korrekten Lösung entsprechend angepasst werden. Angenommen, richtig seien die Antwortoptionen aus dem ersten, dritten und vorletzten Random-Block, dann sollte den Teilnehmern nicht `_A, _C, _D` als korrekte Lösung angezeigt werden, sondern diese zentral hinterlegte Lösung vielmehr in die jeweiligen teilnehmerindividuellen Kennungen „übersetzt" / abgebildet werden. Angenommen, die Antwort `_D` wurde dem betroffenen Teilnehmer als zweite Option „B" präsentiert, soll ihm also als korrekte Lösung also `A, B, E` angezeigt werden.
- Damit das alles möglich wird, ist es nötig, neben der „Roh-Einsendung" (Aufzählung der internen Kennungen der angekreuzten Antwortoptionen), wie `_C, _A` in obigem Beispiel) auch das Ergebnis der individuellen Randomisierung zu speichern. Daher sieht obiges Beispiel pro Frage nicht, wie normalerweise, nur *ein* Eingabefeld vor, sondern *zwei*: In der ersten Frage die Felder `FeldA1` und `FeldA2`, in der zweiten Frage entsprechend `FeldA3` und `FeldA4`.
  - Das erste der beiden Felder speichert jeweils die Einsendung mit Hilfe der internen Kennungen (wird also als `name`-Attribut in den Checkbox-Inputelementen eingetragen).
  - Das zweite der beiden Felder wird hier durch Hidden-Inputs, die die Teilnehmer also nicht zu sehen bekommen, gefüllt, und zwar so, dass darin nach Einsendung jeweils eine Abbildung von internen auf individuelle Kennungen steht. Das wird im Beispielcode oben erreicht, indem der jeweilige Counter-Wert (individuelle Kennung) mit dem festen Wert, der auch im Checkbox-Value steht (interne Kennung) kombiniert wird. Der Feldinhalt ist nach Einsendung eine Aufzählung der Values aller Hidden-Inputs mit demselben `name`, könnte für obigen Beispielfall also z.B. lauten: `_C/A,_D/B,_B/C,_E/D,_A/E`, was bedeutet: Die im HTML erste Antwort (aus `$Random.1` mit Kennung `_A`) wurde dem/-r Teilnehmer/-in als (fünfte) Antwortoption „E" präsentiert, die zweite (`_B`) als „C" u.s.w. Anhand dieser Daten kann z.B. das Korrekturmodul später die „Rückübersetzung" sowohl der im ersten Feld eingegangenen „Roheinsendung" als auch der hinterlegten Musterlösung auf die individuellen Kennungen vornehmen. Natürlich muss das Korrekturmodul entsprechend eingerichtet werden (über die Property `AnzeigeAbbFolgeFeld:`), so dass es tatsächlich zwei Eingabefelder statt nur eines auswertet und das zweite als Abbildung (Mapping) interpretiert. Analog kann auch in einer Quittung eine solche Übersetzung der Einsendung zur Anzeige mit individuellen Kennungen erfolgen,

indem dort für solche Einsendungen eine spezielle Variable (`$Feld1AAFF`) zum Einsatz kommt, die aussagt, dass eine „Anzeige-Abbildung im Folge-Feld" zu erwarten und anzuwenden ist.

- **Wichtig** ist bei diesen Hidden-Fields die Ausstattung mit dem Attribut `class="no-restore"`, welche bewirkt, dass auch bei wiederholten Seitenaufrufen dieses Hidden-Field wirklich immer konstant dasselbe Mapping enthält wie im HTML angegeben. *Ohne* dieses Attribut würde das Online-Übungssystem das Hidden-Field für ein „normales" Eingabefeld halten, in das bei wiederholtem Aufruf der Aufgabe der Wert der letzten Einsendung „vorgefüllt" werden sollte. D.h. jedes dieser hier fünf Hidden-Inputs (eines pro Alternative) hätte dann plötzlich beim wiederholten Aufgabenaufruf das gesamte Mapping aus der vorherigen Einsendung als Wert, und eine Neueinsendung würde wieder die Werte aller fünf gleichnamigen Inputs hintereinander hängen, enthielte das Mapping also dann fünffach, bei nochmaliger Einsendung schon 25-fach etc. Die `no_restore`-Klasse verhindert das Ausfüllen mit der bereits vorliegenden Einsendung und sorgt so dafür, dass auch bei jeder Folgeeinsendung wieder das korrekte Mapping in diesem Feld gespeichert wird.
- Weiterhin wurde das erste der Hidden-Fields jeweils mit `data-ignore="always"` markiert, was besagt, dass eine vorliegende Einsendung im FeldA2 kein Indiz dafür sein soll, dass diese Aufgabe bearbeitet wurde. In Kombination mit `data-ignore="empty"` an den beiden Checkbox-Feldern heißt das für diese Beispielaufgabe, dass sie als bearbeitet gilt, wenn mindestens eine der 10 Checkboxen (zu den zwei Fragen) bei der letzten Einsendung angekreuzt wurde, andernfalls als unbearbeitet. (Die `data-ignore`-Eigenschaft gilt nicht für das jeweilige Input-Element, sondern allgemein für das Eingabefeld (`FeldA1`, `FeldA2` etc.), deshalb genügt es, diese Angabe an nur einem der jeweils fünf Inputs zum Feld anzubringen.)

In obigem Beispiel wurde lediglich die Reihenfolge randomisiert, also jedem/-r Teilnehmer/-in immer auch jede Antwortoption präsentiert. Sie können durch Reduzierung der Selektionsgrößen-Angabe in der `$Randomize`-Variable natürlich auch einstellen, dass jeweils nur eine *zufällig ausgewählte Teilmenge* der Antwortoptionen pro Teilnehmer/-in angeboten werden soll. Falls Sie das jedoch mit einem `1-XausN`-Bewertermodus kombinieren möchten, also eine Multiple-Choice-Frage stellen möchten, bei der Sie garantieren, dass immer mindestens eine der Antwortmöglichkeiten auch korrekt ist und jemand, der/die gar keine Antwort markiert, gar keine Punkte bekommt, weil die Frage dann als unbeantwortet gelten soll, dann müssen Sie sicherstellen, dass *selbst bei zufälliger Auswahl immer mindestens eine richtige Alternative* zur Auswahl steht. Dazu muss die Anzahl der falschen Antworten im HTML-Formular echt kleiner sein als die Selektionsgröße, so dass es eben unmöglich wird, einem Teilnehmer ausschließlich falsche Antworten zur Auswahl anzubieten.

Beispiel 2: Single-Choice/Einfachauswahlfragen randomisieren

Einfachauswahlfragen lassen sich nahezu analog zu den Mehrfachauswahlfragen, also zu Beispiel 1 randomisieren. In erster Linie sind dazu in obigem Beispielcode die Checkboxen durch Radiobuttons zu ersetzen. Falls Sie die Antwortoptionen lediglich mischen wollen (Zufallsreihenfolge, Shuffle), aber immer jedem/-r Teilnehmer/-in *sämtliche* Antwortalternativen zur Auswahl stellen wollen, ist ansonsten wirklich alles analog zu oben.

Eine *Besonderheit* liegt dagegen vor, falls Sie den Teilnehmern/-innen nicht alle, sondern *eine zufällige Auswahl* von Antwortoptionen anbieten wollen. Bei einer Einfachauswahlfrage ist ja in der Regel genau eine der Antworten die richtige, alle anderen möglichen Antworten falsch. Bei zufälliger Auswahl, also effektiv dem zufälligen Ausblenden eines Teils der Antwortmöglichkeiten aus dem

Formular, dürfen dann *ausschließlich falsche* Antwortmöglichkeiten ausgeblendet werden. *Die richtige Antwort muss dagegen in jedem Fall zur Auswahl stehen.* Oder anders gesagt: Es soll eben *keine* zufällige Auswahl aus *allen* im HTML stehenden Antwortalternativen / Radiobuttons erfolgen, sondern *nur aus den falschen* Antworten soll zufällig gewählt werden.

Das erreichen Sie einfach, indem Sie die richtige Antwort in einen Fixed-Block und nur die falschen Antworten in einen Random-Block einfassen. Die in der Randomize-Variable angegebene Selektionsgröße darf dann natürlich nur angeben, wie viele der falschen Antworten (Random-Blöcke) zusätzlich zur korrekten (Fixed-Block) zufällig gewählt werden sollen, ist also in diesem Fall um 1 kleiner als die den Studierenden insgesamt anzuzeigenden Alternativen.

Eine Einfachauswahlfrage mit insgesamt 10 Antwortalternativen, von denen die zweite (`_B`) die korrekte ist und aus den restlichen neunen pro TN zufällig vier ausgewählt werden sollen, so dass also Jede/-r insgesamt eine 1-aus-5-Auswahl präsentiert bekommt, könnte z.B. wie folgt aussehen:

```
$Fixed1
  <h2>Einfachauswahl-Frage</h2>
  $Randomize(4, ShuffleFixed)
  $SetCounter.1(A)
  $Random.1
    <label for="mcA1A">$Counter.1</label>
    <input  id="mcA1A" type="radio" name="FeldA1" value="_A" data-
ignore="empty">
    <input type="hidden" name="FeldA2" value="_A/$Counter.1" class="no-restore"
data-ignore="always">
    …Antworttext zur ersten Antwortalternative…
  $/Random.1

  $Fixed.2
    <label for="mcA1B">$Counter.1</label>
    <input  id="mcA1B" type="radio" name="FeldA1" value="_B">
    <input type="hidden" name="FeldA2" value="_B/$Counter.1" class="no-
restore">
    …Antworttext zur zweiten Antwortalternative…
  $/Fixed.2

  $Random.3
    <label for="mcA1C">$Counter.1</label>
    <input  id="mcA1C" type="radio" name="FeldA1" value="_C">
    <input type="hidden" name="FeldA2" value="_C/$Counter.1" class="no-
restore">
    …Antworttext zur dritten Antwortalternative…
  $/Random.3

  …

  $Random.10
    <label for="mcA1J">$Counter.1</label>
    <input  id="mcA1J" type="radio" name="FeldA1" value="_J">
    <input type="hidden" name="FeldA2" value="_J/$Counter.1" class="no-
restore">
    …Antworttext zur zehnten Antwortalternative…
  $/Random.10
$/Fixed1
```

In diesem Beispiel ist nicht nur die zufällige Auswahl (falscher) Antwortalternativen eingestellt, sondern auch eine *Zufallsreihenfolge*, d.h. die 5 Antwortoptionen jedes/-r Teilnehmers/-in sollen auch noch gemischt werden. Dabei ist folgendes zu beachten:

Da hier nun neun Random-Blöcke und ein Fixed-Block stehen, würde ein „normales Shuffle" (bei Verwendung von `$Randomize(4, Shuffle)`) auch *nur die falschen Antworten mischen*, während der Fixed-Block (`$Fixed.2`) mit der korrekten Antwort an Ort und Stelle fixiert bliebe. Das wiederum hätte zur Folge, dass bei jedem/-r Teilnehmer/-in die richtige Antwort entweder an erster oder an zweiter Stelle stünde – je nachdem, ob der `$Random.1`-Block bei der Selektion ganz ausgeblendet wird (wobei der `$Fixed2`-Block an Position 1 nach oben rutscht) oder ob er sichtbar bleibt (und dann ggf. noch mit einem anderen Block vertauscht wird).

Das Schlüsselwort `ShuffleFixed` in der `$Randomize(...)`-Deklaration bewirkt dagegen, dass auch der `$Fixed.2`-Block mit den anderen neun Random-Blöcken vermischt wird. So bezieht sich die zwar Zufallsauswahl nur auf die neun Random-Blöcke, das Shuffle aber auf alle zehn Sub-Blöcke innerhalb des `$Fixed1`-Blocks.

Beispiel 3: Zuordnungsfragen randomisieren

Neben einfachen Multiple-Choice-Fragen können z.B. mit dem Aufgabenerstellungsassistenten (aber natürlich auch manuell in der fortgeschrittenen Aufgabenserstellung) auch Zuordnungsfragen erstellt werden, die wie folgt aufgebaut sind: Es wird eine Tabelle angeboten, in deren Zeile je eine Einfachauswahlfrage steht (die Frage in der ersten Spalte, die Radiobuttons zur Auswahl einer Antwort in den folgenden Spalten). Die Spalten dieser Tabelle stellen also die Antwortalternativen dar, wobei die Menge der Antwortalternativen eben für alle Fragen identisch ist.

Für dieses Beispiel nehmen wir die folgende Zuordnungsmatrix an:

	A: Anton	B: Bertha	C: Caesar
Pünktchen und …	⊙	⊖	⊖
Die dicke …	⊖	⊙	⊖
Gaius Julius …	⊖	⊖	⊙

Die nun zu erstellende Aufgabenseite soll (in fester Reihenfolge) zwei „Kopien" der obigen Zuordnungsfrage enthalten, allerdings mit unterschiedlicher lokaler Randomisierung: In der ersten Kopie sollen die Zeilen, und in der zweiten statt dessen die Spalten in eine zufällige Reihenfolge gebracht werden.

Aus Gründen der Übersichtlichkeit splitten wir den Beispielcode in zwei Abschnitte (ein Listing pro Variante / Fixed-Block), d.h. wir betrachten zunächst nur das erste Teilbeispiel, die Randomisierung der Tabellenzeilen der Zuordnungsfrage:

```
$Fixed1
  $Randomize(3, Shuffle)
  <h2>Shuffle Rows</h2>
  <table>
    <thead>
      <tr>
        <th></th>
        <th>A: Anton </th>
        <th>B: Bertha </th>
        <th>C: Caesar</th>
      </tr>
    </thead>
    <tbody>
      $Random.1
      <tr>
        <th>Pünktchen und …<input type="hidden" name="FeldA1" value="1"
class="no-restore" data-ignore="always"></th>
        <td><input type="radio" name="FeldA2" value="A" data-
ignore="empty"></td>
        <td><input type="radio" name="FeldA2" value="B"></td>
        <td><input type="radio" name="FeldA2" value="C"></td>
      </tr>
      $/Random.1
      $Random.2
      <tr>
        <th>Die dicke …<input type="hidden" name="FeldA1" value="2" class="no-
restore"></th>
        <td><input type="radio" name="FeldA3" value="A" data-
ignore="empty"></td>
        <td><input type="radio" name="FeldA3" value="B"></td>
        <td><input type="radio" name="FeldA3" value="C"></td>
      </tr>
      $/Random.2
      $Random.3
      <tr>
        <th>Gaius Julius …<input type="hidden" name="FeldA1" value="3"
class="no-restore"></th>
        <td><input type="radio" name="FeldA4" value="A" data-
ignore="empty"></td>
        <td><input type="radio" name="FeldA4" value="B"></td>
        <td><input type="radio" name="FeldA4" value="C"></td>
      </tr>
      $/Random.3
    </tbody>
  </table>
$/Fixed1
```

Die eigentliche Blockstruktur ähnelt den Beispielen zur lokalen Multiple-Choice-Randomisierung, d.h. innerhalb des `$Fixed1`-Blocks gibt es lokale Unterblöcke `$Random.1` bis `$Random.3`.

Da jeder dieser lokalen Random-Blöcke hier nicht eine Antwortalternative zu einer MC-Frage ist, sondern selbst eine *vollständige* Einfachauswahl-Frage darstellt, deren Antworten (Spalten der Matrix) konstant / nicht randomisiert sind, besteht hier kein Bedarf für eine „Anzeige-Abbildung im Folgefeld“ wie in den obigen Beispielen.

Statt dessen kommt hier aber eine andere Besonderheit zum Einsatz, die bei reinem Shuffle ohne

Selektion zwar nicht unbedingt nötig, aber sinnvoll, bei Zeilenauswahl sogar dringend empfehlenswert ist:
Es werden wieder Hidden-Inputs verwendet, um Randomisierungsinformationen zu sammeln und mit ans Korrekturmodul zu übermitteln: In diesem Fall wird dazu gleich das *erste* Eingabefeld der Frage (`FeldA1`) verwendet, und dieses Feld wird wieder über mehrere einzelne Hidden-Inputs gefüllt, die im Wirkungsbereich der Randomisierung stehen, in diesem Fall aber einfach eine absolute Zeilennummer (1 bis 3) der jeweiligen Tabellenzeile (als `value`) enthalten. Wenn also beim Mischen der Zeilen einem Teilnehmer z.B. zuerst Frage 2, gefolgt von Frage 3 und dann Frage 1 angezeigt wird, dann wird im `FeldA1` der Wert `2,3,1` eingesendet.

Das Korrekturmodul muss dazu so eingestellt werden (über die Property `RandomFeld1:`), dass es eben im ersten Feld die Aufzählung der Zeilen-/Fragennummern in der Anzeigereihenfolge erwartet und die Eingaben erst in den nachfolgenden Eingabefeldern (`FeldA2` bis `FeldA4`). Es nutzt bei Zufallsreihenfolge diese Information dazu, die Autokorrekturen dem/-r jeweiligen Teilnehmer/-in dann in derselben, individuellen Reihenfolge anzuzeigen, wie die Fragen im randomisierten Aufgabenformular standen.
Bei aktivierter Zufallsauswahl (wenn also die in `$Randomize` eingestellte Selektionsgröße kleiner als die Anzahl der Random-Blöcke dazu ist) enthält dieses Feld dann nicht nur die Information über die Reihenfolge, sondern auch über die getroffene Auswahl: Zu jeder Fragennummer, die im Feld nicht vorkommt, geht das Korrekturmodul dann davon aus, dass diese Frage auch im Aufgabenformular nicht vorkam, und bewertet daher die Inhalte der Eingabefelder zu diesen Fragen (typischerweise leer, weil ja nichts dazu eingesendet werden konnte) gar nicht.

Auch hier ist wieder zu beachten, dass diese Hidden Inputs mit dem Attribut `class="no-restore"` ausgestattet sein müssen, aus den selben Gründen wie bei den vorhergehenden Beispielen: Die Werte der Hidden-Fields sollen immer konstant aus ihren `value`-Attributen im HTML-Code bestehen und nicht durch eine ggf. schon vorliegende Einsendung überschrieben werden.

Ansonsten sollte alles zum Verständnis des obigen Beispielcodes Nötige aus den Ausführungen zu Beispiel 1 hervorgehen.

Kommen wir nun daher zum zweiten Teil dieses Beispiels, einer weiteren Zuordnungsfrage, dieses Mal jedoch nicht mit Randomisierung der Zeilen, sondern der Spalten der Matrix (also der Antwortmöglichkeiten):

```
$Fixed2
  <h2>Shuffle Cols</h2>
  <table>
    $Fixed.1
      $Randomize(3, Shuffle, 2)
      $SetCounter..1(A)
      <thead>
        <tr>
          <th></th>
          $Random..1 <th>$Counter..1: Anton</th> $/Random..1
          $Random..2 <th>$Counter..1: Bertha</th> $/Random..2
          $Random..3 <th>$Counter..1: Caesar</th> $/Random..3
        </tr>
      </thead>
    $/Fixed.1
    <tbody>
      $Fixed.2
        $Randomize(3, Shuffle, 2)
        $SetCounter..1(A)
```

```
        $SetCounter..1(A)
        <tr>
          <th>Pünktchen und …</th>
          $Random..1
            <td><input type="radio" name="FeldA5" value="_A" data-
ignore="empty">
              <input type="hidden" name="FeldA6" value="_A/$Counter..1"
class="no-restore" data-ignore="always">
            </td>
          $/Random..1
          $Random..2
            <td><input type="radio" name="FeldA5" value="_B">
              <input type="hidden" name="FeldA6" value="_B/$Counter..1"
class="no-restore" >
            </td>
          $/Random..2
          $Random..3
            <td><input type="radio" name="FeldA5" value="_C">
              <input type="hidden" name="FeldA6" value="_C/$Counter..1"
class="no-restore" >
            </td>
          $/Random..3
        </tr>
      $/Fixed.2
      $Fixed.3
        $Randomize(3, Shuffle, 2)
        $SetCounter..1(A)
        <tr>
          <th>Die dicke …</th>
          $Random..1
            <td><input type="radio" name="FeldA7" value="_A" data-
ignore="empty">
              <input type="hidden" name="FeldA8" value="_A/$Counter..1"
class="no-restore" data-ignore="always">
            </td>
          $/Random..1
          … (weitere Spalten analog zu oben)
        </tr>
      $/Fixed.3
      $Fixed.4
        $Randomize(3, Shuffle, 2)
        $SetCounter..1(A)
        <tr>
          <th>Gaius Julius …</th>
          $Random..1
            <td><input type="radio" name="FeldA9" value="_A" data-
ignore="empty">
              <input type="hidden" name="FeldA10" value="_A/$Counter..1"
class="no-restore" data-ignore="always">
            </td>
          $/Random..1
          … (weitere Spalten analog zu oben)
        </tr>
      $/Fixed.4
    </tbody>
  </table>
$/Fixed2
```

Die Zuordnungsfrage besteht ja praktisch aus einer Menge von einzelnen Einfachauswahl-Fragen, wobei jede Zeile eine eigene Frage darstellt und die Spalten jeweils die Antwortoptionen. Die Besonderheit gegenüber drei einzelnen Einfachauswahlfragen liegt „nur" darin, dass alle drei Fragen *dieselben* Antwortoptionen nutzen und dass sich diese Fragen daher in Matrix-/Tabellenform anzeigen lassen (hier mit den Fragen als Zeilen und den Antwortoptionen als Spalten).

So gesehen dürfte es nicht überraschen, dass die Randomisierung der Spalten (Antwortoptionen) vom Grundsatz genauso erfolgt wie bei Einfachauswahlfragen (siehe Beispiel 2 oben) oder Mehrfachauswahlfragen (siehe Beispiel 1 oben):

- Es wird also insbesondere wieder mit Antwortkennungen in Form von Buchstaben gearbeitet,
- wobei die einzusendenden / „internen" Kennungen (siehe `value`-Attribute der Radiobuttons) hier zur Abgrenzung mit einem Unterstrich eingeleitet werden und den Teilnehmern-/innen nicht angezeigt werden,
- während die anzuzeigende / individuelle Kennung per `$Counter`-Variable erzeugt wird, die wiederum per `$SetCounter` so eingestellt ist, dass sie die angezeigten Antwortoptionen – in der Reihenfolge *nach* Randomisierung, also der angezeigten Reihenfolge – mit „A", „B" und „C" bezeichnet.
- Weiterhin werden nun wieder zu jeder der Fragen *zwei* Eingabefelder vorgesehen, das *erste* für die eigentliche Einsendung (`value`-Attribut des vom Nutzer ausgewählten Radiobuttons), das *zweite* („Folgefeld"), gefüllt durch Hidden-Inputs im Einflussbereich der Randomisierung, zur Aufnahme eines Anzeige-Mappings (siehe obige Ausführungen zu „Anzeige-Abbildung im Folgefeld" unter Beispiel 1).

Diese Spaltenrandomisierung weist aber auch einige Besonderheiten auf. Grund: Es gibt im HTML kein Element für eine Tabellenspalte. Vielmehr ist eine Tabelle eine *Folge von Tabellenzeilen*, und diese Tabellenzeilen *untergliedern sich jeweils in Tabellenzellen*.

- Ein erster Schritt zur Umsetzung der Spalten-Randomisierung besteht also darin, lokal *innerhalb jeder Tabellenzeile* eine Randomisierung der *Zellen* vorzunehmen.
- Um eine solche *zeilenlokale Randomisierung* umsetzen zu können, muss also jede Zeile ein einzelner Block sein.
- Da die Zeilen in diesem Fall selbst nicht randomisiert werden, wurden dazu Fixed-Blöcke `$Fixed.1` bis `$Fixed.4` gewählt, einer für die Kopfzeile der Tabelle, deren Zellen, also Spaltenüberschriften, ja auch gemischt werden sollen, und drei für die Frage-Zeilen. (Wenn die Zeilen ebenfalls randomisiert werden sollten, wären die Frage-Zeilen wieder in `$Random.2` bis `$Random.4` einzufassen, während die Kopfzeile natürlich weiterhin fixiert, also in `$Fixed.1` eingefasst bleiben müsste.)
- Innerhalb dieser vier Fixed-Blöcke wird dann lokal randomisiert, wozu die einzelnen, zu mischenden Tabellenzellen wiederum in Random-Unterblöcke `$Random..1` bis `$Random..3` eingefasst werden.
- Damit nicht eine zufällige Zelle pro Zeile ausgewählt wird, sondern immer alle Zellen angezeigt, aber in eine Zufallsreihenfolge gebracht werden, wird eine blocklokale `$Randomize`-Variable verwendet wird, die die Selektionsgröße auf 3 einstellt und per `Shuffle` eine Zufallsreihenfolge aktiviert.
    - Wenn man es nun aber bei `$Randomize(3, Shuffle)` beließe, ergäbe sich ein großes Problem: Jede Tabellenzeile würde *unabhängig* von den anderen Zeilen randomisiert, d.h. die Tabellenstruktur würde gehörig „zerhäckselt", die *Spalten blieben nicht zusammenhängend*, die einzelnen Radiobuttons passten nicht mehr notwendig zur jeweiligen Spaltenüberschrift.
    - Zur Lösung dieses Problems bietet `$Randomize` die Möglichkeit, als dritten (bzw.

zweiten, wenn „Shuffle“ nicht mit angegeben wird) Parameter eine weitere Zahl anzugeben. Die Wirkung ist, dass in allen Blöcken mit derselben Zahl an dieser Stelle dieselbe Zufallsverteilung verwendet wird (vorausgesetzt, die Anzahl und Nummern der Random-Unterblöcke sind jeweils identisch). Hier im Beispiel wurde für alle Tabellenzeilen (die drei Fragezeilen als auch die Tabellenkopfzeilen) jeweils dieselbe Zahl `2` dort eingetragen. Der Wert ist im Prinzip beliebig, wichtig ist aber, dass, falls mehrere Tabellen mit Spaltenrandomisierung in der Aufgabenseite vorkommen, zwar *innerhalb einer Tabelle* jeweils dieselbe Zahl verwendet wird, aber nicht derselbe Wert für *mehrere* Tabellen (damit diese nicht identisch randomisiert werden). Hier bietet es sich an, einfach die Fragennummer (bzw. Nummer des umschließenden `$Fixed-Blocks` zu verwenden).

- Der Counter zur Erzeugung der anzuzeigenden Antwortkennungen nach Randomisierung soll nun natürlich auch mit jeder Antwortoption, also jeder Zelle innerhalb einer Tabellenzeile, sprich: für jeden der `$Random..*`-Subblöcke einer Zeile seinen Wert inkrementieren. Daher wird jeweils ein Counter mit ebenfalls zwei Punkten im Namen verwendet (`$Counter..1`). Dieser wiederum ist jeweils lokal pro umfassendem `$Fixed.*`-Block (Tabellenzeile) und entsprechend auch in jedem dieser Blöcke wieder neu mittels `$SetCounter..1` zu konfigurieren.
- Für die „Anzeigeabbildung im Folgefeld“ wird der bereits für die Mulitple-Choice-Fragen eingeführte Mechanismus wiederverwendet, auch wenn das hier bedeutet, dass pro Frage das Mapping jeweils redundant in einem eigenen Eingabefeld abgelegt wird, obwohl es für alle Fragen gleich ist.

Natürlich ist es auch möglich, die Zeilen- und die Spaltenrandomisierung zu einer Tabelle / Zuordnungsfrage zu kombinieren. Dann sind entsprechend die `$Fixed.*`-Blöcke pro Tabellenzeile durch `$Random.*` zu ersetzen (mit Ausnahme der Tabellenkopfzeile natürlich, die „Fixed“ bleiben muss, da ja nur die Zeilen des Tabellenkörpers randomisiert werden sollen), und es wäre wieder ein erstes Hidden Field für die Zeilenauswahl (zusätzlich zu den Hidden Fields für die Antwortkennungs-Mappings der Zeilen) einzuführen.

Beispielcode dafür geben wir hier nicht spezifisch an, aber im Zweifel können Sie ja auch eine solche Zuordnungsfrage mit Zeilen- und Spaltenrandomisierung live in einer Online-Übungssystem-Kursumgebung per Aufgabenerstellungsassistent erstellen und sich das erzeugte HTML dann ansehen.

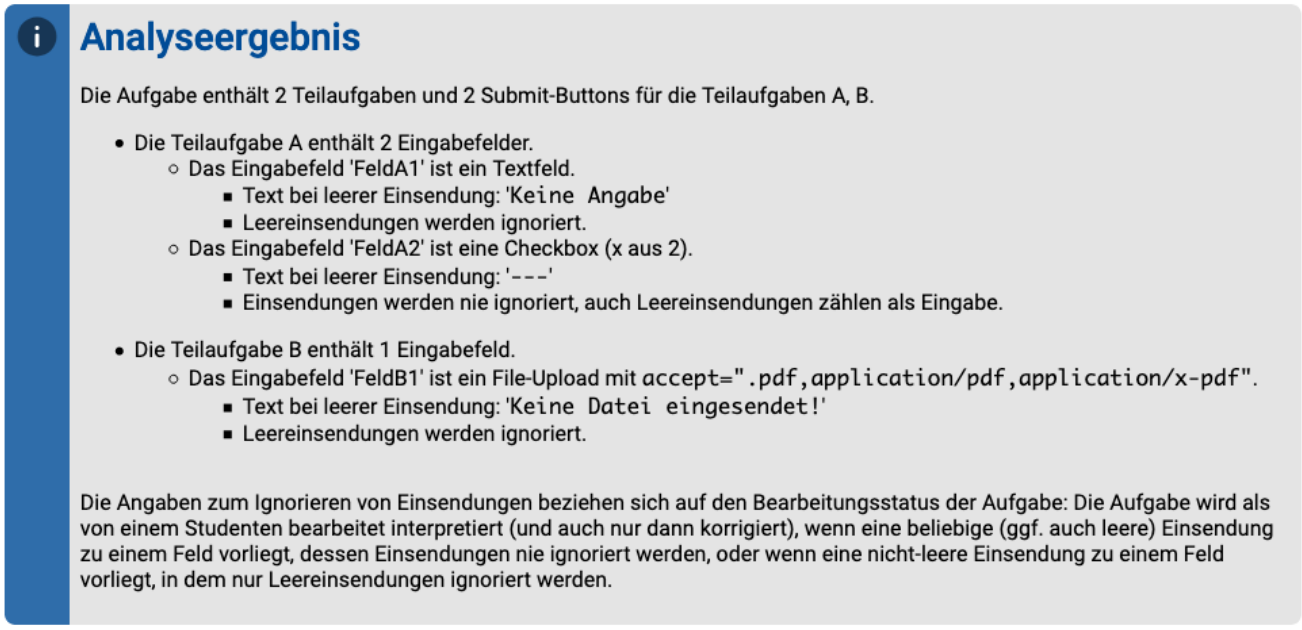
Wichtig: Siehe spätere Abschnitte

Mit der hier beschriebenen Einfügung von Random-Blöcken, ggf. Randomize-Variable ins Aufgabenformular und Unterteilung der Aufgabe in Teilaufgaben (ggf. mit Selektionssubmit) ist die Einrichtung der Fragen-Randomisierung noch nicht abgeschlossen! In den noch folgenden Arbeitsschritten wie Aufgabeneinrichtung, Musterlösungsseitenerstellung, Quittungs- und Korrekturvorlagenerstellung etc. muss die Randomisierung ebenfalls berücksichtigt werden. Die nachfolgenden Abschnitte werden daher das Thema immer wieder aufgreifen.

Teilaufgaben und Eingabefelder erzeugen

Nachdem Sie die Aufgaben-HTML-Datei (z.B. `aufgabe1.1.html`) fertiggestellt haben (online bearbeitet oder offline bearbeitet und dann als Kursressource hochgeladen), öffnen Sie wieder die Ansicht der fortgeschrittenen Aufgabenerstellung. In der Rubrik »Aufgabenseite« finden Sie dort außer den Icons zum Bearbeiten oder Ansehen der Aufgabenseite unten eine Unterrubrik »*Aufgabenseite analysieren und Datenbank anpassen*« (vgl. Abbildung zu Beginn des Abschnitts Aufgabenformular erstellen). Zur ersten Kontrolle Ihrer Datei betätigen Sie den Button »*Vorlage analysieren*« und kontrollieren Sie die anschließend eingeblendete Ausgabe. Diese informiert Sie z.B. darüber, wieviele Teilaufgaben, Submit-Buttons und Eingabefelder in der HTML-Datei erkannt wurden, auch Details wie der Inhalt des `data-if-empty`-, `data-ignore`- oder `accept`-Attributs müssen dort aufgelistet werden – fehlen diese, wurden die Attribute nicht erkannt, z.B. weil sie falsch geschrieben wurden.

Für das *begleitende Fallbeispiel* aus dem vorhergehenden Abschnitt „Eingabefelder“ sieht das Analyseergebnis dann z.B. wie folgt aus:



Analyseergebnis

Die Aufgabe enthält 2 Teilaufgaben und 2 Submit-Buttons für die Teilaufgaben A, B.

- Die Teilaufgabe A enthält 2 Eingabefelder.
  - Das Eingabefeld 'FeldA1' ist ein Textfeld.
    - Text bei leerer Einsendung: 'Keine Angabe'
    - Leereinsendungen werden ignoriert.
  - Das Eingabefeld 'FeldA2' ist eine Checkbox (x aus 2).
    - Text bei leerer Einsendung: '---'
    - Einsendungen werden nie ignoriert, auch Leereinsendungen zählen als Eingabe.
- Die Teilaufgabe B enthält 1 Eingabefeld.
  - Das Eingabefeld 'FeldB1' ist ein File-Upload mit accept=".pdf,application/pdf,application/x-pdf".
    - Text bei leerer Einsendung: 'Keine Datei eingesendet!'
    - Leereinsendungen werden ignoriert.

Die Angaben zum Ignorieren von Einsendungen beziehen sich auf den Bearbeitungsstatus der Aufgabe: Die Aufgabe wird als von einem Studenten bearbeitet interpretiert (und auch nur dann korrigiert), wenn eine beliebige (ggf. auch leere) Einsendung zu einem Feld vorliegt, dessen Einsendungen nie ignoriert werden, oder wenn eine nicht-leere Einsendung zu einem Feld vorliegt, in dem nur Leereinsendungen ignoriert werden.

Analyseergebnis für Aufgabenseite mit obigem Beispiel-Formular

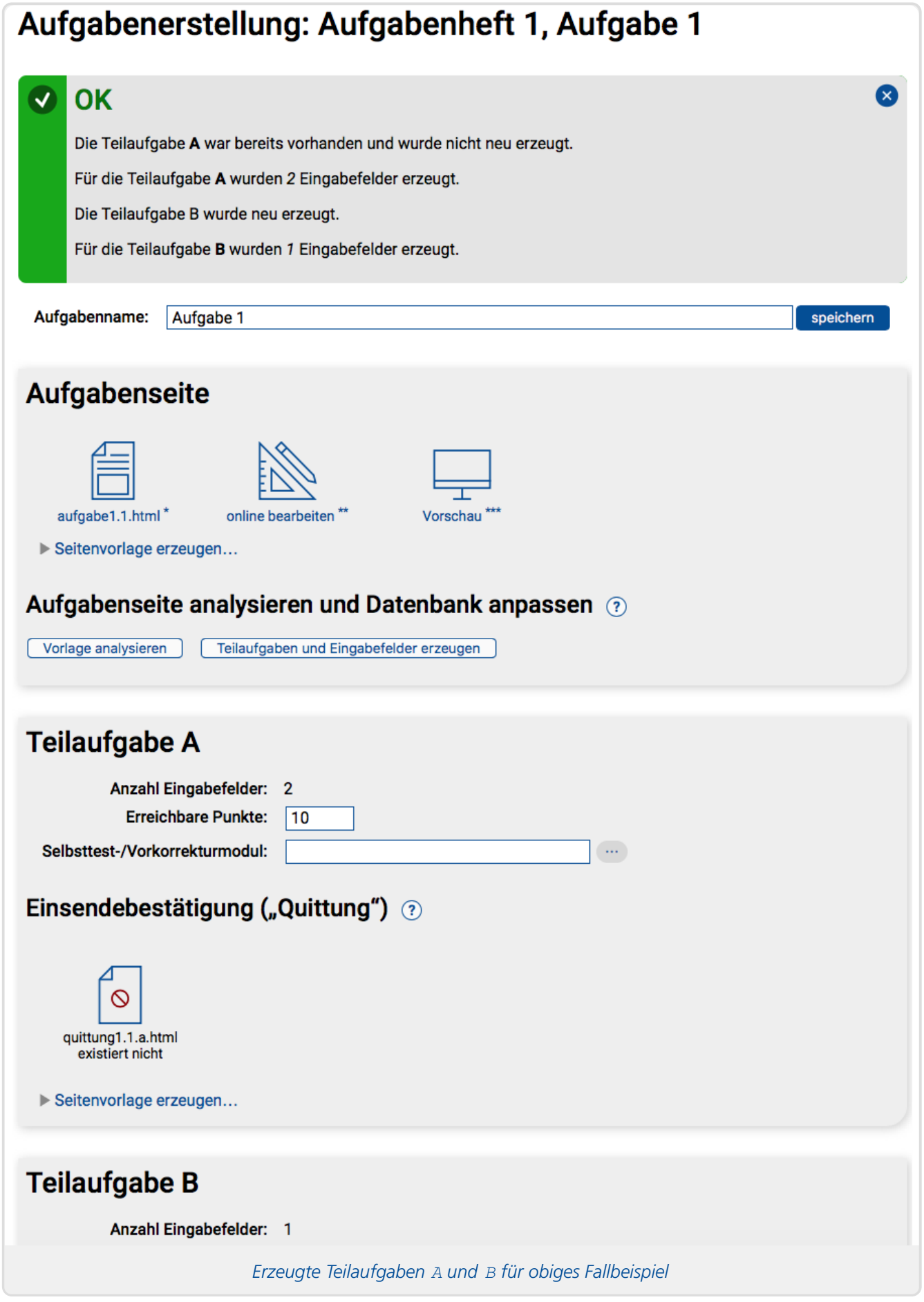
Stimmen die Analyseergebnisse mit der intendierten Aufgabenformular-Struktur überein, betätigen Sie den zweiten Button: »*Teilaufgaben und Eingabefelder erzeugen*«.

Dieser Schritt ist *sehr wichtig*, denn er überträgt die in der HTML-Datei gefundenen Strukturinformationen in die Datenbank. Damit wird insbesondere dafür gesorgt, dass die Datenbank für die Speicherung von Einsendungen in allen vorkommenden Eingabefeldern vorbereitet wird. (Eine typische Fehlersituation: Einem Aufgabenformular wird nachträglich ein Eingabefeld hinzugefügt, jedoch vergessen, diese Funktion erneut aufzurufen. Resultat: Eingaben von Studenten in dem neuen Eingabefeld werden nicht gespeichert.)

Auch Informationen aus den Attributen `data-if-empty`, `data-ignore` und `accept` der `input`-Elemente werden in diesem Schritt erkannt und in die Datenbank übertragen. Nach Änderungen an diesen Attributen in der HTML-Datei muss also ebenfalls diese Funktion erneut aufgerufen werden, damit die Änderungen tatsächlich auch wirksam werden! (Änderungen am `accept`-Attribut sind zwar für die clientseitige Prüfung im Browser sofort nach Speicherung wirksam, aber damit auch die serverseitige Filterung des Übungssystems sich auf ein anderes Accept-

Muster umstellt, ist ebenfalls diese Funktion nach dem Speichern der Anderungen auszuführen!)

Eine dritte Auswirkung dieser Funktion ist das Anlegen der Teilaufgaben (eine Teilaufgabe pro `einsendenX`-Button), was sich auch direkt im User-Interface der fortgeschrittenen Aufgabenerstellung niederschlagen kann: Dort findet sich nun eine Rubrik pro Teilaufgabe:



Erzeugte Teilaufgaben A und B für obiges Fallbeispiel

Nun können für jede Teilaufgabe die Einstellungen vorgenommen werden, hier die erreichbaren Punkte. Die in einer Aufgabe insgesamt erreichbare Punktzahl errechnet sich als Summe der Punktzahlen aller Teilaufgaben. (Das gilt auch, falls Sie die Fragen-Randomisierung <https://online-

uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#fragenrandom> verwenden: Sollten Sie, wie oben im Abschnitt Formular mit Fragen-Randomisierung empfohlen, für jede mögliche Frage eine eigene Teilaufgabe erstellt haben, so kann ein Teilnehmer aufgrund der Randomisierung nur genau eine der Teilaufgaben bearbeiten. Das „weiß" das Übungssystem im Allgemeinen aber nicht, weshalb es dennoch die Punktzahlen aller Teilaufgaben aufaddiert. Daher wird für diesen Fall empfohlen, die erreichbare Punktzahl – die für alle Fragen gleich sein sollte – nur bei der ersten Teilaufgabe einzutragen und die Punktzahl für alle weiteren Teilaufgaben offen zu lassen.)

Soll ein Vorkorrekturmodul <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#moduleoverview> angebunden werden, das bereits programmiert und in Betrieb ist, so dass es sich beim Übungssystem für diesen Kurs als zur Verfügung stehendes Modul angemeldet hat, dann kann es unter »Selbsttest-/Vorkorrekturmodul« über den »…«-Button ausgewählt werden. Andernfalls ist es möglich, den Modulnamen bereits einzutragen, unter dem sich zukünftig einmal ein Modul anmelden soll. Sobald ein Modulname vergeben ist, erscheint auch ein zusätzlicher Button zum Festlegen von Properties fürs Vorkorrekturmodul. Ob und welche Properties es erwartet, hängt ganz vom Modul ab.

Im Fallbeispiel gehen wir davon aus, dass keine Vorkorrekturmodule eingesetzt werden und lassen die Felder daher leer.

Quittungsvorlage(n) erstellen

Nach dem Erstellen des Aufgabenformulars würde man es sicher gerne testen, aber noch ist ein Test relativ witzlos, da man mit dem fertigen Formular zwar schon Daten einsenden könnte, jedoch noch keine Anzeige der eingesendeten Daten unterstützt wird, anhand derer man sehen könnte, was eingesendet wurde. Als nächster Schritt empfiehlt sich daher das Erstellen der Quittungsseiten, die nach einer Einsendung ein erstes Feedback geben, insbesondere anzeigen sollen, was wirklich eingegangen ist.

Da Einsendungen separat pro Teilaufgabe <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#ta> erfolgen, muss für jede Teilaufgabe eine separate Quittungsseite erstellt werden, welche die Einsendungen zur jeweiligen Teilaufgabe bestätigt. (Wenn Ihre Aufgabe nicht in Teilaufgaben zerlegt ist, also alle Eingaben zur gesamten Aufgabe zusammen eingesendet werden, erstellen Sie natürlich auch nur eine Quittungsseite.)

Die zu erstellenden Quittungs-Seitenvorlagen werden Ihnen in der fortgeschrittenen Aufgabenerstellung einzeln angezeigt (vgl. obige Screenshots). Betätigen Sie jeweils den »Neue Einsendungsbestätigungsseite generieren!«-Button, um eine erste Version der jeweiligen Quittungsseite erstellen zu lassen. Anschließend erhalten Sie wieder dieselben Bearbeitungsmöglichkeiten wie schon bei der Aufgabenseite: erstens die Anzeige der Rohfassung im Browser samt Download-Möglichkeit oder Editiermöglichkeit mit Browser-Editor-Suite, zweitens die Online-Bearbeitung in einem Web-basierten Editor.

Bei einer solchen Quittungsseite handelt es sich wieder um eine normale HTML-Seite, die spezielle Variablen als Platzhalter für diverse vom Übungssystem einzufügende Informationen enthält.

Für die Teilaufgabe `A` des obigen Fallbeispiels wird in etwa folgende Quittungsvorlage generiert:

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Best&auml;tigung Ihrer Einsendung</title>
</head>
<body>
  <div style="background-color:#ccc; padding: 0.2em">
    Kurs $KursNr, &bdquo;$Kursname&ldquo;, $Versionsname<br>
    Aufgabenheft $AufgabenheftNr, Aufgabe $AufgabenNr
  </div>
  <h1>Best&auml;tigung Ihrer Einsendung</h1>
  $EMBED
  <p>von $Vorname $Nachname, Matrikelnr. $MatrikelNr<br>
  f&uuml;r &bdquo;$Kursname&ldquo;, $KursNr<br>
  Aufgabenheft $AufgabenheftNr, Aufgabe $AufgabenNr, Teilaufgabe a</p>
  <hr><br>
  <p>In Feld 1 haben Sie eingegeben: </p>
  <pre>$Feld1P</pre>
  <p>In Feld 2 haben Sie eingegeben: </p>
  <pre>$Feld2P</pre>
  $/EMBED
  <hr>
  <a
href="$WebAssignServer/$Veranstaltername/StudentenStartSeite/$KursNr/$VersionsN
r/">Zur&uuml;ck zur Aufgaben&uuml;bersicht</a>
</body>
</html>
```

Die Seite könnte unverändert so eingesetzt werden, sie enthält z.B. bereits Embedding-Marken und Variablen zur Einbindung der Feldinhalte. Die eher generischen Texte/Beschriftungen wie »In Feld 1 haben Sie eingegeben:« möchte man jedoch z.B. oft gerne gegen speziellere Texte mit Bezug zur konkreten Aufgabenstellung austauschen. Eventuell möchte man auch gewisse Formatierungen z.B. per CSS ändern oder die Feldeingaben z.B. in einer Tabelle darstellen, falls auch die Eingabefelder im Aufgabenformular tabellarisch angeordnet waren.

Auch die Art der Einbindung der Einsendungen lässt sich noch beeinflussen, wie im Folgenden genauer beschrieben wird.

Variablen für Quittungsseiten

Sie können alle oben im Abschnitt Variablen des Online-Übungssystems vorgestellten Standard-Variablen auch hier wieder verwenden. Insbesondere wird der Einsatz von `$EMBED` / `$/EMBED` empfohlen, um den damit markierten Inhalt der Quittungsvorlage in eine zentrale Seitenlayout-Vorlage einzubetten, die ein globales Webdesign und Menüs zur Navigation z.B. zurück zur Aufgabenstellung oder zu den anderen Aufgaben desselben Hefts bietet.

Neben diesen Standard-Variablen spielen in Quittungsseiten vor allem diejenigen Variablen eine zentrale Rolle, über welche die Einsendungen und ggf. Vorkorrekturen in der Quittung präsentiert werden. Darüber hinaus gibt es noch spezielle Variablen mit Studentendaten, die in Quittungen und Korrekturen sinnvoll eingesetzt werden können (vgl. z.B. die obige generierte Quittungsseite).

Feldvariablen

Zur Anzeige der studentischen Eingaben in der Quittung spielen die Feldnamen eine zentrale Rolle, mit denen die Eingabefelder im Formular der Aufgabenseite benannt wurden. Die entsprechenden Variablen zum Einfügen der Feldwerte in die Aufgabenseite haben praktisch denselben Namen wie die Felder auch, nur entfällt bei Quittungsseiten die Angabe des Buchstabens für die Teilaufgabe, da jede Quittungsseite ohnehin teilaufgabenspezifisch ist[15].

Eine Variable für Feldinhalte (in Quittungen!) hat folgenden Grundaufbau:

```
$Feld<FeldNr><Suffix>
```

Die Feldnummer entspricht der auch im Feldnamen angegebenen, und das Suffix ist optional und kann die Darstellung des Feldwerts beeinflussen.

Im begleitenden Fallbeispiel mit zwei Teilaufgaben gibt es drei Eingabefelder mit den Namen `FeldA1`, `FeldA2` und `FeldB1`. Dazu wären zwei Quittungsseiten zu erzeugen, wobei in `quittung1.1.a.html` die Variablen `$Feld1` und `$Feld2`, sowie in `quittung1.1.b.html` die Variable `$Feld1` einzubinden sind.

Die folgende Tabelle soll die möglichen Suffixe erläutern; die Feldnummer `1` in den Variablenbezeichnern ist exemplarisch zu verstehen und durch die jeweils konkrete Feldnummer zu ersetzen:

Variable	Bedeutung
`$Feld1`	*Automatik-Modus*: Texte werden unverändert ins HTML übernommen, Dateiuploads normalerweise als Link dargestellt, Bilddateien in geeigneten Formaten als Inline-Bilder statt Links eingebunden, und der Inhalt hochgeladener Plaintext-Dateien wird direkt eingebettet (nicht bei XML- oder HTML-Uploads). *Im Fall von Texteingaben gilt:* Die Eingabe des Studenten wird unverändert (unescaped) eingebettet! Falls der Student HTML-Markup eingegeben hat (bzw. das Eingabefeld eine mit WYSIWYG-Editor versehene Textarea ist, in die der Student formatierten Text eingab, welcher wiederum als HTML-Markup eingesendet wird), wird dieses auch 1:1 ins Quittungs-HTML eingefügt und entsprechend vom Browser als HTML-Markup geparst. Dieser Modus eignet sich im Allgemeinen *nicht* gut für Plaintext-Einsendungen: *Insbesondere darf eine solche Eingabe normalerweise keine Kleiner-Zeichen `<` oder Ampersands `&` als Payload enthalten, da diese vom Browser als Tagklammern oder Entitäten interpretiert werden könnten!*
`$Feld1P`	*Automatik-Modus in Programm-/Plaintext-Variante*: Erkennt ebenfalls automatisch Dateiuploads, aber *im Fall von Texteingaben gilt:* Eingabe des Studenten wird als Plaintext interpretiert und daher vor der Einbettung in die Quittungs-HTML-Seite geeignet „escaped", d.h. alle Vorkommen von `<`, `>` und `&` werden ersetzt durch `<`, `>` bzw. `&`. Sorgt außerdem (durch Einfügen von `$NOEXPAND` und `$/NOEXPAND`, s.o.) dafür, dass in der Einsendung enthaltene mit `$` beginnende Strings wie `$KursNr` bei der Auslieferung der fertigen Korrektur an den Studenten nicht vom Übungssystem als Variablen fehlinterpretiert werden können. *Im Fall von Dateiuploads gilt zusätzlich:* Auch hochgeladene XML- oder HTML-Dateien werden nicht verlinkt, sondern deren Inhalt in die Quittung eingebunden, wobei auch hier Tags und Entities escaped werden, also deren Quelltexte angezeigt werden.
`$Feld1HREF`	Nur für Dateiupload-Felder: Erzwingt die Ausgabe eines Links zur hochgeladenen Datei, verhindert also die Einbettung von Inline-Bildern oder Textinhalten eingesendeter Dateien. Als Linkbeschriftung wird (wie im Automatikmodus) der Dateiname verwendet.
`$Feld1URL`	Nur für Dateiupload-Felder: Gibt nur den URL zur Dateieinsendung aus, sollte also selbst innerhalb des `href`-Attributs eines (selbst beschrifteten) Links stehen, z.B. `<a href="$Feld1URL">Ihre hochgeladene Datei</a>`.
Nur für Dateiupload-Felder: Erzwingt die Ausgabe als `img`-Tag mit dem URL im `src`-	

Variable	Bedeutung
$Feld1IMG	Nur für Dateiupload-Felder: Erzwingt die Ausgabe als `img`-Tag mit dem URL im `src`-Attribut, selbst wenn die Automatik die Einbettung nicht als so darstellbare Bilddatei erkannt hat (nur in Ausnahmefällen zu verwenden).
$Feld1PLAIN	Erzwingt die Einbettung des eingesendeten Inhalts als String in die Quittung, selbst wenn es sich um einen Dateiupload handelt, der nicht anhand seines Content-Types als Text erkannt wurde[16] und im Automatik-Modus daher als Download-Link dargestellt würde. Texte werden *nicht escaped*, siehe `$Feld1`!
$Feld1PPLAIN	Kombination aus `$Feld1P` und `$Feld1PLAIN`: Erzwingt Dateiinhalts-Einbettung mit Escaping.
$Feld1TEX	Erzwingt die Interpretation des eingesandten Textes als TeX-Formel und deren Rendering (vgl. TeX-Formelsatz).
$Feld1JN	Bewirkt die Ausgabe von »Nein«, falls in dem Feld nichts eingesendet wurde, ansonsten von »Ja«. Eine `data-if-empty`-Einstellung für dieses Feld wird ignoriert. Vorgesehen für einzelne Checkboxen (statt Checkbox-Gruppen) in Eingabeformularen, die als Ja-/Nein-Antworten interpretiert und in der Quittung entsprechend ausgegeben werden sollen.
$Feld1NUM	Bewirkt eine Interpretation der Texteingabe als Zahl (ganze Zahl oder Fließkommazahl). Ausgegeben wird neben der unveränderten Eingabe des Studenten auch eine Darstellung der in der Eingabe erkannten Zahl (sofern diese von der Eingabe abweicht). Vor allem vorgesehen für Quittungsseiten zu Zahleneingaben für erst nach Einsendeschluss automatisch bewertete Aufgaben. Details dazu finden Sie im folgenden Abschnitt Quittungen zu Zahleneingaben. Im Gegensatz zur folgenden Variable explizit vorgesehen für freie Texteingaben.
$Feld1DECIMAL	Vorgesehen für Eingaben aus Eingabefeldern der Art `<input type="number" inputmode="decimal" step="any" name="FeldA2"…>`. In Number-Feldern kann kein beliebiger Text mehr eingegeben werden, sondern nur Zahlen. Sofern diese Felder per `inputmode="decimal"` und geeignetem `step`-Attribut auch die Eingabe von Fließkommazahlen erlauben, gilt, dass Browser diese Eingaben immer in einheitlicher Form einsenden, insbesondere mit Punkt als Dezimaltrenner, auch wenn sie in „nationaler Schreibweise“ – in Deutschland mit Komma als Dezimaltrenner – eingegeben wurden. Diese Variable bewirkt, dass für solche Einsendungen wie z.B. „-12.3“der Punkt wieder durch ein Komma ersetzt wird. Und in Einsendungen wie „.5“ wird auch wieder eine Null vorangestellt: „0,5“. Im Gegensatz zu obiger `NUM`-Variablen ist diese also nicht darauf ausgelegt, in einer beliebigen Texteingabe eine Zahl zu suchen, sondern wirklich explizit dazu vorgesehen, Einsendungen aus Number-Inputs wieder in nationaler Schreibweise auszugeben.
$Feld1SPLIT	Optimierte Darstellung von Teileinsendungen aus mehreren gleichnamigen Eingabefeldern. Normalerweise werden Einsendungen aus mehreren gleichnamigen Eingabefeldern so quittiert, wie sie gespeichert werden, also indem die jeweiligen Eingaben durch den im `data-separator`-Attribut des Eingabefelds festgelegten Separator getrennt hintereinander aufgezählt werden, ggf. mit Escaping von Separator-Vorkommen in den Eingaben durch Voranstellen eines Escape-Zeichens, typischerweise Backslash. Wenn Sie solche Einsendungen in einer „hübscheren“, menschenlesbareren Form quittieren möchten, verwenden Sie das `"SPLIT"`-Suffix am Variablennamen. Damit werden solche Einsendungen wieder in ihre Teile zerlegt und entweder durch », « (Komma und Leerzeichen) oder » / « (Querstrich eingerahmt durch Leerzeichen) erneut aufgezählt. Das Übungssystem wählt die „seiner Meinung nach“ passendere Darstellung abhängig davon aus, ob in den Einsendungen selbst Kommas oder Querstriche vorkommen.
$Feld1AAFF	Bewirkt ein Auslesen *zweier* Eingabefelder, der „eigentlichen“ Einsendung aus Feld1 und einer „Anzeige-Abbildung im Folge-Feld“ in Feld2. Der Wert von Feld2 muss dabei eine kommaseparierte Aufzählung von Querstrich-separierten Wertepaaren sein, z.B. `_A/C,_B/E,_C/D,_D/A,_E/B`. Das bewirkt dann, dass in diesem Beispiel alle Vorkommen von `_A` im Wert von Feld1 in der Quittungsanzeige auf `C` abgebildet werden sollen, alle Vorkommen von `_B` auf `E`, allgemein jedes vor einem der Querstriche stehende Vorkommen durch den Wert hinter dem Querstrich. Lautet der Wert von Feld1 z.B. `_E,_A` und der von Feld2 wie oben, dann würde an Stelle dieser Variablen der Eingang von `B,C` quittiert. Diese Variable ist vorgesehen für die Randomisierung z.B. der

Variable	Bedeutung
	Antwortoptionen von Multiple-Choice-Fragen und

Das bedeutendste Suffix ist also das `P`: Variablen mit `P`-Suffix oder ganz ohne Suffix beschreiben beide eine Inhalts-Anzeige im Automatik-Modus. Der Unterschied liegt darin, ob einzufügender Textinhalt escaped wird oder nicht.

> *Faustregel:*
> *Reine Texteingaben sollten mit `P`-Suffix eingebunden werden, damit darin enthaltene `<`-, `>`- und `&`-Zeichen zuverlässig angezeigt und nicht als HTML-Markup fehlinterpretiert werden. Bei Plaintext-Textareas gilt zusätzlich: Damit auch die Zeilenumbrüche oder Einrückungen durch Leerzeichen der Eingabebox mit in die HTML-Seite übernommen werden (nicht vom Browser „reflowed" werden), sollte die Variable in einem `pre`-Element stehen! Einsendungen aus mit WYSIWYG-HTML-Editor versehenen Textareas dagegen dürfen nicht mit `P`-Suffix eingebunden werden (und nicht in einem `pre`-Element stehen), denn diese sind durch HTML-Tags formatiert, welche daher auch vom Browser auch als solche interpretiert werden sollen. Eine `$FeldiP`-Variable würde in diesem Fall den HTML-Quelltext der studentischen HTML-Einsendung in der Quittung zeigen!*

Betrachten Sie noch einmal oben stehenden Quelltext einer generierten Quittungsseite: Das Übungssystem hat automatisch die in den meisten Fällen sinnvollste Kombination `<pre>$Feld1P</pre>` eingefügt, geht also davon aus, dass in Feldern im Normalfall (wenn es sich nicht um Datei-Uploads handelt) reiner Text eingesendet wird, der aus einer Textarea stammen könnte, also Zeilenumbrüche und Whitespace-Formatierungen enthalten könnte.
Speziell in diesem Beispiel könnten die `pre`-Tags auch entfernt werden, da die Eingaben aus einem einzeiligen Text-Input und einer Checkbox-Gruppe stammen. Das `pre` sorgt in diesem Fall standardmäßig auch noch für eine Formatierung in nicht-proportionaler Schrift. Aber eine solche Formatierung (sofern überhaupt gewünscht) könnte man auch per CSS erreichen.

Vorkorrektur

Falls Sie ein Selbsttest-/Vorkorrekturmodul <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#moduleoverview> (im Allgemeinen nur als „Vorkorrekturmodul" bezeichnet, auch wenn es der Realisierung von Selbstkontrollaufgaben <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#ska> dient) verwenden, sollten Sie im Normalfall neben den Eingaben auch die Rückmeldung des Vorkorrekturmoduls mit in die Quittung einbeziehen, um den Studenten ein Sofort-Feedback[17] zu liefern.

Im Normalfall wird dazu einfach die Variable `$Vorkorrektur` eingebunden.

Ein (SOAP-)Vorkorrekturmodul kann einen Text (wahlweise Plaintext oder HTML-formatierten Text) ausgeben sowie optional eine Menge von Dateien (z.B. Anlagen zur Korrektur, modifizierte/korrigierte Dateieinsendungen von Studenten etc.).

Falls Sie ein Modul mit solchen Dateianlagen einsetzen, stehen Ihnen auch hier Varianten dieser Variable durch verschiedene Suffixe zur Auswahl, mit denen Sie entweder alle Vorkorrekturen ausgeben können oder eine ganz bestimmte über ihren *Referenz-Namen* (i.F. `xyz`) ansprechen können. (Das Vorkorrekturmodul muss jeder Datei-Vorkorrektur einen solchen Namen zuordnen, über den man diese gezielt referenzieren kann.)

Variable	Bedeutung
`$Vorkorrektur`	Gibt den vom Vorkorrekturmodul erzeugten Text aus (sofern vorhanden) – automatisch escaped, falls es sich um Plaintext handelt bzw. unverändert, falls er HTML-formatiert ist[18]. Ggf. vom Vorkorrekturmodul erzeugte Dateien werden durch diese Variable nicht ausgegeben.
`$Vorkorrekturen`	Alle Vorkorrekturen werden ausgegeben, d.h. der Text wie im Falle von `$Vorkorrektur` sowie zusätzlich ggf. eine Aufzählung (Bulleted List) von Downloadlinks zu allen vom Vorkorrekturmodul erzeugten Dateien.
`$VorkorrekturHREF(xyz)`	Erzeugt einen Downloadlink zur Dateivorkorrektur mit Referenz-Namen `xyz` (s.o.)
`$VorkorrekturIMG(xyz)`	Versucht, die Dateivorkorrektur namens `xyz` als Bild (`img`) einzubinden. Sollte nur verwendet werden, wenn das Vorkorrekturmodul sicherstellt, dass die erzeugten Dateien dieses Namens immer Bilder in einem von Browsern unterstützten Format sind.
`$VorkorrekturURL(xyz)`	Erzeugt lediglich den URL zur Vorkorrektur namens `xyz`, der manuell in der Quittungsseite in ein Attribut wie das `href`-Attribut eines Links oder das `src`-Attribut eines `img`s eingesetzt werden kann.

Falls Sie eine Dateivorkorrektur mit einer der drei letztgenannten Variablen einbinden wollen und Ihr Vorkorrekturmodul ohnehin *nur eine einzige Dateivorkorrektur* erzeugt, kann die Namensreferenz `(xyz)` auch entfallen, d.h. es kann dann auch einfach eine Variable der Art `$VorkorrekturIMG` angegeben werden.

Student

Es ist sinnvoll, in Quittungen (falls ein Student sich diese z.B. ausdruckt und später jemandem vorlegen will) Matrikelnummer und Namen des Studenten mit auszugeben. Das können Sie mit folgenden Variablen erreichen:

Variable	Bedeutung
`$MatrikelNr`	Matrikelnummer des einendenden Studenten
`$Vorname`	Vorname des einsendenden Studenten
`$Nachname`	Nachname des einsendenden Studenten
`$EMailAdresse`	E-Mail-Adresse des einsendenden Studenten

Automatisch generierte Quittungsseiten enthalten bereits einen Header mit Matrikelnummer und Namen des Studenten (siehe Listing oben).

Teilaufgabe

Variable	Bedeutung
`$TeilaufgabenNr`	Kleinbuchstabe (`a` – `z`), der die Teilaufgabe bezeichnet, zu der eingesendet wurde.

Spezial

Zunächst stehen Ihnen auch in Quittungsseiten dieselben Kontrollvariablen wie im Aufgabenformular zur Verfügung. Genau wie bei Feldvariablen auch, können Sie dabei die Teilaufgabenkennung weglassen, da jede Quittungsseite ohnehin teilaufgabenspezifisch ist (die Einsendung zu genau einer Teilaufgabe quittiert). So können Sie z.B. per `$IfNotEmpty1(...)` Text ausgeben, der nur sichtbar wird, wenn im Feld 1 der quittierten Teilaufgabe eine nicht-leere Einsendung vorliegt.

Speziell für Quittungen zu Dateiuploads gibt es noch folgende Spezialvariable, deren Anwendung im anschließenden Abschnitt noch demonstriert wird:

Variable	Bedeutung
`$HinweisLeereDateieinsendung`	Fügt einen zentral vom Übungssystem vorgegebenen Text in die Quittungsseite ein, der auf eine leere Einsendung hinweisen soll. (Der Wortlaut der Meldung kann sich jederzeit ändern, da der Text zentral vom ZMI gewartet wird.) Diese Variable sollte unbedingt *innerhalb* eines `$IfEmpty`...-Blocks stehen, also wirklich nur dann eingebunden werden, wenn eine Leereinsendung festgestellt wurde. (Zwischen diesen Variablen `$IfEmpty`..., `$HinweisLeereDateieinsendung` und `$/IfEmpty`... können Sie dann noch die Darstellung der Meldung gestalten, z.B. per `$WarnungsBox`..., s.u.)

Eigentlich eher für den Übungssystem-internen Gebrauch vorgesehen sind spezielle Variablen zur Einbindung farblich und mit Icons hervorgehobener Boxen für Erfolgs-, Fehler- oder Warnmeldungen. Für den Einsatz in Aufgaben sind diese normalerweise nicht zu verwenden, und wenn überhaupt (da es sich um Feedback handelt), dann vor allem in Quittungsseiten. (Die korrekte Anzeige solcher Boxen in Ihren Quittungs- oder Aufgabenressourcen setzt voraus, dass Ihr Kurs das jeweils aktuelle Webdesign für die Aufgabenseiten verwendet!)

Insbesondere die Warnmeldung kann im Kontext leerer Dateieinsendungen sinnvoll sein (siehe folgenden Abschnitt):

Variable	Bedeutung
`$WarnungsBox(Überschrift)` ...`$/WarnungsBox`	Blendet eine Warnmeldung ein. Der Text in Klammern hinter dem öffnenden `$WarnungsBox` bildet die Überschrift, der Inhalt ... zwischen der schließenden Klammer und `$/WarnungsBox` den eigentlichen Inhalt der Box (HTML-Code).
`$InfoBox(...)`...`$/InfoBox`	Analog für einen Informationstext
`$ErfolgBox(...)`...`$/ErfolgBox`	Analog für eine Erfolgsmeldung
`$FehlerBox(...)`...`$/FehlerBox`	Analog für eine Fehlermeldung
`$ConfirmationBox(...)` ... `$/ConfirmationBox`	Analog für eine Bestätigungsbox. (Gibt nur Sinn, wenn der Inhalt wiederum ein HTML-Formular mit Interaktionsmöglichkeit enthält.)

Quittungen zu (einfachen) Dateiuploads

Normalerweise wird eine Quittungsseite für jedes Eingabefeld i.W. eine Feld-Variable wie `$FeldB1` enthalten, der ein geeigneter einleitender Text vorangestellt wird.

Speziell für Dateiuploads gibt es einige Suffixe wie `HREF`, die Sie an die Variable anhängen können,

um je nach Dateityp die Darstellung (z.B. Darstellung eines hochgeladenen Bildes direkt in der Seite oder Darstellung immer Link) zu steuern, siehe obigen Abschnitt zu den Variablen.

Der vorangestellte Text könnte bei Dateiuploads z.B. lauten: „Die folgende Datei ist eingegangen:"

Somit könnte der Quittungsteil zu einem Dateiupload (wie in obigem begleitenden Fallbeispiel) wie folgt aussehen:

```
<p><strong>Die folgende Datei ist eingegangen:</strong>
$Feld1HREF</p>
```

Falls, wie im Abschnitt zu einfachen Dateiuploads im Aufgabenformular empfohlen, per `data-if-empty`-Attribut ein Ersatztext (z.B. „Es wurde keine Datei hochgeladen") für leere Einsendungen festgelegt wurde, wird im Falle einer leeren Einsendung die Feld-Variable durch diesen Ersatztext ersetzt.

Nimmt nun aber ein Student eine *Leereinsendung* vor, vielleicht aus Versehen, indem er keine Datei auswählt, bevor er den Upload-Button drückt, wird er „nur" eine Quittung der folgenden Art erhalten:

> ***Bestätigung Ihrer Einsendung***
> *Die folgende Datei ist eingegangen: Es wurde keine Datei hochgeladen.*

Möchten Sie auf diesen Umstand prominenter hinweisen, können Sie in der Quittung gezielt eine Fallunterscheidung vornehmen und für den Fall einer Leereinsendung eine prominente Fehlerbox einblenden lassen:

```
$IfNotEmpty1
<p><strong>Die folgende Datei ist eingegangen:</strong>
$Feld1HREF</p>
<p>Bitte überprüfen Sie, ob die Datei fehlerfrei übertragen wurde. Im Zweifel
können Sie den Upload wiederholen oder eine korrigierte Fassung der Datei
einsenden.</p>
$/IfNotEmpty1
$IfEmpty1
$WarnungsBox($Feld1)
$HinweisLeereDateieinsendung
$/WarnungsBox
$/IfEmpty1
```

Hier wird zunächst zwischen `$IfNotEmpty1` und `$/IfNotEmpty1` ein Block eingeleitet, der nur angezeigt wird, falls im Feld 1 (der Teilaufgabe des Dateiuploads) eine nicht-leere Einsendung vorliegt, also wirklich eine Datei hochgeladen wurde. Darauf folgt ein mit `$IfEmpty1` und `$/IfEmpty1` geklammerter Block, der nur bei Leereinsendungen angezeigt wird. Somit wird *immer genau einer* der beiden Blöcke in der Quittung angezeigt.

Der erste Block für nicht-leere Dateieinsendungen enthält die bereits oben entworfene Standard-Quittung für eingegangene Dateien.

Den zweiten Block für leere Dateieinsendungen könnten Sie prinzipiell beliebig gestalten; in obigem Beispiel wird er aus „Fertigbausteinen" des Online-Übungssystems zusammengefügt. So wird eine `$WarnungsBox`-Variable (s.o.) verwendet, um eine Standard-Warnungsbox des Übungssystems generieren zu lassen. Als Überschrift für die Warnungsbox (Text in Klammern hinter der öffnenden

Variablen) wurde hier einfach `$Feld1` eingesetzt, was wiederum, wie bereits bekannt, bei leeren Einsendungen durch den im Aufgabenformular per `data-if-empty`-Attribut festgelegten Ersatztext ersetzt wird. Natürlich können Sie hier auch direkt eine Überschrift wie „Keine Datei hochgeladen“ eintragen. Als Inhalt der Warnungsbox wurde in obigem Beispiel die Variable `$HinweisLeereDateieinsendung` gewählt, die durch einen Standardtext des Online-Übungssystems ersetzt wird.

Insgesamt sieht die Quittung bei leerer Dateieinsendung dann wie in folgendem Screenshot aus:

Warnungsbox bei leerer Einsendung

Quittungen zu Texteingaben

Das Wesentliche wurde oben im Abschnitt Feldvariablen bereits grundlegend erklärt, insbesondere die Verwendung des Suffixes `P` an der Feldvariablen für alle Texteingaben, die als Plaintext und nicht als HTML-Text zu interpretieren sind.

Dieser Abschnitt soll der Übersichtlichkeit halber noch einmal die empfohlene Einbindung der Feldvariablen zu verschiedenen Texteingabe-Szenarien (vgl. auch Abschnitt Eingabefelder/Textareas) zusammenstellen:

Die Feld-Variable für Text- oder HTML-Einsendungen

Plaintext-Eingaben werden mit einer Feldvariablen mit Suffix P (z.B. `$Feld1P`) eingebunden. Dieses bewirkt insb., dass in der Eingabe vorkommende `<`- und `>`-Zeichen nicht unverändert in das Quittungs-HTML übernommen und somit vom Browser als Tagklammern interpretiert werden, sondern vom Übungssystem vor der Einbindung in die Quittungsseite durch `<` bzw. `>` ersetzt und damit den Studierenden wieder als Textzeichen angezeigt werden. Dagegen werden HTML-Eingaben (typischerweise aus WYSIYG-Texteditoren) ohne dieses Suffix ins Quittungs-HTML eingefügt (also z.B. `$Feld1`), damit eben die eingesendete HTML-Tags auch wieder unverändert zum Browser gesendet und von diesem interpretiert werden, während bei Verwendung einer `P`-Variablen die Quittung den HTML-Quelltext der Einsendung anzeigen würde.

Pre-Elemente für Plaintext-Eingaben aus Textareas

Haben Sie in Ihrer Aufgabenseite einfache Texteingaben über `<textarea>`-Elemente *ohne* WYSIWYG-Editor für formatierten (HTML-)Text entgegengenommen, so sollten Sie diesen Text in der Quittung (und Korrektur) jeweils als `<pre>`-Element einbinden. Der Grund ist, dass diese Plaintext-Einsendungen erstens manuelle Zeilenumbrüche und zweitens auch Texteinrückungen durch Leerzeichenfolgen etc. enthalten können, die vom Browser bei „normale“ Einbindung ins HTML ansonsten komplett ignoriert bzw. zu einfachen Leerzeichen zusammengefasst würden.

Im einfachsten Fall könnte die Quittung zu einer Textarea-Eingabe also wie folgt aussehen:

```
<pre>$FeldA1P</pre>
```

Wie schon bei den Textareas im Aufgabenformular (siehe Textareas) haben Sie dabei auch hier die Möglichkeit, über CSS die Darstellung zu bestimmen. Folgende CSS-Klassen sind dazu im Online-Übungssystem vordefiniert:

class	Beschreibung
`proportional`	Aktiviert eine Proportionalschrift (analog zu Absatztext) an Stelle der Standard-Festweitenschrift, die ansonsten für `pre`-Elemente verwendet wird.
`autowrap`	Aktiviert einen automatischen Zeilenumbruch für Textzeilen, die ansonsten Überbreite hätten und horizontal gescrollt werden müssten.

Beachten Sie den Unterschied zwischen Textareas und Pre-Elementen: In Textareas ist standardmäßig immer ein automatischer Zeilenumbruch aktiv, das Übungssystem bietet daher eine Klasse `hardwrap` an, um dies zu verhindert. Für Pre-Elemente ist dagegen standardmäßig nur ein harter Zeilenumbruch (also nur an Stellen, an denen ein CR bzw. LF im Code steht) aktiv, und das Übungssystem bietet die Klasse `autowrap` an, um hier einen automatischen Umbruch zu aktivieren. Die Klasse `proportional` dagegen steht für beide identisch zur Verfügung, da sowohl für Textareas als auch für Pre-Elemente eine Festweitenschrift wie Courier ansonsten der Default ist.

Nehmen wir z.B. an, Sie haben eine Textarea für Plaintext mit Soft-Wrap und Proportionalschrift wie folgt im Aufgabenformular eingebunden:

```
<textarea name="FeldA2" class="proportional fullwidth" rows="10" cols="100">
```

Dann sähe die empfohlene Einbindung in die Quittungsseite dazu wie folgt aus:

```
<pre class="autowrap proportional">$Feld2P</pre>
```

Sollten Sie gezielt nach der Quellcode in einer bestimmten Programmiersprache gefragt haben (dann auch typischerweise in einer Textarea mit `class="hardwrap"`, also ohne automatische Zeilenumbrüche und ohne Proportionalschrift), dann sollte dieser Code innerhalb des Pre-Elements auch noch per Code-Element als Quellcode gekennzeichnet werden:

```
<pre><code>$Feld5P</code></pre>
```

Dann wird automatisch Syntax-Highlighting des Quellcodes versucht. Über ein `class`-Attribut können Sie weiterhin auch die erwartete Programmiersprache fürs Syntax-Highlighting vorgeben. Details dazu finden Sie im separaten Handbuch zu WYSIWYG-Editoren, In-Browser-Korrektur und Syntax-Highlighting <https://online-uebungssystem.fernuni-hagen.de/download/WYSIWYG_IBK/WYSIWYG_IBK.html> .

Quittungen zu Zahleneingaben

Für Eingaben in Textfelder

Angenommen, Ihre Aufgabe enthält Eingabefelder, in denen die Teilnehmer nur jeweils eine Zahl eintragen sollen, aber prinzipiell beliebigen Text eingeben können (`type="text"`).

Falls diese Zahleneingabe nun nicht nur manuell durch einen Korrektor interpretiert und bewertet werden soll, sondern durch ein automatisches Korrekturmodul (z.B. den internen Bewerter <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#aufgabenbewerter> für Fragen nach Zahlen in Lösungsintervallen), so ist nicht unbedingt sichergestellt, dass der Bewerter die Zahleneingabe genauso versteht wie der Student sie gemeint hat. Insbesondere die Darstellung von Fließkommazahlen ist nicht international einheitlich: Im deutschen Sprachraum wird ein Komma als Dezimaltrenner verwendet und ein Punkt kann unter Umständen als (optionaler) Tausendertrenner verwendet werden (z.B.: 1.299.999,75). Im englischen Sprachraum dagegen werden Komma und Punkt genau umgekehrt verwendet. Und gerade im Bereich von EDV/Informatik wird auch hierzulande oft die Schreibweise mit Punkt als Dezimaltrenner verwendet.

Zumindest der interne Aufgabenbewerter arbeitet daher wie folgt:

- Da hierzulande sowohl das Komma (vor allem im Druck) als auch der Punkt (vor allem im Bereich der Computerprogrammierung) als Dezimaltrenner in Gebrauch sind, sollen beide Schreibweisen von Studenten erlaubt sein: Eine Eingabe von 0.5 oder von 0,5 soll also jeweils identisch als »Null-Komma-Fünf« gelesen und bewertet werden.
- Ein Tausender-Trenner – der ohnehin nur eine Lesehilfe vor allem im Druckbereich ist und nicht zur Bedeutung der Zahl beiträgt – wird daher in der Eingabe nicht erlaubt (sonst wäre die erste Regelung nicht möglich).

Während damit zwar nun „englische“ Eingaben wie 0.5 korrekt erkannt werden, können andere nach deutscher Schreibweise mit dem optionalen Tausendertrenner dagegen nicht korrekt interpretiert werden:

- Eine Eingabe wie 1.000,2 wäre ungültig, da sie zwei Dezimaltrenner enthält. Wenn für das Eingabefeld die Zahlen-Eingabevalidierung aktiv ist (siehe separates Handbuch <https://online-uebungssystem.fernuni-hagen.de/download/JSFormValidation/FormularValidierung.html>), ist so eine Eingabe

gar nicht möglich. Wird die Eingabe aber ermöglicht, würde der interne Aufgabenbewerter nur nach der ersten vollständigen Zahl (nach obiger Regel) in der Eingabe suchen, also nur bis zum Komma lesen und die Eingabe folglich als 1,0 (Eins) interpretieren.

- Eine Eingabe wie 1.234 würde als 1,234 interpretiert.
- Bei einer Eingabe von 1.000 dagegen nimmt das Übungssystem dagegen heuristisch an, dass die Wahrscheinlichkeit, dass jemand damit wirklich 1,000 meint und unnötig viele Nullen hinters Komma schreibt, geringer ist, also die Wahrscheinlichkeit, dass damit die Zahl 1000 gemeint ist. Also wird speziell in diesem Fall (ein Punkt gefolgt von drei Nullen) der Punkt ingoriert. Der Aufgabenbewerter zeigt in so einem Fall auch explizit einen Text an wie »Eingabe '1.000' interpretiert also 1000«.
    - Diese Heuristik greift aber explizit nur bei drei Nullen. Bei einer Eingabe wie 1.001 ist es durchaus wieder plausibel, dass es sich um eine Zahl mit drei Stellen nach dem Komma (1,001) handeln könnte.

Wenn nun die Aufgabe nicht als Soforttestaufgabe konfiguriert ist, so dass der Student sofort sieht, wie seine Eingabe vom Aufgabenbewerter interpretiert und bewertet wurde, sondern wenn vielmehr erst einmal nur seine Eingabe als eingegangen und gespeichert quittiert werden soll und erst mit der Korrektur (i.d.R. nach Einsendeschluss) die Autokorrektur folgt, dann können derartige Fehl-Erkennungen ärgerlich für die betroffenen Teilnehmer sein: Gibt ein Teilnehmer z.B. „1.001“ ein, meint damit aber nicht 1,001, sondern 1001, und bekommt als eingegangene Einsendung genau seine Eingabe angezeigt, wird er intuitiv davon ausgehen, dass er die Zahl Eintausendundeins korrekt eingeschickt hat, und erst nach Einsendeschluss erfahren, dass diese Eingabe als 1,001 statt 1001 bewertet wurde.

Daher ist es möglich und für Quittungen zur späteren automatischen Auswertung auch durchaus sinnvoll, nach der Einsendung nicht grundsätzlich nur die „nackte“, uninterpretierte Texteingabe zu quittieren, sondern auch die Zahl, so wie der interne Aufgabenbewerter sie später ggf. interpretieren wird. Genau dazu dient das das Suffix `NUM` an der Feldvariable.

Soll z.B. eine solche Zahleneingabe in einem Eingabefeld `Feld1` erfolgen, so sollten Sie in der Quittung nicht einfach nur `$Feld1` (oder `$Feld1P`) einbinden, womit die Eingabe einfach nur exakt wiederholt würde, sondern Sie sollten die Zahleneingabe per `$Feld1NUM` in die Quittungsseite einbinden.

Falls die Zahlendarstellung der *interpretierten* Zahl von der *Eingabe* abweicht, wird dann in der Quittung (an der Stelle dieser Variablen) weder nur die Originaleingabe noch nur die Interpretation angezeigt, sondern beides, und zwar in der Form »Eingabe 'x' wird als folgende Zahl interpretiert: y« (Änderungen im Wortlaut vorbehalten).

Beispiele:

Eingabe in `Feld1`	Ausgabe per `$Feld1NUM`
0,5	0,5
0.5	Eingabe '0.5' wird als folgende Zahl interpretiert: 0,5
-22	-22
+22 (*)	Eingabe '+22' wird als folgende Zahl interpretiert: 22
13,75	13,75
1.234,5 (*)	Eingabe '1.234,5' wird als folgende Zahl interpretiert: 1,234
1.2E2	Eingabe '1.2E2' wird als folgende Zahl interpretiert: 120
+32,0 (*)	Eingabe '+32,0' wird als folgende Zahl interpretiert: 32
1.0	Eingabe '1.0' wird als folgende Zahl interpretiert: 1
1.00	Eingabe '1.00' wird als folgende Zahl interpretiert: 1
1.000	Eingabe '1.000' wird als folgende Zahl interpretiert: 1000
1.000.00 (*)	Eingabe '1.000.00' wird als folgende Zahl interpretiert: 1000
1.000.000 (*)	Eingabe '1.000.000' wird als folgende Zahl interpretiert: 1000000
1.000,000 (*)	Eingabe '1.000,000' wird als folgende Zahl interpretiert: 1000
1,000.000 (*)	Eingabe '1,000.000' wird als folgende Zahl interpretiert: 1

(*): Diese Eingaben sind nicht möglich, falls für das Text-Eingabefeld die (alte/deprecated) JavaScript-Eingabevalidierung für Zahleneingaben aktiviert wurde (und JavaScript im Browser des jeweiligen Nutzers nicht deaktiviert ist).
(In echten Zahlen-Eingabefeldern (`<input type="number"...>`) sind die meisten der obigen Eingaben nicht möglich, aber für solche Felder ist auch die `NUM`-Variable nicht vorgesehen, siehe dazu den nächsten Abschnitt.)

Wenn Sie also Zahleneingaben mit dieser `NUM`-Variablen quittieren und ein Student eine Zahl so eingibt, dass der Aufgabenbewerter sie eventuell anders interpretieren würde als von ihm gemeint, so wird ihm diese aus seiner Sicht falsche Interpretation direkt in der Quittung angezeigt. Auf diese Weise hat er die Chance, seine Eingabe noch rechtzeitig vor Einsendeschluss zu korrigieren.

Hinweis: Die Darstellung der interpretierten Zahl in dieser Ausgabe ist auf maximal zehn Nachkommastellen limitiert, wird also ggf. auf die zehnte Nachkommastelle gerundet. Ein Korrekturmodul (wie insb. der interne Aufgabenbewerter) ist davon nicht betroffen und kann auch noch weitere Nachkommastellen berücksichtigen (bis zur bestmöglichen Genauigkeit des Datentyps `double`).

Für spezifische Zahleneingabefelder

Der vorhergehende Abschnitt geht noch davon aus, dass beliebige Texteingaben möglich sind. Die vom `NUM`-Variablen erzeugte Ausgabe kann daher explizit einmal die Zahl (in normierter Darstellung) ausgeben, als die die Eingabe interpretiert wurde, sowie zusätzlich (falls abweichend) die wörtliche Eingabe.

Wie in Abschnitt Zahleneingabefelder oben zur HTML-Formular-Erstellung aber schon ausgeführt wurde, können Sie auf zwei Arten Felder auf Zahleneingaben beschränken.

1. Zahlenfelder mit `type="number"`.
   Für Number-Felder im Dezimal-Modus mit Support für Nachkommastellen gilt nun (wie

ebenfalls bereits oben ausgeführt), dass Browser zwar eine Schreibweise mit Komma *anzeigen* und erwarten, dann aber bei der Einsendung *immer einen Punkt als Dezimaltrenner verwenden*.

Falls Sie also ein solches `<input type="number" inputmode="decimal" step="any" name="FeldA1"…>` in Ihr Aufgabenformular einbinden, …

- und Eingaben aber nur mit `$Feld1` quittierten, würde dort immer ein Punkt statt eines Kommas in der Quittung angezeigt, auch wenn explizit ein Komma eingegeben wurde. Das ist zumindest nicht optimal, den Studierenden, die Eingaben mit Komma vornehmen, dann ihre Einsendung in abweichender, „amerikanischer Schreibweise“ zu quittieren.
- Quittierten Sie dagegen mit `$Feld1NUM` und gäbe jemand korrekt eine Zahl mit Komma ein (z.B. „12,3“), erhielte er/sie eine Quittung wie („Eingabe '12.3' wird als folgende Zahl interpretiert: 12,3“). Das ist noch weniger sinnvoll.
- Daher ist für diese Fälle vielmehr die Variable `$Feld1DECIMAL` vorgesehen: Diese ersetzt bei der Anzeige der Einsendung den Punkt wieder durch ein Komma und versucht so, die vom Browser vor Einsendung vorgesehene „Normierung“ wieder zurückzunehmen und als dem für Datenverarbeitung optimierten Format wieder ein zur Anzeige in Deutschland optimiertes Format zu machen. Neben der Ersetzung von Punkt durch Komma wird außerdem im Zweifel eine 0 vorangestellt, wenn die Einsendung direkt mit einem Punkt beginnt: Eine Eingabe wie „0,5“ kann als „.5“ eingesendet werden und wird eben im DECIMAL-Modus wieder zur Anzeige „0,5“ zurück konvertiert.

2. JavaScript-Validierung für Textfelder per `class="digits"` oder `class="number"` (deprecated).
   Da insbesondere im zweiten Fall (Fließkommazahlen) die Einsendungen dann wahlweise ein Komma oder einen Punkt als Dezimaltrenner enthalten können (s.o.), könnten Sie auch hier eine Variable mit `NUM`-Suffix verwenden. Sollte dann jemand eine Dezimalzahl wie „12.3“ einsenden, erhält er eine Quittung der Art »Eingabe '12.3' wird als folgende Zahl interpretiert: 12,3«. Wenn aber nicht eine später automatisch zu bewertende Zahleneingabe quittiert werden soll, sondern im Anschluss z.B. manuell korrigiert werden soll, genügt eine Einbindung ganz ohne Suffix (also reine Anzeige der unveränderten Einsendung ohne jegliche Interpretation).

Quittungen bei Randomisierung / Selektionssubmit

Falls Sie Fragen-Randomisierung <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#fragenrandom> zur Auswahl von x aus N Fragen einer Aufgabenseite (1 < x < N) verwenden, wurde im Abschnitt zur Aufgabenformular-Erstellung die Verwendung des Selektionssubmits empfohlen.

Zur Erinnerung: Sie sehen einen einzigen Einsende-Button (mit Key `einsendenSelektion`) vor, der die Eingaben zu mehreren (per `Selektion` ausgewählten) Teilaufgaben in einem Schritt einsendet.

In diesem Abschnitt wurde bereits beschrieben, dass es zu jeder Teilaufgabe eine Quittungsvorlage gibt. Normalerweise erfolgt ja eine Einsendung immer zu genau einer Teilaufgabe, und der Eingang der Eingaben wird dann unter Ausfüllen der Quittungsvorlage zu dieser Teilaufgabe bestätigt/quittiert.

Da der Selektionssubmit nun aber die Eingaben zu *mehreren* Teilaufgaben gleichzeitig vornimmt,

muss auch die ausgelieferte Quittung nun die Eingaben zu mehreren Teilaufgaben zusammenstellen. Es ist auch im Allgemeinen nicht handhabbar, für jede mögliche Kombination von Teilaufgaben jeweils eine eigene Quittungsvorlage zu erstellen. Vielmehr setzt das Online-Übungssystem nach einem Selektionssubmit die Quittung aus den Quittungsvorlagen der selektierten Teilaufgaben *dynamisch* zusammen. Einfach gesprochen werden dazu die Quittungsausgaben der eingesandten Teilaufgaben einfach hintereinander gehängt.

Betrachten wir dazu eine typische Quittungsvorlage wie die Folgende:

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Best&auml;tigung Ihrer Einsendung</title>
</head>
<body>
  <div style="background-color:#ccc; padding: 0.2em">
    Kurs $KursNr, &bdquo;$Kursname&ldquo;, $Versionsname<br>
    Aufgabenheft $AufgabenheftNr, Aufgabe $AufgabenNr
  </div>
  <h1>Best&auml;tigung Ihrer Einsendung</h1>
  $EMBED
  <p>von $Vorname $Nachname, Matrikelnr. $MatrikelNr<br>
  f&uuml;r &bdquo;$Kursname&ldquo;, $KursNr<br>
  Aufgabenheft $AufgabenheftNr, Aufgabe $AufgabenNr</p>
  <hr>
  <h2>Ihre Eingabe zu Frage 1</h2>
  <p>$Feld1</p>
  <p>(Sie können Ihre Eingaben noch bis zum Einsendeschluss überarbeiten.)</p>
  $/EMBED
  <hr>
  <a
href="$WebAssignServer/$Veranstaltername/StudentenStartSeite/$KursNr/$VersionsN
r/">Zur&uuml;ck zur Aufgaben&uuml;bersicht</a>
</body>
</html>
```

Angenommen, Sie haben eine Aufgabenseiten erstellt, die 6 Fragen enthält, und jeder Teilnehmer soll 3 dieser Fragen beantworten. Dazu wurde eine Teilaufgabe pro Frage erzeugt und ein Selektionssubmit angelegt, der die Eingaben zu den zufällig ausgewählten 3 der 6 Teilaufgaben einsendet. Die Quittungen der 6 Teilaufgaben A bis F seien alle wie obiges Beispiel aufgebaut.

Im Normalfall wird den Einsendern *genau einer* Teilaufgabe dann einfach der zwischen `$EMBED` und `$/EMBED` stehende Teil der Quittung ins aktuelle (bzw. in den Kursparametern eingestellte) Webdesign eingebettet angezeigt. (Im Fehlerfall würde die gesamte Seite mit den Fallback-Inhalten, die hier vor und hinter den EMBED-Marken stehen, ausgeliefert.)

Wenn nun in diesem Beispiel ein Selektionssubmit für 3 Teilaufgaben stattfindet, würde die erzeugte Quittung im Normalfall (mit Embedding) zumindest die Teile zwischen den EMBED-Marken aus allen drei Teilquittungen hintereinander hängen. Die Quittung für einen Selektionssubmit, bei dem für einen Studenten beispielsweise die drei Fragen 1, 4 und 5 zufällig ausgewählt wurden, sähe dann in etwa wie folgt aus:

> ***Bestätigung Ihrer Einsendung***
>
> *von Max Mustermann, Matrikelnr. 1234567*
> *für „Testkurs“, 01234*
> *Aufgabenheft 1, Aufgabe 2*
>
> ---
>
> ***Ihre Eingabe zu Frage 1***
> *Hallo Welt*
>
> *(Sie können Ihre Eingaben noch bis zum Einsendeschluss überarbeiten.)*
>
> *von Max Mustermann, Matrikelnr. 1234567*
> *für „Testkurs“, 01234*
> *Aufgabenheft 1, Aufgabe 2*
>
> ---
>
> ***Ihre Eingabe zu Frage 4***
> *Dies ist meine zweite Antwort…*
>
> *(Sie können Ihre Eingaben noch bis zum Einsendeschluss überarbeiten.)*
>
> *von Max Mustermann, Matrikelnr. 1234567*
> *für „Testkurs“, 01234*
> *Aufgabenheft 1, Aufgabe 2*
>
> ---
>
> ***Ihre Eingabe zu Frage 5***
> *Kommen wir zum Abschluss…*
>
> *(Sie können Ihre Eingaben noch bis zum Einsendeschluss überarbeiten.)*

Was dieses Beispiel illustrieren soll, ist, dass eine solche Quittungsvorlage aus verschiedenen Teilen besteht: Der „Kern“, nämlich die Wiedergabe der eingegangenen Einsendungen, muss auf jeden Fall pro eingegangener Teilaufgabeneinsendung ausgefüllt werden. Der *Kopfbereich* dagegen, der hier Angaben zum Studenten, Kurs und Heft zusammenstellt, sollte eben *nicht* (wie in obigem Beispiel) für jede Teilaufgabe wiederholt werden, sondern nur genau *einmal* am Beginn der erzeugten Quittung stehen. Weiterhin kann eine Quittung einen *Fußbereich* enthalten, in diesem Beispiel den Absatz »(Sie können Ihre Eingaben noch bis zum Einsendeschluss überarbeiten.)«, der auch in der erzeugten Selektionssubmit-Quittung nicht nach jeder Teilaufgabenquittung, sondern nur am Fuße der Gesamtquittung stehen sollte.

Die erzeugte Gesamtquittung sollte also besser wie folgt aussehen:

> ***Bestätigung Ihrer Einsendung***
>
> *von Max Mustermann, Matrikelnr. 1234567*
> *für „Testkurs“, 01234*
> *Aufgabenheft 1, Aufgabe 2*
>
> ---
>
> ***Ihre Eingabe zu Frage 1***
> *Hallo Welt*
>
> ***Ihre Eingabe zu Frage 4***
> *Dies ist meine zweite Antwort…*
>
> ***Ihre Eingabe zu Frage 5***
> *Kommen wir zum Abschluss…*
>
> *(Sie können Ihre Eingaben noch bis zum Einsendeschluss überarbeiten.)*

Um das zu erreichen, können Sie diejenigen Bereiche Ihrer Quittungsvorlage, die *nicht* zum Kerninhalt gehören, mit einer der beiden folgenden Klassen ausstatten:

Klasse	Bedeutung
`quittung-header`	Wird nur für die erste Teilquittung ausgegeben, bei allen folgenden Teilquittungen unterdrückt.
`quittung-footer`	Wird nur für die letzte Teilquittung ausgegeben, bei allen vorhergehenden Teilquittungen unterdrückt.

Es handelt sich hierbei explizit um Klassen statt IDs. Damit ist es möglich, mehr als einen Teilbereich mit einer solchen Klassen auszustatten.

Das obige Beispiel der Quittungsvorlage wäre also z.B. wie folgt anzupassen, um den gewünschten Effekt zu erzielen, dass Name, Kurs- und Heftangaben nur einmalig oben in der erzeugten Gesamtquittung stehen, und der abschließende Absatz auch nur einmal unterhalb allen Eingaben:

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Best&auml;tigung Ihrer Einsendung</title>
</head>
<body>
  <div class="quittung-header">
    <div style="background-color:#ccc; padding: 0.2em">
      Kurs $KursNr, &bdquo;$Kursname&ldquo;, $Versionsname<br>
      Aufgabenheft $AufgabenheftNr, Aufgabe $AufgabenNr
    </div>
    <h1>Best&auml;tigung Ihrer Einsendung</h1>
  </div>
  $EMBED
  <div class="quittung-header">
    <p>von $Vorname $Nachname, Matrikelnr. $MatrikelNr<br>
    f&uuml;r &bdquo;$Kursname&ldquo;, $KursNr<br>
    Aufgabenheft $AufgabenheftNr, Aufgabe $AufgabenNr</p>
    <hr>
  </div>

  <h2>Ihre Eingabe zu Frage 1</h2>
  <p>$Feld1</p>

  <div class="quittung-footer">
    <p>(Sie können Ihre Eingaben noch bis zum Einsendeschluss
überarbeiten.)</p>
  </div>
  $/EMBED
  <div class="quittung-footer">
    <hr>
    <a
href="$WebAssignServer/$Veranstaltername/StudentenStartSeite/$KursNr/$VersionsN
r/">Zur&uuml;ck zur Aufgaben&uuml;bersicht</a>
  </div>
</body>
</html>
```

Hinweise:

- Die Anwendung dieser Klassen auf die Fallbackinhalte außerhalb des Embedding-Bereichs ist nicht unbedingt nötig, da diese Inhalte normalerweise ohnehin nicht angezeigt werden, aber so ist es sauberer, und so wird auch demonstriert, dass es jeweils mehr als einen Block dieser Klassen geben darf.
- Diese Klassen dürfen natürlich grundsätzlich in allen Quittungsseiten zur Markierung von Kopf- und Fußinhalten verwendet werden, die hier beschriebene Wirkung entfalten sie aber nur beim Selektionssubmit.
- Falls Sie auch noch „Shuffle“ einsetzen, also eine zufällige Fragenreihenfolge, werden die Teilquittungen in der Reihenfolge zusammengesetzt, in der die entsprechenden Fragen/Teilaufgaben dem Studenten im Aufgabenformular angezeigt wurden. D.h. er erhält die Quittungen zu den einzelnen Fragen in derselben Reihenfolge, in der ihm auch in der Aufgabenseite die Fragen selbst präsentiert wurden.

Quittungen bei Multiple-Choice-Randomisierung

Falls Sie, wie im Abschnitt Blocklokale Randomisierung beschrieben, die Antwortalternativen zu Mehrfach- oder Einfachauswahlfragen (oder auch Zuordnungsfragen) ebenfalls randomisieren und dabei (über Hidden Inputs) eine Abbildung von internen auf angezeigte Antwortkennungen eingeführt haben, ist bei Quittungsseiten, die nicht direkt die automatische Vorkorrektur anzeigen, sondern einfach nur den Eingang der Eingaben quittieren, folgendes zu beachten:

Zu einer solchen Auswahlfrage werden ja zwei Eingabefelder verwendet, das erste (z.B. `Feld1`) für die studentische Eingabe, gebildet aus den Values der Checkboxen oder Radiobuttons, das zweite (`Feld2`) für das Mapping, das zu jeder in der Aufgabenseite präsentierten Antwortoption eine Abbildung von der internen Kennung auf die dem/-r Studenten/-in angezeigte Kennung enthält.

Dieses zweite Feld (hier `Feld2`) ist in der Quittung normalerweise gar nicht mit auszugeben, sondern nur das erste Feld mit der Einsendung (`Feld1`). Würde man es aber einfach nur per Variable `$Feld1` anzeigen, so gäbe die Quittung ja nur die – dem/-r Studenten/-in unbekannte – interne Antwortkennung wieder, damit könnte er/sie nicht viel anfangen. Binden Sie daher die Feldvariable mit dem Suffix `AAFF` (für Anzeige-Abbildung aus dem Folgefeld) ein. D.h. die Variable `$Feld1AAFF` wird die in Feld 1 vorliegende Einsendung auswerten, aber jede Antwortkennung aus dieser Einsendung gemäß dem Mapping aus Feld 2 (immer dem Feld mit der nächstgrößeren Nummer, dem „Folgefeld") konvertieren, d.h. jede interne Antwortkennung aus Feld1 gegen die dieser durch die Abbildung aus Feld 2 zugeordnete individuelle Antwortkennung austauschen. Auf diese Weise sieht der/die Teilnehmer/-in dann wieder genau die Kennungen (Buchstaben) zu den Antworten, die sie/er in der Aufgabenseite markiert hatte.

Falls Sie eine Sofortauswertung über das Vorkorrekturmodul `Teilaufgabenbewerter` verwenden, ist in der Quittungsseite selbst nichts zu beachten, sondern wie gehabt nur die Variable `$Vorkorrektur` einzufügen. Die Konfiguration, dass die „Anzeigeabbildung aus dem Folgefeld" in der Vorkorrektur genutzt werden soll, ist dann vielmehr in den Properties des Vorkorrekturmoduls vorzunehmen, siehe Konfiguration der internen Module.

Quittungen bei Zuordnungsfragen-Randomisierung

Während Sie bei den „normalen" Einfach- und Mehrfachauswahlfragen analog zu den Beispielen 1 und 2 aus Abschnitt Blocklokale Randomisierung nichts weiter beachten müssen, also insbesondere die Randomisierung nicht nochmal in der Quittungsschablone analog nachbauen müssen, sieht die Sache bei den Zuordnungsfragen schon anders aus, und zwar immer dann, falls Sie die *Tabellenzeilen* einer Zuordnungsfrage randomisiert haben.

Hier kann nicht, wie bei der Fragenrandomisierung selbst, auf einen Selektionssubmit zurückgegriffen werden: Die Zuordnungsfrage bildet immer eine Teilaufgabe. Selbst wenn nicht jeder Teilnehmer alle „Unter-Fragen" (Zeilen der Zuordnung) beantworten soll, sondern ihm zufällig eine Teilmenge davon präsentiert wird, wird immer eine vollständige Einsendung zu allen Eingabefeldern der Teilaufgabe und somit auch zu allen Zeilen/Unterfragen erzeugt – zu denjenigen, wo keine Eingabe möglich war (da die Frage durch Randomisierung ausgeblendet wurde), wird eine leere Einsendung erzeugt. Diese leeren Eingaben zu gar nicht gestellten Fragen möchten Sie in der Regel aber nicht quittieren.

Und selbst wenn Sie zwar keine *Zeilenauswahl* treffen, aber die Zeilen in *Zufallsreihenfolge* in der Aufgabe anzeigen, ist es ja wünschenswert, wenn die Einsendungen dazu in der Quittungsseiten möglichst in derselben Reihenfolge stehen wie die Zeilen in der Tabelle der Aufgabenseite.

Daher empfiehlt es sich hier, innerhalb der Quittungsseite die Randomisierung nochmal analog zum

Aufgabenformular durchzuführen.

Die Quittung zum Beispiel 3 von Blocklokale Randomisierung kann z.B. wie folgt aussehen:

```
$Fixed1
  <h2>ShuffleRows</h2>
  <p>Ihre Antworten:</p>
  <ol>
    $Randomize(3, Shuffle)
    $Random.1
      <li>Pünktchen und $FeldA2</li>
    $/Random.1

    $Random.2
      <li>Die dicke $FeldA3</li>
    $/Random.2

    $Random.3
      <li>Gaius Julius $FeldA4</li>
    $/Random.3
  </ol>
$/Fixed1

<h2>ShuffleCols</h2>
<p>Ihre Antworten:</p>
<ol>
  <li>Frage 1: $FeldA5AAFF</li>
  <li>Frage 2: $FeldA7AAFF</li>
  <li>Frage 3: $FeldA9AAFF</li>
</ol>
```

Für das zweite Teilbeispiel („ShuffleCols“), bei dem nur die Spalten randomisiert wurden, genügt die Verwendung der AAFF-Feldvariablen.

Für das erste Teilbeispiel („ShuffleRows“), bei dem nur die Zeilen randomisiert wurden, wurde die Blockstruktur vom Aufgabenformular übernommen. **Wichtig** ist dabei, dass alle Blocknummern (insb. auch die 1 von `$Fixed1` in diesem Beispiel) identisch zu denen der korrespondierenden Blöcke der Aufgabenseite sein müssen. Andere Nummern könnten zu abweichender Randomisierung führen!

Einsendetests

Nachdem Sie Aufgabenformular und Quittungsvorlage(n) erstellt haben, können Sie nun erste Tests vornehmen:

Durch Aufrufen der Vorschau der Aufgabenseite erhalten Sie ein immer leeres Formular, in dem Sie als Aufgabenautor ohne anderen Login etwas einsenden können[19]. Sie können das leere Eingabeformular ganz oder teilweise ausfüllen, einsenden und überprüfen, ob die daraufhin angezeigte Quittung Ihre Eingaben enthält und auf die gewünschte Weise darstellt.

Ein paar *typische Fehler*:

- *Bestimmte Eingaben werden in der Quittung gar nicht angezeigt.* Einige mögliche Ursachen:
  - In der Quittungsseite fehlen entsprechende `$Feld...`-Variablen.

    - Die Eingaben erfolgten in Eingabefeldern, die nachträglich zur Aufgabenseite hinzugefügt wurden, und es wurde vergessen, anschließend die Funktion »Teilaufgaben und Eingabefelder erzeugen« auszuführen.
    - Die Eingabefelder im Aufgabenformular sind falsch benannt oder wurden einer falschen Teilaufgabe zugeordnet (Kontrolle per *»Vorlage analysieren«*).
- *Bei leer gelassenem Eingabefeld wird nicht der Text angezeigt, der im `data-if-empty`-Attribut eingetragen wurde.*
    - Auch hier wurde offenbar vergessen, die Funktion »Teilaufgaben und Eingabefelder erzeugen« auszuführen.
- *Die Eingaben werden nicht korrekt dargestellt.* Typische Ursache: `$Feld...`-Variable trägt nicht das passende Suffix für die einsendbaren Inhalte oder der HTML-Code, in dem sie steht, ist unpassend:
    - Beispiel 1: *Bei Eingabe von `1 < 6 > 2` in ein Textfeld wird nur `1 2` o.ä. angezeigt.* Abhilfe: Verwenden Sie eine `$Feld...P`-Variable, damit die Kleiner- und Größer-Symbole auch als solche angezeigt und nicht als Tagklammern interpretiert werden.
    - Beispiel 2: *Bei Einsendung von Plaintext via normaler Textarea werden in der Quittung eingegebene Zeilenumbrüche oder Folgen von Leerzeichen nicht angezeigt,* sondern der ganze Text zu einem Absatz und Leerzeichenfolgen zu einzelnen Leerzeichen zusammengezogen. Grund: Die `$Feld...`-Variable steht nicht in einem `pre`-Element!
    - Beispiel 3: *Vom Studenten (per WYSIWYG-Editor) eingegebener formatierter Text wird als HTML-Quellcode angezeigt.* Grund: Einbindung per `$Feld...P`-Variable. Entfernen Sie das `P`-Suffix.
- Fehlermeldung beim Einsendeversuch: *HTTP-Protokollfehler in WebAssignMultipartPostService! Multipart-Boundary-String nicht gefunden in Content-Type: application/x-www-form-urlencoded*
    - Wahrscheinliche Ursache: Sie haben im URL des `form`-Tags den Servicenamen `EinsendungMultipart` eingetragen, jedoch *nicht* das Attribut `enctype="multipart/form-data"` angegeben.
- Fehlermeldung *Fehlerhafte Aufgabeneinrichtung! Request enthält keinen 'einsenden_'-Key, Aufgabenformular nicht korrekt!* Mögliche Ursachen:
    - Der `name` des zum Einsenden betätigten Submit-Buttons beginnt nicht mit `einsenden`.
    - Sie haben im `form`-Tag ein `enctype`-Attribut angegeben, obwohl Sie den normalen `Einsendung`-Service im `action`-URL referenzieren.
- *Beim Versuch, eine Datei einzusenden, wird in der Quittung nur der Dateiname statt des Dateiinhalts angezeigt.*
    - Wahrscheinliches Problem: Sie haben ein einfaches Text-Formular (Service `Einsendung` ohne `enctype`-Attribut) statt eines Multipart-Formulars (Service `EinsendungMultipart` mit `enctype="multipart/form-data"`) in der Aufgabenseite verwendet. Damit sind keine Dateieinsendungen möglich.

Zum Test, ob auch existierende Einsendungen bei erneutem Aufruf der Aufgabenstellung korrekt in der Aufgabenseite wieder eingeblendet werden, müssen Sie sich direkt über die Kursstartseite unter »Studentenzugang« mit einem Teststudenten anmelden. In dieser Phase wird ein Teststudent mit Sonderrechten empfohlen, typischerweise der `7777777`-Login. Die Sonderrechte erlauben ein Bearbeiten der Aufgabe auch außerhalb des Bearbeitungszeitraums und bei gesetzter Sperre (die es echten Studenten nicht erlaubt, auf die ggf. noch unfertigen Aufgaben zuzugreifen).

Korrekturvorlage erstellen

Sind Aufgabenformular und Quittungsvorlage(n) fertig, fehlt noch mindestens die Korrekturvorlage.

Eine typische Korrekturseite enthält folgende Abschnitte:

1. Kopf mit Angabe des Studenten (Name, Matrikelnr), ggf. des Korrektors (bei manueller Korrektur) und Korrekturdatums, und natürlich mit der Angabe, um welche Aufgabe es geht (vorzugsweise mit Kursnummer und Semester).
2. Einsendungen des Studenten.
3. Ggf. Vorkorrekturen (vgl. Einsatzszenario für Vorkorrekturmodule <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#vkmodule>) oder Autokorrektur.
4. Bei manueller Korrektur werden in der Regel Abschnitte für Korrektor-Kommentare vorgesehen.
    - Bei In-Browser-Korrektur <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#ibk> werden dazu Korrekturformularelemente in einer speziellen Syntax eingebunden. Bei der Korrektur werden diese dann für den Korrektor in ein HTML-Formular konvertiert, bei der Ansicht einer Korrektur durch Betreuer oder Studenten werden an deren Stelle nur die Eingaben des Korrektors angezeigt.
    - Bei herkömmlicher HTML-Korrektur <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#htmlk> kann der Korrektor ohnehin an beliebiger Stelle in einer Korrekturseite Text einfügen. Dennoch ist es üblich, Kommentarabschnitte mit Überschrift und einem vom Korrektor zu ersetzenden Platzhaltertext vorzusehen.
5. Anzeige der Bewertung, z.B. »Erreichte Punkte: 5 von 10«

Sie können sich auch hier wieder per Knopfdruck eine Korrekturseite autogenerieren lassen und diese anschließend nachbearbeiten. (Genau wie vor der Generierung der Quittungsseiten auch, sollte zuerst die Funktion Teilaufgaben und Eingabefelder erzeugen ausgeführt worden sein, dann werden automatisch die in Punkt 2. benötigten Feldvariablen eingefügt. Die automatisch erzeugte Korrekturseite für das begleitende Fallbeispiel hat etwa folgenden Aufbau:

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Einsendung zu &quot;$Kursname&quot;, Aufgabe
$AufgabenheftNr.$AufgabenNr</title>
</head>
<body>
  <div style="background-color:#ccc; padding: 0.2em">
    Kurs $KursNr, &bdquo;$Kursname&ldquo;, $Versionsname<br>
    Aufgabenheft $AufgabenheftNr, Aufgabe $AufgabenNr
  </div>
  <h1>Einsendung zu $Aufgabenname</h1>
  $EMBED
  <p><strong>Einsendung von</strong> $Vorname $Nachname, Matrikelnr.
$MatrikelNr</p>
  <p><strong>Korrektor:</strong> $Korrektor</p>
  <p><strong>Datum:</strong> $KorrekturDatum</p>
  <hr>
  <h2>Aufgabe 1 a)</h2>
  <h4>Eingabe in Feld 1:</h4>
  <pre>$FeldA1P</pre><p><hr>
  <h4>Eingabe in Feld 2:</h4>
  <pre>$FeldA2P</pre><p><hr>
  <h3>Anmerkungen</h3>
  <p>{hier bitte zus&auml;tzl. Anmerkungen eintragen}</p>
  <h3>Erreichte Punktzahl f&uuml;r Teil a: {hier bitte die erreichte Punktzahl
eintragen}</h3><p><hr>
  <h2>Aufgabe 1 b)</h2>
  <h4>Eingabe in Feld 1:</h4>
  <pre>$FeldB1P</pre><p><hr>
  <h3>Anmerkungen</h3>
  <p>{hier bitte zus&auml;tzl. Anmerkungen eintragen}</p>
  <h3>Erreichte Punktzahl f&uuml;r Teil b: {hier bitte die erreichte Punktzahl
eintragen}</h3><p><hr>
  $/EMBED
</body>
</html>
```

Auch diese Seite können Sie analog zur Quittungsseite nach Belieben anpassen und die generischen Überschriften wie »Aufgabe 1 a)« oder »Eingabe in Feld 1:« durch sprechenderen Text passend zur Aufgabenstellung ersetzen.

Im Folgenden sei genauer auf den oben aufgelisteten Grundaufbau einer Korrekturseitenvorlage mit Blick auf dieses Beispiellisting eingegangen:

Zu 1.: Schon unter Quittungsvorlage erstellen wurde für Quittungen ein solcher Kopf vorgeschlagen, in Korrekturen ist er noch wichtiger. Falls Sie *keinen* solchen Kopf einfügen, aber Embedding (`$EMBED`) mit den Standard-Seitenvorlagen des Online-Übungssystems verwenden, wird durch die Korrektur-Seitenlayout-Vorlage automatisch ein solcher Kopf eingefügt (in der Studenten-Ansicht), aber wenn Sie ihn selbst einfügen, sind sie nicht nur auf der sicheren Seite, sondern können sein Layout auch selbst beeinflussen.

Zu 2.: Dieser Teil der Korrekturseitenvorlage sieht fast genauso aus wie eine Quittungsvorlage, mit folgenden Unterschieden:

- Bei Aufgaben, die aus mehreren (technischen) Teilaufgaben besteht, gibt es dennoch wieder nur eine einzige Korrekturseite, in der alle Einsendungen zu allen Teilaufgaben zusammengefasst werden.
- Entsprechend sind die `$Feld*`-Variablen – anders als in Quittungen – wieder mit der Teilaufgaben-Referenz auszustatten, müssen also genauso heißen, wie die Feldnamen im Aufgabenformular, lediglich mit vorangestelltem `$`.

Ansonsten gilt das oben zum Erstellen von Quittungsvorlagen bereits Gesagte, insbesondere im Hinblick auf die Suffixe wie `P` und die Einbindung in `pre`-Elemente. Quittungs- und Korrekturvorlagen sollten hier zueinander konsistent gehalten werden, d.h. gleiche Felder in beiden Vorlagen gleich formatiert sein.

Zu 3.: Siehe nachfolgenden Unterabschnitt zu Vorkorrektur- und Autokorrektur-Variablen.

Zu 4.: Automatisch erzeugte Korrekturvorlagen *sehen noch keine In-Browser-Korrektur vor*, sondern lediglich einen Anmerkungs-Abschnitt pro Teilaufgabe. Das Einfügen von In-Browser-Korrekturelementen wird empfohlen, siehe separates Handbuch <https://online-uebungssystem.fernuni-hagen.de/download/WYSIWYG_IBK/WYSIWYG_IBK.html> !

In der Regel, gerade bei In-Browser-Korrektur, ist 4. kein alleinstehender Abschnitt in der Korrekturseite, sondern die Korrektorkommentare werden normalerweise an verschiedenen Stellen zwischen den Einsendungen und ggf. Vorkorrekturen möglich sein.

Zu 5.: Die Datenbank des Online-Übungssystems speichert nur eine einzige Punktzahl zur gesamten Aufgabe: die erreichten Gesamtpunkte. Sollen dem Studenten Teilpunktzahlen für einzelne Fragen oder Teilaufgaben bekanntgegeben werden, kann dies nur durch entsprechende Texteinträge in der Korrekturseite erfolgen. Die oben abgebildete erzeugte Korrekturvorlage sieht z.B. in der Korrekturseite manuelle Einträge der erreichten Teilpunkte pro Teilaufgabe vor. Nach Speicherung der Korrekturseite muss der Korrektor noch die erreichte Gesamtpunktzahl über ein Status-Formular in der Datenbank speichern. Soll diese Gesamtpunktzahl auch in der Korrekturseite angezeigt werden, kann dafür die Variable `$KorrekturPunkte` verwendet werden – dann muss der Korrektor die Punktzahl nicht redundant an zwei Stellen eintragen. Folgendes Markup wird z.B. in automatisch erzeugte Korrekturseitenvorlagen eingebunden, wenn die Aufgabe *nicht* in Teilaufgaben zerlegt ist:

```
<h3>Anmerkungen</h3>
<p>{hier bitte zus&auml;tzl. Anmerkungen eintragen}</p>
<h3>Erreichte Punktzahl: $KorrekturPunkte von $MaximalPunkte</h3>
```

So etwas können Sie natürlich auch dann in Ihre Korrekturseite schreiben, wenn Sie Teilaufgaben verwenden.

Variablen für Korrekturseiten

Sie können auch in Korrekturseiten alle im Abschnitt Variablen des Online-Übungssystems beschriebenen Standard-Variablen verwenden.

Weiterhin stehen auch fast alle in Variablen für Quittungsseiten beschriebenen Variablen zur Verfügung, mit folgenden Ausnahmen bzw. Besonderheiten:

- Die Variable `$TeilaufgabenNr` ist nicht definiert, da eine Korrektur keinen Teilaufgabenbezug hat und
- bei `$Feld...`-Variablen ist zu beachten, dass – anders als in Quittungsseiten – die

Teilaufgabenkennung mit anzugeben ist, s.o.

- Bei Feldern für *Dateieinsendungen* sind einige Besonderheiten zu beachten und wurden einige neue Modi für Feldvariablen speziell für Korrekturseiten eingeführt, siehe Feldvariablen für Dateieinsendungen
- Insbesondere gibt es ein neues Suffix `ABKUPL` für Feldvariablen, siehe Dateianhangs-Upload für Korrektoren.

Im Folgenden seien nur noch Variablen aufgelistet, die nicht in den vorherigen Abschnitten bereits erläutert wurden.

Korrektur allgemein

Variable	Bedeutung	Beispiel
`$Korrektor`	Name des Korrektors. (Der festgelegte „öffentliche“ Anzeigename.)	Klaus Klug
`$KorrekturDatum`	Datum, an dem der Korrektor die Korrektur zuletzt gespeichert hat.	02.03.2014
`$KorrekturPunkte`	Vom Korrektor vergebene Gesamtpunktzahl	20
`$KorrekturNote`	Vom Korrektor für die Aufgabe vergebene Note – nur definiert, wenn für das Aufgabenheft eine Einzelbenotung pro Aufgabe eingestellt ist.	2,7

Anmerkung: Variablen in Korrekturseiten werden zu *zwei verschiedenen Zeitpunkten* durch konkrete Werte ersetzt: Die meisten Variablen einschließlich der `$Feld...`-Variablen, Vor- und Autokorrekturen, Studentendaten etc. werden direkt beim sog. Heft-Schließen, also beim Erzeugen der individuellen Korrekturseite für einen Studenten, ersetzt, so dass ein Korrektor später auch statt der Variablen die zu korrigierenden Einsendungen etc. vorfindet. Obige Variablen zur Korrektur selbst werden dagegen natürlich nicht schon vor der Korrektur ersetzt, sondern bleiben während des Korrekturprozesses als Variable erhalten und werden erst bei Anzeige der fertigen Korrektur für den Studenten durch konkrete Werte gefüllt.

Vorkorrekturen und Autokorrekturen

Vorkorrekturen werden in Korrekturseiten fast genauso eingebunden wie in eine Quittungsseite auch, insofern gilt das im Abschnitt zu Vorkorrektur-Variablen für Quittungen Gesagte – mit einem wesentlichen Unterschied:

Da Vorkorrekturen sich auf eine konkrete Teilaufgabe beziehen, muss im Rahmen der Korrekturseite zu jeder `$Vorkorrektur...`-Variable eine Teilaufgabenreferenz (`A` bis `Z`) hinzugefügt werden. Dies geschieht durch Anhängen am Ende (aber noch vor einem optionalen Vorkorrekturnamen in Klammern), d.h. die Variablen heißen nun für z.B. Teilaufgabe `A`:

- `$VorkorrekturA`
- `$VorkorrekturenA`
- `$VorkorrekturHREFA(name)`, `$VorkorrekturIMGA(name)` bzw. `$VorkorrekturURLA(name)`

Es ist *auch* möglich, schlicht die Variable `$Vorkorrektur` einzufügen, die dann einfach *alle* für den Studenten gefundenen Vorkorrekturen zu sämtlichen Teilaufgaben der Aufgabe hintereinander ausgibt. (Dabei haben Sie aber natürlich weniger Gestaltungsmöglichkeiten als bei explizit getrennter Ausgabe der einzelnen Vorkorrekturen.)

Falls Sie ein Korrektur-/Bewertermodul <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#moduleoverview> verwenden, sollte natürlich die Ausgabe des Korrekturmoduls in die Seite eingefügt werden. Dies geschieht analog zur Einbindung einer Vorkorrektur, nur beginnen die Variablen nicht mit `$Vorkorrektur`, sondern schlicht mit `$Korrektur`, und enthalten keine Teilaufgabenreferenz, heißen also:

- `$Korrektur`
- `$Korrekturen`
- `$KorrekturHREF(name)`, `$KorrekturIMG(name)` bzw. `$KorrekturURL(name)`

Außerdem sind speziell bei Vorliegen von Autokorrekturen auch noch folgende Variablen verfügbar:

Variable	Bedeutung
`$AutokorrekturDatum`	Datum, an dem das Korrekturmodul seine Autokorrektur erzeugt hat
`$AutokorrekturPunkte`	Vom Korrektur-/Bewertermodul vergebene Bewertung in Punkten

Diese werden im Normalfall nicht benötigt und stimmen bei vollautomatisch bewerteten Aufgaben mit den oben genannten Variablen `$KorrekturDatum` bzw. `$KorrekturPunkte` überein. Ein Unterschied ergibt sich bei automatischer Bewertung mit Hand-Nachkorrektur <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#autohand> : Die beiden `$Autokorrektur...`-Variablen werden direkt bei Erzeugung der Korrekturseite durch die konkreten Werte ersetzt: Öffnet anschließend ein Korrektor die Korrektur, so sieht er hier die Punkte und das Datum der Autokorrektur, während `$KorrekturDatum` bzw. `$KorrekturPunkte` noch als Variablen in der Korrektur stehen, da deren Werte während der manuellen Nachkorrektur noch gar nicht feststehen, sondern erst (bei Korrektureinsicht) das Ergebnis der manuellen Nachkorrektur widerspiegeln sollen.

Feldvariablen für Dateieinsendungen

Für Felder, in denen Studenten Dateien hochladen sollen (die also im Aufgabenformular typischerweise in der Form `<input type="file" name="FeldA1">` definiert wurden), sollte in der Korrekturseite im Normalfall *immer* der Automatikmodus verwendet werden, d.h. das Feld ist einfach per `$FeldA1` – *ohne* weiteres Suffix (wie `HREF`) – in die Korrekturseite einzubinden.

Dieser Automatikmodus bewirkt dann folgendes Verhalten:

1. Wird die Korrekturseite von einem Korrektor geöffnet, sieht er einen Dateilink und ein Dateiupload-Feld. Über den Link kann der Korrektor die vom Studenten eingesandte Datei herunterladen. Dann kann er optional die Datei bearbeiten (z.B. durch Hinzufügen von Markierungen oder Kommentaren) und über das Dateiupload-Feld kann er die bearbeitete/annotierte Version der Einsendung wieder hochladen.
    - Durch einen solchen Upload wird eine so genannte *Beikorrektur* <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#bk> gespeichert. Die Originaleinsendung des Studenten wird dadurch nicht überschrieben, aber im Folgenden zeigt der Downloadlink in der Korrekturseite immer auf die Beikorrektur, also die vom Korrektor bearbeitete Version der Einsendung.
    - Eine *Ausnahme* von der oben beschriebenen Verlinkung kann vorliegen, wenn der studentische Upload aus einer Textdatei besteht. Diese wird im Automatikmodus dann in der Regel nicht verlinkt, sondern ihr Textinhalt direkt (analog zu Texteingabe in Eingabeboxen) in die Korrekturseite aufgenommen. Wenn Sie gezielt bestimmte

Textdateiformate wie HTML, CSS, CSV etc. als Uploads entgegennehmen möchten und sicherstellen möchten, dass diese nicht in die HTML-Korrektur eingebettet, sondern in jedem Fall immer verlinkt werden wie Binärdateien auch, mitsamt der Uploadmöglichkeit für den Korrektor, dann verwenden Sie statt des Automatikmodus eine Variable der Art `$FeldA1UPLFORM` (s.u.)!

2. Ruft ein Betreuer oder der Student selbst die Korrektur auf, wird nur der Link angezeigt (ohne Uploadmöglichkeit). Dieser Link verweist auf die aktuelle Beikorrektur (vom Korrektor annotierte Einsendung), bzw. – falls der Korrektor nie eine solche hochgeladen hat – auf die Originaleinsendung.

Alternativ zu diesem Automatikmodus können Feld-Variablen auch mit bestimmten Suffixen kombiniert werden (teils schon in Feldvariablen für Quittungsseiten eingeführt). Dabei ist dann immer genau zu beachten, ob die Variable dann die Original-Einsendung oder die Beikorrektur verlinkt!

Variable	Bedeutung
`$FeldA1HREF`	Erzwingt die Ausgabe eines Links zur hochgeladenen Datei (Originaleinsendung). Als Linkbeschriftung wird (wie im Automatikmodus) der Dateiname verwendet. An dieser Stelle bekommt also weder der Korrektor die Möglichkeit zum Upload einer annotierten Version (Beikorrektur) noch der Student eine Möglichkeit zum Download einer Beikorrektur, sondern es wird immer die Originaleinsendung verlinkt.
`$FeldA1URL`	Gibt nur den URL zur Dateieinsendung aus, sollte also selbst innerhalb des `href`-Attributs eines (selbst beschrifteten) Links stehen, z.B. `<a href="$Feld1URL">Ihre hochgeladene Datei</a>`.
`$FeldA1IMG`	Erzwingt die Ausgabe der Originaleinsendung als `img`-Tag mit dem URL im `src`-Attribut. Sollte nur angewendet werden, wenn ausschließlich geeignete Dateiformate (i.W. JPEG, GIF und SVG) hochgeladen werden dürfen. Auch hier wird immer die Originaleinsendung ausgegeben, und der Korrektor bekommt an dieser Stelle keine Möglichkeit zum Upload einer Beikorrektur.
`$FeldA1UPLFORM`	Bei Dateiupload-Feldern ist dies i.d.R. identisch zum Automatikmodus (oder anders gesagt: Der Automatikmodus bei Angabe der Variablen `$FeldA1` ohne Suffix wählt bei vorliegenden Dateieinsendungen automatisch diesen Modus aus): Erzeugt zunächst (für alle Benutzerrollen) einen Link zum Download der Beikorrektur (bzw. Originaleinsendung, wenn keine Beikorrektur hochgeladen wurde) und gibt dem Korrektor weiterhin die Möglichkeit zum Upload einer Beikorrektur. Insgesamt kann ein Korrektor hier also die Dateieinsendung herunterladen, bearbeiten/annotieren und geändert wieder hochladen. Dieser Beikorrekturupload-Modus sollte nur für Dateiupload-Felder benutzt werden, und da genügt die Variable im Automatikmodus. **Unterschied zum Automatikmodus**: Der Automatikmodus kann für hochgeladene Textdateien entscheiden, diese nicht zu verlinken, sondern analog zu Eingaben in Textboxen direkt in die HTML-Korrekturseite aufzunehmen. Wenn Sie das verhindern möchten und sicherstellen möchten, dass jeder Upload, auch in Formaten wie z.B. CSV oder XML o.ä. *immer* verlinkt wird und mit der Beikorrekturupload-Möglichkeit für Korrektoren versehen wird, dann verwenden Sie dieses Variablensuffix. Ansonsten wird es kaum benötigt.
`$FeldA1BKHREF`	Wie `$FeldA1HREF`, allerdings wird ein Link zur Beikorrektur statt zur Originaleinsendung erzeugt. Damit überhaupt eine von der Originaleinsendung abweichende Beikorrektur existieren kann, muss die Korrekturseite zumindest *zusätzlich* eine Variable Uploadmöglichkeit bieten (Automatik-Modus per `$FeldA1` oder eben `$FeldA1UPLFORM`, s.o.). Da allerdings diese neben dem Upload-Form auch schon selbst einen Link zur Beikorrektur einbindet, besteht normalerweise für diese Variable (für zusätzliche Downloadlinks) kein Bedarf – es sei denn, Sie wollen explizit mehrere Downloadmöglichkeiten für dieselbe Beikorrektur an verschiedenen Stellen der Korrekturseite einbinden.
`$FeldA1BKURL`	Wie `$FeldA1URL`, allerdings wird der URL zum Beikorrektur-Dowload statt zum Download der Originaleinsendung ausgegeben. Siehe auch Anmerkungen zu `$FeldA1BKHREF`!
`$FeldA1BKIMG`	Wie `$FeldA1IMG`, nur wird die Beikorrektur an Stelle der Originaleinsendung als Bild (`img`) eingebunden. Siehe auch Anmerkungen zu `$FeldA1BKHREF`!
`$FeldA1BKUPLF`	Erzeugt in der Korrektorensicht ausschließlich eine Uploadmöglichkeit für den Korrektor und ist in allen anderen Ansichten unsichtbar. Im Normalfall (Automatikmodus oder `$FeldA1UPLFORM`) benötigen Sie diese Variable nicht, da eine entsprechende Uploadmöglichkeit dort (in der Korrektorensicht) *zusammen* mit dem Downloadlink erzeugt wird. Falls Sie aber explizit Downloadlinks selbst mit einer der oben genannten Variablen wie `$FeldA1BKHREF` einfügen und ein Korrektor dennoch die Dateieinsendung nicht nur herunterladen, sondern (in bearbeiteter Form) auch wieder als Beikorrektur hochladen können soll, ergänzen Sie Ihre selbst gestaltete Downloadmöglichkeit mittels dieser Variablen um eine Uploadmöglichkeit.

In aller Regel genügt zu einem Dateieinsendung-Feld also das Einbinden der Korrekturvariable im

Automatikmodus (`$FeldA1`).

Mit den oben beschriebenen Spezialvariablen haben Sie aber fortgeschrittene Möglichkeiten.

Falls Sie z.B. vorsehen wollen, dass die Originaleinsendung immer zusätzlich zur Beikorrektur in der Korrekturseite verlinkt sein soll, fügen Sie einfach zwei Variablen ein: `$FeldA1HREF` als Downloadlink zur Originaleinsendung und `$FeldA1` als Downloadlink zur Beikorrektur (mit der Möglichkeit für Korrekturen, Beikorrekturen hochzuladen). In diesem Fall muss die Beikorrektur dann auch nicht mehr unbedingt eine annotierte Fassung des Originals sein, in dem sämtliche Originaleingaben erhalten bleiben müssen, sondern kann auch eine ganz andere Datei sein, z.B. ein Bewertungsformular. Andererseits bietet sich für solche Zwecke auch der im folgenden Abschnitt beschriebene Dateianhangs-Upload für Korrektoren an.

Dateianhangs-Upload für Korrektoren

Neben „normalen“ Beikorrekturen zu Dateieinsendungen können Sie auch weitere Upload-Felder zu einer Korrektur vorsehen, zu denen keine Dateieinsendungen von Studenten existieren, sondern in denen ein Korrektor einfach eine weitere Datei als Dateianlage zur Korrektur hinzufügen kann. Auch diese werden als Beikorrekturen <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#bk> gespeichert und – in Abgrenzung zu Beikorrekturen zu Dateieinsendungen – auch als „angehängte Beikorrekturen“ bezeichnet.

Da jede Beikorrektur sich auf ein Eingabefeld des Aufgabenformulars bezieht, muss zunächst in der *Aufgabenseite* ein entsprechendes Eingabefeld eingefügt werden. Da in diesem Feld aber ein Student eben *keine* Dateieinsendung (oder andere Eingabe) vornehmen können soll, sondern das Feld einfach leer und für den Studenten unsichtbar bleiben soll und lediglich der *Korrektor* später zu diesem Feld eine Datei hochladen können soll, ist dazu typischerweise ein Hidden-Field zu verwenden (in die Aufgabenseite einzufügen):

```
<input type="hidden" name="FeldA2">
```

Die Kennung `A2` (zweites Eingabefeld der ersten Teilaufgabe) ist natürlich nur als Beispiel zu verstehen.

Dieses Input-Element ist notwendig, damit die Teilaufgaben- und Felderzeugung das Eingabefeld `FeldA2` erzeugt, dem der Korrektor später eine Datei zuordnen können soll. Dass es bei Einsendungen leer bleibt, unterscheidet es von „normalen“ Datei-Einsendefeldern, in denen ein Korrektor normalerweise nur annotierte Kopien der studentischen Einsendung speichert (und normalerweise auch nur dann etwas hochladen kann, wenn überhaupt eine Einsendung vorliegt).

Da der Student das Feld nicht sieht und nichts darin einsendet, wird es in der *Quittungsseite* gar nicht berücksichtigt.

In der *Korrekturvorlage* nun wird zu diesem Feld eine spezielle Variable eingefügt:

Variable	Bedeutung
`$FeldA2ABKUPL`	Erzeugt in der Korrektorsicht eine Uploadmöglichkeit für einen Dateianhang und einen Downloadlink, mit dem der Korrektor seinen ggf. schon hochgeladenen Anhang auch wieder herunterladen/einsehen kann. In der Studentensicht (und Korrekturvorschau f. Betreuer oder Korrektor) wird ein Downloadlink mit Standardbeschriftung angezeigt – sofern eine Datei vom Korrektor hochgeladen wurde. Liegt kein Dateianhang vor, wird auch kein Downloadlink generiert.
`$FeldA2ABKUPL{Downloadlink-Text}`	Dieselbe Funktionalität, legt aber zusätzlich eine Link-Beschriftung für den Downloadlink (in der Studentensicht bzw. Vorschau) fest.

Das Suffix `{Downloadlink-Text}` ist also optional. Wenn es angegeben wird, legt es die Beschriftung des Links fest, der später den Studenten an dieser Stelle in der Korrektur angezeigt wird und den Download der vom Korrektor hochgeladenen Datei („Beikorrektur") ermöglicht. Das Variablensuffix `ABKUPL` sorgt insb. für ein spezielles Upload-Feld, in dem der Korrektor auch dann Beikorrekturen hochladen kann, wenn keine Einsendung des Studenten vorliegt (im Gegensatz zu normalen `$Feld...`-Variablen bei Dateieinsendungsfeldern).

In dieser Reinform ist ein solches Dateiupload-Feld für jede neue Korrektur zunächst leer, ein Korrektor kann darin eine beliebige Datei pro Korrektur hochladen. Dieses Verfahren kann noch erweitert werden, indem dem Eingabefeld ein spezielles PDF-Formular <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#pdf_forms> zugeordnet wird. Diese Variable bietet dem Korrektor dann nicht nur eine Upload-Möglichkeit für eine Datei, sondern – so lange er noch keine Datei hochgeladen hat – auch eine Downloadmöglichkeit für dieses Formular. Mehr zu diesem Thema findet sich in einer gesonderten Dokumentation <https://online-uebungssystem.fernuni-hagen.de/download/PDFForms/KorrekturPdfFormulare.html> .

Variablen bei Mehrfachkorrektur

Siehe auch Abschnitt Korrekturmodi/Mehrfachkorrektur <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#mehrfachkorrektur> in Teil I des Handbuchs!

Für jede von einem Studenten bearbeitete Aufgabe wird immer genau eine Korrektur erstellt. Kern dieser Korrektur ist die (HTML-)Korrekturseite, deren Vorlagedatei in diesem Kapitel behandelt wird.

Bei *Einfachkorrektur* wird die Korrektor genau einem Korrektor zugeteilt, der die Korrekturseite bearbeiten kann und/oder Beikorrekturen erstellen zur Korrektur erstellen kann, wobei es sich um bearbeitete Kopien von Dateieinsendungen oder um eigene vom Korrektor zur Korrektur hinzugefügte Dateianhänge handeln kann, siehe vorhergehende Abschnitte. Neben diesen Dokumenten gehören auch noch Metadaten zur Korrektur (wie der Zustand, der Name des Korrektors, dem sie zugeteilt ist, die vergebene Punktzahl und/oder Note und das Datum der letzten Änderung der Korrektur).

Bei *Mehrfachkorrektur* sollen *mehrere* Korrektoren (parallel oder sequentiell) die Korrektur bearbeiten. Dazu werden so genannte Teilkorrekturen zur Korrektur erzeugt, und jede Teilkorrektur wird einem anderen Korrektor zur Bearbeitung zugeteilt. Die Teilkorrekturen umfassen wieder die oben genannten Metadaten (insb. vergibt jeder Korrektor eine eigene Teilpunktzahl und/oder Note, und die Gesamtpunktzahl bzw. Gesamtnote der Korrektur errechnet sich jeweils als Durchschnitt der Teilkorrekturergebnisse). Auch die Beikorrekturen werden pro Teilkorrektur dupliziert, d.h. jeder Korrektor bekommt seine eigene Kopie jeder Dateieinsendung, die er bearbeiten/annotieren kann,

und falls die Möglichkeit zum Upload eines Dateianhangs für einen Korrektor geboten wird, kann in dem Feld jeder Korrektor seinen eigenen Anhang hochladenden.

Es gibt aber dennoch auch bei Mehrfachkorrektur immer nur *eine* (HTML-)Korrekturseite. Das ist die Seite, die dem Studenten bei Ergebniseinsicht angezeigt wird. Diese zentrale Korrekturseite sollte alle Informationen über die einzelnen Teilkorrekturen zusammentragen, z.B. …

- wer die einzelnen Korrektoren sind,
- welche Teilpunktzahlen/-noten von den einzelnen Korrektoren vergeben wurden und
- wie die Gesamtpunktzahl/-note (Durchschnitt der Teilkorrektur-Wertungen) lautet.
- Da im Falle von Dateieinsendungen jeder Korrektor seine eigene Beikorrektur erstellen kann, sollte z.B. zu einem Dateieinsendungs-Feld statt eines einzelnen Downloadlinks für die Beikorrektur dann eine Liste von Links zu allen Beikorrekturen zur Dateieinsendung eingefügt werden.

Um das zu verwirklichen, gibt es teils neue Variablen, teils besondere Varianten oder Anwendungen bereits oben beschriebener Variablen.

Die `$Teilkorrekturen`-Schleife

Allgemein kann immer, wenn Informationen über alle Teilkorrektoren zur einer Korrektur ausgegeben werden sollen (z.B. alle Korrektoren, alle Teilbewertungen, Links zu den einzelnen Korrekturen), eine „Für-alle-Teilkorrekturen-gib-jeweils-Folgendes-aus"-Schleife konstruiert werden. Diese wird durch folgende Variablen definiert:

Variable	Bedeutung
`$Teilkorrekturen`	leitet den für jede Teilkorrektur einzeln auszugebenden Text ein,
`$/Teilkorrekturen`	beendet den zu wiederholenden Text.

Zwischen diesen beiden Variablen kann HTML-Code stehen, der wiederum alle bereits bekannten Variablen enthalten darf. Folgende Variablen beziehen sich dabei innerhalb dieses Blocks auf die jeweils auszugebende Teilkorrektur, ändern also in jeder Wiederholung des Blocks ihren Wert:

Variable	Bedeutung	Beispiel
`$Korrektor`	Öffentlicher Anzeigename des Korrektors der jeweiligen Teilkorrektur.	Klaus Klug
`$KorrekturDatum`	Datum, an dem der Korrektor die jeweilige Teilkorrektur zuletzt gespeichert hat.	02.03.2019
`$KorrekturPunkte`	Vom Korrektor zur jeweiligen Teilkorrektur vergebene Punktzahl	20
`$KorrekturNote`	Vom Korrektor zur jeweiligen Teilkorrektur vergebene Note – falls für das Aufgabenheft eine Einzelbenotung pro Aufgabe eingestellt ist.	2,7
`$TeilkorrekturNr`	Die Nummer der jeweiligen Teilkorrektur aus. (Die Zählung beginnt bei 1.)	2
`$TeilkorrekturNrErst`	Gibt für die ersten Teilkorrektur den Wortbeginn „Erst“ aus, für die zweite Teilkorrektur „Zweit“, für die dritte „Dritt“ etc.. Diese Variable wird typischerweise durch Anhängen eines Restwortes komplettiert z.B. `$TeilkorrekturNrErstprüfer` erzeugt für die einzelnen Teilkorrekturen die Wörter „Erstprüfer“, „Zweitprüfer“, „Drittprüfer“ etc.	Zweit
`$TeilkorrekturNrerst`	analog, aber mit Kleinbuchstaben beginnend. Damit lassen sich z.B. per `$TeilkorrekturNrerste Wertung` Ausgaben wie „erste Wertung“, „zweite Wertung“ etc. erzeugen.	zweit

Auch von den Feldvariablen für Dateieinsendungen sind alle Varianten, die sich auf eine Beikorrektur beziehen, innerhalb einer solchen Teilkorrekturen-Schleife jeweils unterschiedlich belegt: Da es zu jeder Dateieinsendung genau eine Beikorrektur (ggf. vom Korrektor bearbeitete Kopie der eingesandten Datei) pro Teilkorrektur gibt, bezieht sich eine solche `$FeldA1BK...`-Variable innerhalb einer Teilkorrektur-Schleife immer auf genau die Beikorrektur zur jeweils auszugebenden Teilkorrektur.

Die Variablen wurden oben bereits ausführlicher beschrieben und werden hier nur nochmals knapp zusammengestellt:

Variable	Bedeutung
`$FeldA1BKHREF`	Link zur Beikorrektur (komplettes `a`-Element mit Standardbeschriftung)
`$FeldA1BKURL`	URL zum Beikorrektur-Dowload (kann in ein `href`-Attribut eines `a`-Elements eingefügt werden, um einen Link mit eigener Beschriftung zu erzeugen)
`$FeldA1BKIMG`	Beikorrektur als Bild (`img`-Element mit Beikorrektur-URL als `src`)

Weiterhin gibt es – wie unten in Abschnitt Angehängte Beikorrekturen bei Mehrfachkorrektur noch genauer ausgeführt – eine spezielle Variable zum Verlinken einer erst vom Korrektor angehängten Beikorrektur (Dateianlage, der keine Dateieinsendung gegenübersteht, siehe vorhergehenden Abschnitt) innerhalb einer Teilkorrekturschleife:

Variable	Bedeutung
`$FeldA1ABKLNK`	Erzeugt einen Link zur vom Korrektor angehängten Beikorrektur – sofern eine solche vorhanden ist. Hat der Korrektor in diesem Feld keinen Upload vorgenommen, wird die Variable ersatzlos entfernt. In dieser einfachen Form erhält der Link eine Standardbeschriftung basierend auf dem Dateinamen der vom Korrektor hochgeladenen Datei.
`$FeldA1ABKLNK{Linktext}`	Wie oben, aber der in geschweiften Klammern angegebene Text wird als Linkbeschriftung verwendet. Falls darin das Teilwort „Erst“ oder „erst“ vorkommt, wird das abhängig von der Nummer der gerade ausgegebenen Teilkorrektur ersetzt. Lautet der Linktext z.B. „Erstgutachten“, so wird er für die zweite Teilkorrektur automatisch in „Zweitgutachten“ umgewandelt.

Variablen mit Teilkorrekturnr.

Alternativ zur Verwendung innerhalb einer `$Teilkorrekturen`-Schleife können Sie die vier eben genannten Feld-Variablen übrigens auch (außerhalb einer solchen Schleife) mit Nummernsuffixen ausstatten, z.B.: `$FeldA1BKHREF1` erzeugt einen Link zur Beikorrektur (zur Dateieinsendung aus Eingabefeld 1 der ersten Teilaufgabe) der ersten Teilkorrektur, auf die zweite Teilkorrektur verlinkt entsprechend `$FeldA1BKHREF2` etc. Auch der Variable `$Korrektor` können Sie eine Teilkorrekturnummer anhängen, siehe nächsten Abschnitt.

Solche Variablen sind aber nur sinnvoll in Kontexten, in denen die Teilkorrekturanzahl feststeht. Selbst bei Prüfungen mit normalerweise zwei Prüfern könnte im Einzelfall ein Drittgutachter hinzugezogen werden, daher ist die Teilkorrekturanzahl in der Regel variabel – und dann ist die Verwendung einer `$Teilkorrekturen`-Scheife vorzuziehen, da sie immer genauso viele Ausgaben erzeugt wie tatsächlich im konkreten Einzelfall Teilkorrekturen vorliegen.

Hinweise zu `$Feld...ABKLNK...`-Variablen:

- Wenn hier eine Teilkorrekturnummer angehängt werden soll *und* ein Linktext angegeben werden soll, so ist die Teilkorrekturnummer direkt an `ABKLNK` anzuhängen, also vor dem `{Linktext}` einzufügen.
- Die Ersetzung des Teilwortes „Erst“ / „erst“ im Linktext der `ABKLNK`-Variablen erfolgt in diesem Fall nicht: Da die Teilkorrekturnummer fest angegeben wird und nicht variabel ist, wird auch der Linktext als endgültig angenommen und nicht angepasst.
- Beispiel: Eine Variable der Art `$FeldA3ABKLNK2{Erster Versuch}` wird einen Link auf den Dateianhang des zweiten Korrektors ausgeben und ihn dennoch mit „Erster Versuch“ (und nicht etwa „Zweiter Versuch“) beschriften.

Korrektornamen bei Mehrfachkorrektur

Die Namen der einzelnen Korrektoren, die an der Korrektur mitgewirkt haben (die jeweils eine Teilkorrektur bearbeitet haben), könnten prinzipiell über eine `$Teilkorrekturen`-Schleife (s.o.) und darin die `$Korrektor`-Variable ausgegeben werden. Es gibt dazu aber noch folgende Alternativen zur „direkten“ Verwendung ohne Schleife:

Variable	Bedeutung
`$Korrektoren`	Erzeugt eine kommaseparierte Aufzählung der Namen aller Korrektoren, beginnend mit dem Erstkorrektor.
`$Korrektor`	Die bereits bekannte Variable `$Korrektor` gibt im Falle einer Korrektur mit mehreren Teilkorrekturen den Namen des ersten Korrektors (Korrektors der ersten Teilkorrektur) aus, genau wie `$Korrektor1`, s.u.. (Innerhalb einer `$Teilkorrekturen`-Schleife, siehe oben, gibt `$Korrektor` den Namen des Korrektors der jeweiligen Teilkorrektur aus.)
`$Korrektor2`	Gibt den Namen des Korrektors der zweiten Teilkorrektur aus. Die Nr. 2 in diesem Fall ist nur exemplarisch zu verstehen und kann eine ganze Zahl ab 1 bis hin zur Anzahl der Teilkorrekturen sein. (Diese Variablen sind allerdings normalerweise nur in Kontexten sinnvoll, in denen die Anzahl der Teilkorrekturen immer feststeht, siehe auch obige Anmerkungen Variablen mit angehängter Teilkorrekturnummer.)

Das oben zu Beginn des Abschnitts »Korrekturvorlage erstellen« stehende Beispiel enthält im Seitenkopf z.B. die Zeile:

```
<p><strong>Korrektor:</strong> $Korrektor</p>
```

Bei Mehrfachkorrektur kann diese Zeile am einfachsten wie folgt umgestellt werden:

```
<p><strong>Korrektoren:</strong> $Korrektoren</p>
```

Das erzeugt z.B. eine Ausgabe der Art:

> ***Korrektoren:*** *Jane Doe, Max Mustermann*

Eine Teilkorrekturen-Schleife benötigen Sie nur, wenn Sie die Korrektoren anders formatiert ausgeben möchten, z.B. als Tabelle der Art:

```
<table>
<tbody>
$Teilkorrekturen
 <tr>
  <th>$TeilkorrekturNrErster Korrektor</th>
  <td>$Korrektor</td>
 </tr>
$/Teilkorrekturen
</tbody>
</table>
```

Ausgeben der Bewertungen (Teilwertungen & Gesamtwertung)

Die bekannten Variablen `$KorrekturPunkte` und `$KorrekturNote` geben im globalen Kontext (*außerhalb* einer `$Teilkorrekturen`-Schleife) die Gesamtpunktzahl bzw. Gesamtnote aus, *innerhalb* einer `$Teilkorrekturen`-Schleife dagegen, wie oben schon beschrieben, die Bewertung der jeweiligen Teilkorrektur.

Damit könnte die Bewertungsausgabe am Fuße einer Korrekturseitenvorlage z.B. für eine mit Punkten und Note bewertete Prüfung mit zwei (oder mehr) Prüfern wie folgt aussehen:

```
<h3>Teilwertungen:</h3>
<p>$Teilkorrekturen
$TeilkorrekturNr.) Wertung des $TeilkorrekturNrErstprüfers ($Korrektor):
$KorrekturPunkte von $MaximalPunkte Punkten, Note: $KorrekturNote
<br>
$/Teilkorrekturen
</p>
<h3>Erreichte Punktzahl: $KorrekturPunkte von $MaximalPunkte</h3>
<h3>Note: $KorrekturNote</h3>
```

Dateieinsendungen / Beikorrekturen bei Mehrfachkorrektur

Die einfachste Möglichkeit zum Einfügen einer Dateieinsendung in eine Korrekturseite ist – bei Mehrfachkorrektur genau wie bei Einfachkorrektur – die simple Aufnahme der Variablen `$FeldA1` für eine Dateieinsendung (hier im ersten Eingabefeld der ersten Teilaufgabe (A)). Ohne weiteres Variablen-Suffix gilt hier ein Automatikmodus, der im Falle von Mehrfachkorrektur automatisch wie folgt arbeitet:

- Ansicht für den *Korrektor* einer Teilkorrektur
  - Ein Korrektor sieht an dieser Stelle zum Ersten einen Downloadlink für*seine* Beikorrektur. Genauer: Hat er noch nie eine Beikorrektur gespeichert, kann er über diesen Link die Originaleinsendung herunterladen. Hat er bereits eine bearbeitete Version davon hochgeladen, wird im Anschluss der Link diese bearbeitete Kopie wieder herunterladen.
  - Zum Zweiten bekommt der Korrektor hier auch eine Uploadmöglichkeit geboten, um die heruntergeladene und bearbeitete Beikorrektur wieder hochladen zu können.
  - Prinzipiell verhält sich die Korrektorensicht hier also praktisch genauso wie bei Einfachkorrektur, nur dass eben mehrere Korrektoren parallel arbeiten können und jeder seine eigene Beikorrektur (von ihm bearbeitete Kopie der Dateieinsendung) einstellen kann.
  - Die Beikorrekturen der anderen Korrektoren kann der Korrektor im Korrekturmodus selbst nicht sehen – wohl aber, wenn er anschließend eine Korrekturvorschau aufruft: Dann sieht er im Allgemeinen die Korrektur fast genauso, wie der Student sie später sieht, samt aller Teilkorrekturen. Ausnahme: In den Kursparametern kann ein Exklusivzugriff aktiviert werden, der bewirkt, dass Korrektoren die Wertungen und Beikorrekturen der anderen Korrektoren nicht sehen dürfen (um unabhängig von den anderen Korrektoren zu beurteilen).
- Ansicht für einen *Studenten*
  - Nach Korrekturfreigabe sieht der Student an Stelle dieser `$FeldA1`-Variablen entweder einen einzigen Link oder eine Aufzählung von Links zu den einzelnen Beikorrekturen:
    - Sofern mindestens ein Korrektor eine modifizierte Kopie der Einsendung als Beikorrektur hochgeladen hat, wird dem Studenten eine Liste von Links angezeigt, mit einem Link pro Teilkorrektur. Gibt es also z.B. zwei Teilkorrekturen (eines Erst- und eines Zweitkorrektors) zur Korrektur, so werden hier *zwei* Links ausgegeben, einmal zur vom Erstkorrektor ggf. modifizierten/annotierten Einsendung und einmal zur vom Zweitkorrektor bearbeiteten Einsendung.
      - Diese Links bekommen eine Standardbeschriftung der Art „Erste Korrektur zu *Dateiname*“, sofern der Korrektor eine modifizierte Datei hochgeladen hat, sich die verlinkte Datei also (wahrscheinlich) von der Originaldatei unterscheidet, ansonsten der Art „Erste Kopie von *Dateiname*“.

        - Die Links werden (derzeit) durch Komma und Zeilenumbruch getrennt aufgezählt.
      - Gibt es zwar mehrere Teilkorrekturen, aber kein einziger Korrektor hat „seine“ Kopie der Dateieinsendung bearbeitet (annotiert), d.h. jeder Korrektor hat die Dateieinsendung unverändert belassen, dann wird dem Studenten genau ein Link zu seiner unveränderten Einsendung angeboten, beschriftet einfach nur mit dem Dateinamen der eingesandten Datei. (Es wird also explizit *nicht* eine Liste von mehreren Links der Art „Erste/Zweite/… Kopie von Dateiname“ erzeugt, die alle auf dieselbe unveränderte Einsendung verweisen würden.)
- Ansicht für einen *Betreuer*
  - Ein Betreuer bekommt i.W. dieselbe Ansicht zu sehen wie die Studenten auch – nur auch schon vor der Korrekturfreigabe.

Wenn Ihnen diese Standard-Ausgaben (mit Standard-Linkbeschriftung und einfachem Zeilenumbruch zwischen den Links) nicht zusagen, haben Sie aber auch alternative Möglichkeiten, den Feldinhalt auszugeben (wieder mit Hilfe einer `$Teilkorrekturen`-Schleife, s.o).

Möchten Sie z.B. eine Aufzählungsliste (`<ul>`) mit den Links zu den Beikorrekturen ausgeben, können Sie das wie folgt tun:

```
<ul>
$Teilkorrekturen
<li>$FeldA1BKHREF</li>
$/Teilkorrekturen
</ul>
$FeldA1BKUPLF
```

Dieses Beispiel verwendet eine `$Teilkorrekturen`-Schleife und gibt darin Listenelemente mit Beikorrektur-Links aus. Diese Links verwenden dieselbe Standardbeschriftung („Erste Korrektur zu Dateiname“ bzw. „Erste Kopie von Dateiname“) wie im Automatikmodus.

Beachten Sie, dass – anders als im Automatikmodus – hier *nur* die Links ausgegeben werden – und das auch nur in der Studenten-/Betreueransicht: Ein Korrektor sieht diese Link-Liste *nicht* (außer in der Korrekturvorschau, aber nicht in seiner normalen Korrekturansicht)!

Damit ein Korrektor wenigstens „seine“ Kopie der Dateieinsendung herunterladen und vor allem auch eine bearbeitete Kopie davon wieder hochladen kann, ist in diesem Fall ein entsprechendes Upload-Feld über die Variable `$FeldA1BKUPLF` hinzuzufügen. (Im Automatikmodus wird auch diese Uploadmöglichkeit automatisch bereitgestellt, in dieser selbst formatierten Ausgabe müssen Sie auch das selbst erledigen.)

Diese BKUPLF-Variable (steht für „Beikorrektur-Upload-Feld“) erzeugt besagte Down- und Uploadmöglichkeit für den Korrektor während der Korrekturphase. Für andere Nutzer (Studenten/Betreuer) bzw. in der Korrekturvorschau für den Korrektor ist diese Variable wiederum unsichtbar.

Als letztes Beispiel nehmen wir noch an, dass Sie auch die Linkbeschriftung für die Beikorrektur-Links selbst festlegen wollen. Dann können Sie auch die `$FeldA1BKHREF`-Variable nicht benutzen, da diese eine Standardbeschriftung generiert, sondern schreiben den Link selbst, z.B. wie folgt:

```
<ul>
$Teilkorrekturen
<li><a href="$FeldA1BKURL" target="_blank">Einsendung, ggf. mit Anmerkungen des
$TeilkorrekturNrersten Prüfers</a></li>
$/Teilkorrekturen
</ul>
$FeldA1BKUPLF
```

Dieses Beispiel erzeugt eine Link-Liste, die in der fertigen Korrektur dann etwa so aussieht:

- Einsendung, ggf. mit Anmerkungen des ersten Prüfers
- Einsendung, ggf. mit Anmerkungen des zweiten Prüfers

Angehängte Beikorrekturen bei Mehrfachkorrektur

Im Abschnitt Dateianhangs-Upload für Korrektoren wurde bereits auf die Möglichkeit eingegangen, ein spezielles Eingabefeld (in der Aufgabenseite leer und versteckt) einzuführen, in dem Studenten nichts einsenden können, zu dem ein Korrektor aber eine sog. angehängte Beikorrektur speichern kann. Bei *Mehrfachkorrektur* gilt auch dies für jede Teilkorrektur, d.h. jeder der Korrektoren kann seinen eigenen Dateianhang zur Korrektur hinzufügen.

Sie können auch bei Mehrfachkorrektur – genau wie bei Einfachkorrektur – einfach die bereits in oben genanntem Abschnitt eingeführte Variable `$FeldA1ABKUPL{Beschriftung}` verwenden. Die Auswirkung ist ähnlich wie der oben beschriebene Automatikmodus zu Dateieinsendungen:

- Jeder Korrektor bekommt eine Uploadmöglichkeit und einen Link, mit dem er „seine“ zuletzt hochgeladene Datei auch wieder herunterladen/einsehen kann,
- und dem Studenten wird in der Korrekturseite eine Liste von Links zu allen vorliegenden Dateianhängen angezeigt.

Auch hier gibt es alternativ die Möglichkeit, über andere Variablen die Darstellung feiner zu beeinflussen. Die folgende Tabelle zeigt zunächst alle „ABK-Variablen“ im Überblick:

Variable	Bedeutung
`$FeldA2ABKUPL`	Erzeugt in der Korrektorsicht eine Uploadmöglichkeit für einen Dateianhang und einen Downloadlink, mit dem der Korrektor seinen ggf. schon hochgeladenen Anhang auch wieder herunterladen/einsehen kann. In der Studentensicht (und Korrekturvorschau f. Betreuer oder Korrektor) wird eine Liste von Downloadlinks zu allen vorliegenden Dateianhängen (der verschiedenen Korrektoren) erzeugt. Hat kein einziger Korrektor eine Datei hochgeladen, wird auch kein einziger Downloadlink ausgegeben.
`$FeldA2ABKUPL{Downloadlink-Text}`	Dieselbe Funktionalität, legt aber zusätzlich eine Link-Beschriftung für den Downloadlink (in der Studentensicht bzw. Vorschau) fest.
`$FeldA2ABKLNK`	Innerhalb einer `$Teilkorrekturen`-Schleife wird ein Link zu dem Dateianhang des Korrektors der jeweiligen Teilkorrektur ausgegeben – sofern vorhanden. Außerhalb einer `$Teilkorrekturen`-Schleife erzeugt diese Variable eine Linkliste zu allen vorliegenden Beikorrekturen, genau wie sie auch von `$FeldA1ABKUPL` in der Studentensicht ausgegeben wird – nur ohne die Uploadmöglichkeit. Außerdem kann an die Variable eine Teilkorrekturnummer angehängt werden, um gezielt einen Link zur Beikorrektur zur jeweiligen Teilkorrektur zu erzeugen (sofern Upload vorhanden), siehe Variablen mit Teilkorrekturnr..
`$FeldA2ABKLNK{Downloadlink-Text}`	Dasselbe, aber mit benutzerdefinierter Beschriftung für den Downloadlink. Falls in diesem Linktext das Teilwort „Erst“ oder „erst“ vorkommt, wird es bei Einsatz innerhalb einer `$Teilkorrekturen`-Schleife entsprechend der jeweiligen Teilkorrekturnummer ersetzt, z.B. durch „Zweit“ oder „Dritt“.
`$FeldA2ABKUPLF`	Erzeugt explizit eine Uploadmöglichkeit für den Korrektor. Wird nur benötigt, wenn *nicht* die `$FeldA2ABKUPL`-Variable verwendet wurde, sondern über andere Variablen bislang explizit *nur* Downloadmöglichkeiten für Dateianhänge geschaffen wurden. Dann ist hierüber für Korrektoren auch eine Uploadmöglichkeit zu schaffen. (Von der `$FeldA1BKUPLF`-Variablen unterscheidet sich diese Variable durch Optimierung für den Fall angehängter Beikorrekturen: Ein Upload ist auch möglich, wenn keine Dateieinsendung vorliegt. Falls zum Eingabefeld ein vom Korrektor auszufüllendes PDF-Formular hinterlegt wurde (s.o.), wird dem Korrektor auch entsprechend ein Download dieses Formulars ermöglicht.)

Um z.B. innerhalb einer `$Teilkorrekturen`-Schleife auch Links zu angehängten Beikorrekturen auszugeben, können Sie die Variable `$FeldA1ABKLNK` verwenden.

Nehmen wir beispielhaft eine Aufgabe an, in der ein Student in Feld A1 eine Datei hochladen soll, und in der ein verstecktes Feld A2 vorgesehen ist, in dem der Student nichts einsenden kann, aber der Korrektor einen weiteren Dateianhang, z.B. ein Gutachten, hochladen kann. Dann könnten Sie z.B. in Ihrer Korrekturseite (statt z.B. eines Absatzes mit einer `$FeldA1`-Variablen, die später zur Liste aller annotierten Beikorrekturen zur Dateieinsendung wird, sowie eines darauf folgenden Absatzes mit einer `$FeldA2ABKUPL`-Variablen, die später in der Korrekturfassung zu einer Liste aller angehängten Gutachten ersetzt wird) eine Tabelle der folgenden Art bauen:

```
<table>
 <thead>
  <tr>
   <th>Korrektor</th>
   <th>Ggf. korrigierte Dateieinsendung</th>
   <th>Ggf. Gutachten</th>
  </tr>
 </thead>
 <tbody>
 $Teilkorrekturen
  <tr>
   <th>$Korrektor</th>
   <td>$FeldA1BKHREF</td>
   <td>$FeldA2ABKLNK{Erstgutachten}</td>
  </tr>
 $/Teilkorrekturen
 </tbody>
</table>
$FeldA1BKUPLF
$FeldA2ABKUPLF
```

Da bei Verzicht auf die `ABKUPL`-Variable, die Download- und Uploadmöglichkeiten kombiniert anbietet, die Tabelle ausschließlich Downloadlinks enthält (und diese auch nur in der späteren Korrekturansicht für Studenten bzw. Korrekturvorschau für Betreuer und Korrekturen funktionieren, nicht aber in der normalen Korrektorsicht), ist in diesem Fall die `$FeldA2ABKUPLF`-Variable nicht zu vergessen (in obigem Beispiel unterhalb der Tabelle), damit die Korrektoren überhaupt die Möglichkeit bekommen, Dateien hochzuladen.

Ein Hinweis zum Abschluss:
Vielleicht ist Ihnen der Unterschied aufgefallen, wie Sie selbst beschriftete Downloadlinks zu normalen Beikorrekturen (zu Dateieinsendungen) erzeugen (vgl. vorherigen Abschnitt) und wie Sie entsprechend selbst beschriftete Downloadlinks zu *angehängten* Beikorrekturen erzeugen: Im ersten Fall schreiben Sie selbst ein `a`-Tag und verwenden `$FeldA1BKURL` als Wert fürs `href`-Attribut, im zweiten Fall verwenden Sie die `$FeldA2ABKLNK{…}`-Variable und geben den Linktext in den geschweiften Klammern an. Die erstere Methode für ist die flexiblere, sie bietet Ihnen noch mehr (HTML-)Gestaltungsspielraum (wie z.B. Bilder im Link, CSS etc.), aber sie erzeugt eben *immer* einen Link. Bei „normalen“ Beikorrekturen ist das OK und gewollt, der Link ist immer definiert: Liegt eine Beikorrektur (vom Korrektor bearbeitete Kopie der Einsendung) vor, kann diese über den Link heruntergeladen werden, andernfalls verweist der Link auf die Original-Dateieinsendung des Studenten. Bei den *angehängten* Beikorrekturen gibt es dagegen keine Original-Einsendung. Hat hier ein Korrektor *keine* Datei angehängt, soll auch gar kein Link erzeugt werden. Daher wird hier insbesondere keine `$FeldA1ABKURL`-Variable angeboten, weil eben nicht in jedem Fall überhaupt eine Datei existiert, auf die mit einem solchen URL verlinkt werden könnte. Vielmehr ist stets der gesamte Link über eine Variable zu erzeugen, und das Übungssystem entscheidet darüber, ob es überhaupt einen Link ausgibt (nämlich nur, sofern eine Datei vorliegt) oder nicht.

Korrekturvorlagen bei Randomisierung

Die Korrekturvorlage enthält in der Regel Abschnitte zu sämtlichen Eingabefeldern der gesamten Aufgabe. Falls Sie aber die Fragen-Randomisierung <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#fragenrandom> verwenden sollten und dazu bereits ein Aufgaben-Formular mit Fragen-Randomisierung erstellt haben, dann sollte auch die Korrekturseite am besten nur die Eingaben zu genau den Eingabefeldern wiedergeben, die der Student im Aufgabenformular überhaupt bearbeiten / ausfüllen konnte. Oder anders gesagt: Die Korrekturseite sollte idealerweise *nicht* zu den Fragen, die der Student gar nicht sehen und beantworten konnte, lauter Leereinsendungen aufführen.

Dieselbe Fragen-Struktur mit Random-Variablen sollte sich daher auch in der Korrekturvorlage wiederfinden. Dabei gilt: Alle Teile der Korrekturvorlage, die zwischen `$Random1` und `$/Random1` stehen, werden nur denjenigen Einsendern angezeigt, denen im Aufgabenformular genau die Frage zwischen den gleichnamigen Variablen präsentiert wurde. Teilnehmer, die eine andere der Fragen ausgewählt bekommen hatten, sehen dann auch diesen Teil der Korrekturseite nicht.

Der wesentliche Ausschnitt der Korrekturvorlage zum ersten Beispielformular aus Abschnitt Formular mit Fragen-Randomisierung (ohne `$Randomize`-Variable) könnte wie folgt aussehen:

```
$Random1
<h2>Überschrift erste Frage</h2>
<pre>$FeldA1P</pre>
$/Random1

$Random2
<h2>Überschrift zweite Frage</h2>
<pre>$FeldB1P</pre>
$/Random2
```

Wichtig ist, dass die Menge der Random-Variablen in der Korrekturseite *zu 100% mit der in der Aufgabenseite übereinstimmt*, also zu jeder `$Randomx`-Variable der Aufgabenseite auch eine gleichlautende `$Randomx`-Variable in der Korrekturseite existiert und umgekehrt. Auch die `$Randomize`-Variable, sofern in der Aufgabenseite vorhanden, ist in der Korrekturseite identisch anzubringen. (Bei Abweichungen würde nicht mehr dieselbe Randomisierung, also Fragenauswahl, berechnet, was hier zu unerwartetem Verhalten wie insbesondere dem Fehlen bestimmter Blöcke führen könnte.)

Nur in einem Punkt darf die Randomize-Variable der Korrekturvorlage von der der Aufgabenseite abweichen: Falls Sie in der Aufgabenseite eine **zufällige Fragenreihenfolge** nutzen (`$Randomize(x, Shuffle)` für eine Fragenanzahl x), haben Sie in der Korrekturvorlage die Wahl, ob Sie das `Shuffle`-Argument wieder angeben oder nicht. *Wenn* Sie es auch in der Korrekturseite wieder angeben, so werden die Eingaben zu den Fragen in der konkreten Korrektur eines Studenten wieder in derselben Reihenfolge stehen, wie ihm die Fragen gestellt wurden. Das erhöht also die Übersicht für den Studenten (und ist die empfohlene Einstellung), da die Korrekturseite genau „zu seiner Aufgabenseite passt". Dafür muss sich ein Korrektor darauf einstellen, dass er bei jedem Studenten die Antworten zu den einzelnen Fragen in einer anderen Reihenfolge sieht (nämlich der individuellen Reihenfolge, in der die Fragen dem jeweiligen Studenten gestellt wurden). Lassen Sie das `Shuffle`-Argument in der Ranomize-Variable dagegen weg, so werden die Antworten in der Korrektur jeweils (für den Korrektor, aber auch in der finalen Ansicht für den Studenten) in der „Standardreihenfolge" stehen, also in der Reihenfolge, in der Sie die Fragen in der Korrekturseitenvorlage angeordnet haben. Das ist vielleicht angenehmer für den

Korrektor, wenn er die Eingaben jedes Studenten in derselben Reihenfolge sieht, dafür müssen die Studenten sich später bei Einsicht in die Korrektur umorientieren, da sie Korrekturen zu ihren Antworten dann in einer anderen Reihenfolge sehen, als ihnen die Fragen gestellt wurden.

Wie schon im Grundlagen-Abschnitt Hinweise zur Fragenauswahl bei wiederholten Zugriffen <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#randomfragenauswahl> erläutert wird, kann das Übungssystem bei aktiver Randomisierung berücksichtigen, welche Fragen von einem Studenten bereits bearbeitet wurden. Das stellt sicher, dass einem Studenten immer die schon von ihm bearbeiteten Fragen angezeigt werden, selbst wenn z.B. das Aufgabenformular nachträglich (nach Bearbeitungsbeginn und ersten Einsendungen) um weitere Fragen ergänzt werden sollte. Auch für diese Random-Abschnitte in der Korrekturseite gilt dasselbe: Falls zu einem der Felder, die innerhalb eines Random-Blocks per Variable `$FeldXy` (oder anderen Variablen mit Feldbezug wie `$IfExistsXy`) referenziert werden, schon eine Einsendung vorliegen sollte, wird auch dieser Random-Block in der Korrekturseite ausgegeben, selbst wenn zwischenzeitliche Änderungen der Aufgabe eigentlich eine andere zufällige Auswahl bewirken würden.
Sie sollten daher auf jeden Fall sicherstellen, dass Sie keine Random-Blöcke verwenden, die *keine* Feldreferenzen enthalten!

Speziell für Korrekturseiten gelten auch Vorkorrekturvariablen mit Teilaufgabenbezug (also z.B. `$VorkorrekturA`) als „Feldbezug", auch wenn sie sich streng genommen nur auf eine Teilaufgabe und nicht auf ein spezifisches Feld einer Teilaufgabe beziehen: Random-Blöcke, die eine solche Vorkorrektur-Variable enthalten, werden immer (auch wenn sich nach Aufgabenänderungen etwas an der Randomisierung geändert haben sollte) ausgegeben, wenn zu mindestens einem Feld der zugehörigen Teilaufgabe eine Einsendung des Studenten vorliegt.

Nehmen wir z.B. an, sie haben eine Aufgabenseite mit 5 Fragen erstellt, von denen drei zufällig ausgewählt werden sollen und diese auch noch in zufälliger Reihenfolge gestellt wurden, und zu allen fünf Teilaufgaben sei ein Vorkorrekturmodul eingerichtet, dessen Ausgabe jeweils in die Korrekturseite übernommen werden soll. Dann können Sie das wie folgt erledigen:

```
$Randomize(3, Shuffle)
$Random1 $VorkorrekturA $/Random1
$Random2 $VorkorrekturB $/Random2
$Random3 $VorkorrekturC $/Random3
$Random4 $VorkorrekturD $/Random4
$Random5 $VorkorrekturE $/Random5
```

Auf diese Weise (anders als bei Angabe einfach nur der Variablen `$Vorkorrektur`, die alle vorliegenden Vorkorrekturen in der Reihenfolge der Teilaufgabennummern ausgibt) würden die Vorkorrekturen dann auch in der „korrekten" Reihenfolge ausgegeben, in der dem Studenten die Fragen gestellt worden waren.

Obige Lösung bezieht sich explizit nur auf die Einbindung von **Vorkorrekturen**, die also jeweils schon zum Zeitpunkt der Einsendung erzeugt wurden und vermutlich auch als Sofortfeedback in der Quittungsseite eingebunden wurden.

Anders sieht es aus, wenn Sie ein Korrekturmodul (ausgeführt zum Zeitpunkt des Heft-Schließens) verwenden. Das erzeugt eine zusammenhängende Korrekturausgabe über alle vorliegenden Einsendungen, also teilaufgabenübergreifend. Diese Autokorrektur wird *immer*, also auch bei Randomisierung, per Variable `$Korrektur` (ggf. mit Suffixen) eingebunden, siehe Vorkorrekturen und Autokorrekturen.

Für solche Autokorrekturen gilt bei Randomisierung:

- Wenn Sie eine zufällige Fragenauswahl nutzen, also ein Student nur eine Teilmenge der Teilaufgaben bearbeitet, hängt es vom Korrekturmodul ab, ob es die unbearbeiteten Teilaufgaben ignoriert oder wie es sie in seiner Korrektur darstellt. Der interne `Aufgabenbewerter` erzeugt nur Korrekturausgaben zu bearbeiteten Teilaufgaben.
- Wenn Sie zufällige Fragenreihenfolge („Shuffle“) nutzen, wirkt sich das standardmäßig *nicht* auf die Reihenfolge der Ausgaben in der Autokorrektur eines Korrekturmoduls aus. Ein Korrekturmodul „weiß nicht“, in welcher Reihenfolge der Student die Teilaufgaben bearbeitet hat und erzeugt seine Ausgaben daher normalerweise in einer festen Reihenfolge. Dazu gibt es aber seit Juli 2023 eine Lösung, siehe Selektionssubmit mit Speicherung der Selektion. Wenn Sie diese verwenden, kann das Korrekturmodul die gespeicherte Information über die Zufallsreihenfolge nutzen, um intern seine Ausgaben entsprechend zu sortieren.

Falls Sie den internen »Teilaufgabenbewerter« als Vorkorrekturmodul verwenden und Sie immer *alle* Fragen (also keine zufällige Auswahl der Fragen) in zufälliger Reihenfolge stellen möchten („Shuffle Only“), dann haben Sie prinzipiell *zwei* Möglichkeiten:

1. Entweder Sie zerlegen die Aufgabenseite in einzelne Teilaufgaben (eine Teilaufgabe pro Random-Block, wie für Randomisierung generell empfohlen). Dann wird für jede Teilaufgabe / Frage eine einzelne Vorkorrektur erzeugt, und Sie können wie oben beispielhaft gezeigt diese Vorkorrekturen in derselben „Shuffle-Reihenfolge“ in der Korrekturseite ausgeben, wie dem jeweiligen Teilnehmer auch in der Aufgabenseite die Fragen gestellt wurden. Was dann allerdings *nicht* möglich ist, ist die Funktion des Bewerters, eine *Gesamtwertung* über alle Fragen zu erstellen, weil er ja für jede Teilaufgabe/Frage einzeln ausgeführt wird.
2. Wenn Sie Wert auf eine Gesamtwertung legen, müssen dazu alle Fragen zu einer einzigen Teilaufgaben zusammengefasst werden. (So lange Sie keine Teilauswahl der Fragen anbieten, sondern jeder Teilnehmer stets *alle* Fragen einsendet, und nur die Präsentationsreihenfolge randomisiert wird, ist das problemlos möglich.) Dann wird auch nur eine einzige Vorkorrektur zur einzigen Teilaufgabe erstellt, die aus den Korrekturen zu allen einzelnen Fragen besteht und auch eine Gesamtwertung über alle Fragen umfassen kann. Dann allerdings geht, wie bei der Autokorrektur auch, die„Shuffle-Reihenfolge“ verloren, d.h. die Teilkorrekturen sind immer in der festen Reihenfolge der Fragen in Ihrer Aufgabenseiten-Vorlage sortiert und nicht in der individuell randomisierten.

Testen von Korrekturen

Während Sie von Aufgabenseiten jederzeit eine Vorschau öffnen und Quittungsseiten im Rahmen von Probe-Einsendungen z.B. aus dieser Vorschau heraus testen können, muss zum Testen der Korrekturseiten einmal ein Korrekturprozess begonnen werden:

- Einsendung mit mindestens einem Teststudenten. Es kann sinnvoll sein, mehrere Teststudenten anzulegen und mit diesen verschiedene Eingaben zur selben Aufgabe zu speichern, um den Prozess nicht so oft wiederholen zu müssen, sondern verschiedene Testfälle parallel abarbeiten zu können.
- Schließen der Hefte (in der Einsendungs- und Korrekturübersicht), wobei die Korrekturseiten zu den einzelnen Einsendungen erzeugt werden (siehe Szenario zu Standard-Kursressourcen).
- Einsehen der erzeugten Korrekturseiten in der Einsendungs- und Korrekturübersicht.

Über das reine Erzeugen und Einsehen der Korrekturseiten hinaus sollte bei handbewerteten Aufgaben auch der Korrekturprozess selbst getestet werden – vor allem bei In-Browser-Korrektur:

- Zuteilen der erzeugten Korrekturen an einen Korrektor.
- Einloggen mit diesem Korrektoraccount, einsehen und Bearbeiten der Korrekturen. Punkte vergeben und Korrekturstatus auf »korrigiert« setzen.
- Einsehen und freigeben der Korrekturen als Betreuer
- Einsicht der fertigen Korrektur als Teststudent.

Neuerzeugung nach Änderungen an der Seitenvorlage oder Korrekturmoduleinstellungen

Wenn Sie im Rahmen eines Tests einer Korrektur einen Fehler bemerkt und eine Änderung an der Korrekturseitenvorlage (`korrektur1.1.html`) oder an den Einstellungen eines Korrektur-/Bewertermoduls vorgenommen haben und nun die Korrekturseite zu einer bereits vorliegenden studentischen Einsendung neu erzeugen möchten, so muss das Heft-Schließen (siehe Abschnitt zu Standard-Kursressourcen) wiederholt werden.

Speziell für Neuerzeugung der Korrekturen aller Studenten für eine einzelne Aufgabe finden Sie auf der Betreuerstartseite unter Rubrik »Einsendungen und Korrekturen« die Funktion »Erneute Hefteinsendung für einzelne Aufgaben«.

Um das Heft-Schließen für alle Aufgaben eines Aufgabenhefts aller oder bestimmter Studenten zu wiederholen, benutzen Sie die Funktionen zum »Heft-Öffnen« in der Einsendungs- und Korrekturübersicht oder »Hefteinsendung löschen« in der Übersicht aller Korrekturen eines Korrektors sowie anschließend die »Hefte schließen«-Funktion zum Aufgabenheft in der Einsendungs- und Korrekturübersicht.

Korrekturhinweis- und Musterlösungsseite erstellen

Abschließend können Sie eine Musterlösungsseite für Studenten und bei manuell zu bewertenden Aufgaben eine Seite mit Hinweisen für die Korrekturkräfte erstellen (siehe Standard-Kursressourcen).

Diese beiden Seiten sind optional, wobei zumindest für handbewertete Aufgaben immer eine Musterlösung erzeugt werden sollte. Wie für die anderen Standard-Kursressourcen auch, können Sie sich hier eine Vorlagedatei erzeugen lassen und diese anschließend nachbearbeiten. Die HTML-Seiten können Sie weitestgehend frei bearbeiten, um darin ihre Texte unterzubringen. Dabei stehen Ihnen die oben vorgestellten allgemeinen Variablen zur Verfügung.

Falls Sie in Musterlösungen weitere Kursressourcen referenzieren wollen (einbinden von Grafiken oder verlinken von anderen Dateien wie PDFs oder andere Musterdokumente), beachten Sie die Namenskonventionen zur Taktung <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#taktung> , um sicherzustellen, dass auf diese Musterdokumente nicht bereits vor der Freigabe der Musterlösung zugegriffen werden kann.

Musterlösungsseiten bei Randomisierung

Normalerweise enthalten Musterlösungsseiten einfach nur einen statischen Text (ggf. mit Bildern oder Links zu PDFs o.ä.).

Falls Sie die Fragen-Randomisierung <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#fragenrandom> verwenden, könnten Sie natürlich auch einfach eine globale Seite erstellen, welche die Lösungen zu allen möglichen Fragen für alle Studenten ausgibt. Möchten Sie aber, dass jeder Student nur die Lösungen zu genau den Fragen

angezeigt bekommt, die er beantworten musste, und ihm die Lösungen zu den Fragen, die ihm nie gestellt wurden, entsprechend vorenthalten werden sollen, dann übertragen Sie die Random-Fragenstruktur auch auf die Musterlösungsseite.

Im Prinzip funktioniert das genauso, wie im Abschnitt Korrekturvorlagen bei Fragen-Randomisierung beschrieben: Fügen Sie auch in die Musterlösungsseite dieselben `$Random`- und `$Randomize`-Variablen ein, wie sie schon im Aufgabenformular und der Korrekturvorlage auftreten, und schreiben Sie die Musterlösungstexte zu den entsprechenden Fragen zwischen die zugehörigen Random-Variablen. (Falls Sie in der Aufgabenseite eine zufällige Fragenreihenfolge nutzen, also eine `$Randomize(x, Shuffle)`-Variable eingefügt haben, gilt hier analog zur Korrekturseite: Angabe des `Shuffle`-Arguments auch in der Musterlösungsseite bewirkt, dass jeder Student die Random-Blöcke der Musterlösungsseite in „seiner individuellen Reihenfolge“ analog zu seinem Aufgabenformular sieht, während bei Weglassen des `Shuffle`-Arguments aus der `$Randomize`-Variable die Blöcke der Musterlösungsseite für Alle in konstanter Reihenfolge präsentiert würden. Die Übernahme des `Shuffle`-Arguments wird hier empfohlen.)

Eine Besonderheit sollte dabei aber beachtet werden: Wie schon in Hinweise zur Fragenauswahl bei wiederholten Zugriffen <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#randomfragenauswahl> beschrieben, kann das Übungssystem sicherstellen, dass einem Studenten immer dieselbe Frage wieder präsentiert wird, die er bereits bearbeitet hat, selbst falls der Aufgabenautor später noch weitere Fragen zur Aufgabe hinzugefügt haben sollte (was die automatische Auswahl normalerweise beeinflusst). Und wie schon am Ende von Abschnitt Korrekturvorlagen bei Fragen-Randomisierung beschrieben wird, sollte dazu jeder Random-Block immer eine Referenz auf mindestens ein Eingabefeld der jeweiligen Frage enthalten. Bei der Korrekturseitenvorlage ist das normalerweise immer der Fall, in Musterlösungstexten dagegen kommen normalerweise *keine* Feldreferenzen vor!

Ein Fallbeispiel: Angenommen, zum Bearbeitungsbeginn besteht eine Aufgaben aus genau zwei Fragen. Ein Student öffnet die Aufgabenseite und bekommt als zufällige Auswahl die erste Frage angezeigt und sendet zu dieser etwas ein. Später fügt der Aufgabenautor aber noch während des Bearbeitungszeitraums zwei weitere Fragen ein. Hätte der Student die Aufgabe noch nicht bearbeitet, würde ihm nun wahrscheinlich eine andere, z.B. die dritte Frage präsentiert, aber da von ihm schon Einsendungen zu Frage 1 existieren, wird ihm weiterhin die erste Frage präsentiert, so dass er sie bis zum Einsendeschluss noch nachbearbeiten kann. Auch in seiner Korrektur werden korrekt seine Eingaben zur ersten Frage aufgenommen. Aber würde er nun die Musterlösungseite aufrufen und enthielte diese nur einfachen Lösungstext zwischen den Random-Variablen, aber keine Feldreferenzen, so könnte das Übungssystem keine der vier Musterlösungen zur bearbeiteten Frage zuordnen und würde die Musterlösung zur dritten Frage – also nicht zur vom Studenten bearbeiteten Frage – anzeigen.

Um dieses Verhalten zu verhindern, fügen Sie irgendeine für den Studenten unsichtbare Feldreferenz in den Musterlösungstext ein, z.B. eine Variable der Form `$IfExistsA1()`. Diese legt fest, dass im Falle einer vorliegenden Einsendung des Studenten zu Feld A1 der Text zwischen den Klammern angezeigt werden soll, andernfalls gar kein Text. Da aber zwischen den Klammern auch kein Text steht, wird effektiv *immer* kein Text ausgegeben, womit die Variable für Studenten effektiv immer unsichtbar bleibt. Für den Randomisierer dagegen hat sie sehr wohl eine Auswirkung: Falls zum Feld A1 eine Einsendung existiert, wird in jedem Fall der Random-Block ausgewählt, in dem diese Variable steht.

Der wesentliche Ausschnitt aus einer Musterlösungsseite zum Beispiel aus Abschnitt Formular mit Fragen-Randomisierung könnte also wie folgt aufgebaut sein:

```
$Random1
$IfExistsA1()
<h2>Überschrift erste Frage</h2>
<p>ML-Text zur ersten Frage</p>
$/Random1

$Random2
$IfExistsB1()
<h2>Überschrift zweite Frage</h2>
<p>ML-Text zur zweiten Frage</p>
$/Random2
```

Eine zufällige Auswahl erfolgt im Übrigen „nur“ für Studenten. Falls sich dagegen ein Korrektor die Musterlösung (oder auch das Aufgabenformular) ansieht, so werden ihm stets *alle* Fragen bzw. Lösungen zu den Fragen angezeigt. Bei aktiviertem Embedding (wenn Ihre Musterlösungsseite bzw. Aufgabenseite die Marken `$EMBED` und `$/EMBED` enthalten) werden die einzelnen Blöcke dann als sog. Akkordeon präsentiert, so dass immer nur eine Frage bzw. Lösung gleichzeitig sichtbar ist, der Korrektor aber dazwischen wechseln kann. Wenn ein Korrektor zu einer gerade geöffneten Korrektur die Musterlösungsseite öffnet, wird ihm im Akkordeon aber gerade die „passende“ Lösung geöffnet, also der Teil zu jenau der Random-Sektion, die der Student bearbeitet hat, zu deren Feldern also eine Einsendung vorliegt. (Die zu öffnende Akkordeon-Sektion wird ebenfalls anhand der `$IfExistsA1()`-Variablen innerhalb der `$Random`-Blöcke identifiziert, d.h. es wird diejenige geöffnet, für die Einsendungen des Studenten, dessen Korrektur gerade bearbeitet wird, zu einem der damit referenzierten Felder vorliegen.)

Korrekturhinweise bei Randomisierung

Korrekturhinweise werden lediglich Korrektoren präsentiert, und für Korrektoren erfolgt nie eine zufällige Auswahl. Von daher ist ein Einfügen von Random-Variablen in die Korrekturhinweisseite „eigentlich“ nicht nötig.

Andererseits kann es aber doch sinnvoll sein, die Korrekturhinweise zu den einzelnen Fragen genauso durch `$Random`-Variablen zum umschließen wie schon die Musterlösungen zu den entsprechenden Fragen. (Auf das Einfügen der Feldreferenzen kann dabei verzichtet werden.) Der Grund: Wenn der Korrektor schon bei Ansicht des Aufgabenformulars und der Musterlösungsseite jeweils eine Akkordeon-Ansicht bekommt, in der er immer nur eine Frage bzw. Lösung sieht und zwischen den einzelnen Fragen und Lösungen umschalten kann, wirkt eine Korrekturhinweisseite, in der stets die Hinweise zu allen Fragen untereinander stehen, möglicherweise etwas inkonsistent. Durch das Einfügen der Random-Variablen aktivieren Sie hier eine analoge Akkordeon-Darstellung, in der der Korrektor die Hinweise zu den einzelnen Fragen durchblättern kann. (Das betrifft übrigens nur die Bildschirmdarstellung: Beim Ausdruck der Seite werden stets alle Hinweise bzw. Lösungen bzw. Fragen ausgegeben.)

Wenn Sie die `$Random`-Variablen verwenden und dem Korrektor somit eine Akkordeon-Ansicht bieten, ist es auch sinnvoll, analog zu den Musterlösungen (s.o.) `$IfExistsA1()`-Variablen zu den Feldern der jeweiligen Frage mit einzubinden. Diese versetzen das Online-Übungssystem in die Lage, einem Korrektor sofort die „passenden“ Korrekturhinweise anzuzeigen, wenn er zu einer bestimmten Korrektur die Korrekturhinweisseite öffnet: Das Übungssystem öffnet dann in der Akkordeon-Ansicht genau die Sektion zu derjenigen Frage, die der Student (dessen Korrektur gerade geöffnet ist) bearbeitet hat.

(Vor-)Korrektur-/Bewertermodule registrieren und konfigurieren

Soll ein Vorkorrekturmodul zu einer Teilaufgabe oder ein Korrektur-/Bewertermodul zur gesamten Aufgabe (vgl. (Vor-)Korrekturmodule <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#moduleoverview>) eingesetzt werden, ist ggf. zunächst ein solches Modul zu programmieren, in Betrieb zu nehmen und anzubinden. In der Maske der fortgeschrittenen Aufgabenerstellung können Sie jeweils ein Modul aus der Liste der angemeldeten Module auswählen oder den Namen eines noch nicht angemeldeten Moduls von Hand eintragen, wenn Sie davon ausgehen, dass es bis zum Go-Live sich noch anmelden wird.

Neben selbst programmierten Modulen stehen derzeit zwei *intern* vom Übungssystem angebotene Module zur Auswahl: »Aufgabenbewerter« (Korrektur-/Bewertermodul für eine ganze Aufgabe) und »Teilaufgabenbewerter« (Selbsttest-/Vorkorrekturmodul, daher mit Teilaufgabenbezug). Beide bilden den im Kapitel „Korrekturmodi“ bereits kurz beschriebenen Internen Aufgabenbewerter <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#aufgabenbewerter> für Standardaufgaben wie Multiple Choice etc.

Externe Module

Verwenden Sie selbst entwickelte Korrekturmodule, so ist folgendes zu beachten:

- Ein Vorkorrektur- oder Bewertermodul muss zunächst auf einem Server in Betrieb genommen werden und es muss sich, sobald es ausgeführt wird, beim Online-Übungssystem für den Kurs, in dem es verwendet werden soll (genauer: für jede Kursversion, d.h. jedes Semester) anmelden.
- Sobald sich ein Modul angemeldet hat, ist es in der fortgeschrittenen Aufgabenstellung in der Liste verfügbarer Module (`...`-Button) auswählbar.
- Sobald ein Modul ausgewählt wurde, wird ein `Eigenschaften...`-Button eingeblendet, mit dem sich ein Dialog zur Eingabe eines mehrzeiligen Strings mit Properties öffnen lässt.
- Der Modul-Entwickler bestimmt, ob und was in diesen Eigenschaften einzutragen ist. Dieser Property-Text dient dazu, das Modul auf die konkrete Aufgabe einzustellen, denn selten wird ein Modul spezifisch für eine ganz bestimmte Aufgabenstellung entwickelt. Ein Vorkorrekturmodul zum Beurteilen von Java-Programmen z.B. kann von Haus aus in der Lage sein, einen Java-Quelltext zu compilieren und zu testen, wird aber in der Regel aufgabenspezifisch mitgeteilt bekommen müssen, wie viele Tests es durchführen soll, mit welchen Eingaben es den Prüfling dabei testen soll und vor allem, in welchen Fällen die Antwort des Prüflings als korrekt im Sinne der Aufgabe zu werten ist. Für Bewertermodule können auch Informationen zur aufgabenspezifischen Punktevergabe in den Eigenschaften benötigt werden.
- Im Ausnahmefall kann einem Korrekturmodul noch ein Korrekturmoduladapter <https://online-uebungssystem.fernuni-hagen.de/download/Korrekturmoduladapter/Korrekturmoduladapter.html> vorgeschaltet werden. Auch das geschieht über den `Eigenschaften...`-Dialog.

Mehr Informationen finden Sie im Betreuer-Interface des Kurses unter »Entwicklung eigener Korrekturmodule«.

Konfiguration der internen Module

Im Normalfall wird empfohlen, Aufgaben, die mit dem internen Bewerter automatisch korrigiert werden sollen, mit dem Aufgabenerstellungsassistenten (AB) zu erstellen, der dann auch die gesamte Einrichtung des Vorkorrektur- bzw. Bewertermoduls übernimmt.

Falls Sie aber den internen Bewerter im Rahmen der fortgeschrittenen Aufgabenerstellung direkt konfigurieren (oder eine assistentengestützt erzeugte Aufgabenkonfiguration manuell nachbearbeiten) möchten, sei hier dessen Konfiguration beschrieben.

Zunächst ist zu entscheiden, ob die automatische Auswertung sofort bei Einsendung erfolgen soll (Selbsttest) oder erst nach Einsendeschluss (Einsendearbeit), wie schon im Teil I <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html> zur assistentengestützten Aufgabenerstellung beschrieben.

- Für **Auswertung nach Einsendeschluss** ist das Modul »Aufgabenbewerter« als Korrektur-/Bewertermodul für die Gesamtaufgabe auszuwählen, in der Quittungsvorlage ist lediglich der Eingang der Eingaben mit `$Feld…`-Variablen zu bestätigen (eine Auto-Korrektur liegt ja zum Einsendezeitpunkt noch nicht vor), und in die Korrekturvorlage ist die Variable `$Korrektur` einzufügen. Denken Sie außerdem daran, die *erreichbare Punktzahl* zur Aufgabe korrekt einzutragen (als Summe der in allen automatisch bewerteten Fragen erreichbaren Punktzahlen aus der Bewerterkonfiguration).
- Für **Sofortauswertung** ist das Modul »Teilaufgabenbewerter« als Vorkorrekturmodul für die jeweilige (meist einzige) Teilaufgabe auszuwählen, in die Quittungsvorlage ist die Variable `$Vorkorrektur` und in die Korrekturvorlage die Variable `$VorkorrekturA` (für die erste Teilaufgabe) einzufügen, wie oben beschrieben.
  - Vorkorrekturmodule können nur ein in der Quittung direkt nach Einsendung anzuzeigendes Sofortfeedback anzeigen, aber keine Aufgabenbewertung speichern. Für einen reinen Selbsttest ohne abschließende Korrektur nach Einsendeschluss legen Sie daher weiterhin fest, dass in dieser Selbsttestaufgabe *0 Punkte erreichbar* sind. (Der interne Bewerter errechnet zwar eine Punktzahl und zeigt diese im Sofortfeedback in der Quittung auch an, aber diese wird nicht als in der Aufgabe erreichte Punktzahl gespeichert. Bei reinen Selbstkontrollaufgaben, die innerhalb eines Aufgabenhefts mit z.B. bewerteten Einsendeaufgaben kombiniert werden, ist das auch sinnvoll: Nur letztere speichern eine erreichte Punktzahl und die Selbstkontrollaufgaben gehen mit ihren 0 Punkten nicht in die Aufgabenheft-Gesamtwertung mit ein.)
  - Soll dagegen die in einer Selbstkontrollaufgabe erreichte Punktzahl auch in der Datenbank gespeichert, in der Korrekturübersicht angezeigt werden und so in die Heft-Gesamtwertung mit eingehen (was i.d.R. höchstens Sinn ergibt, wenn das gesamte Aufgabenheft nur aus solchen Selbstkontrollaufgaben besteht), dann richten Sie *sowohl* den Teilaufgabenbewerter als Vorkorrekturmodul *als auch* den Aufgabenbewerter als Korrekturmodul ein und tragen Sie für beide (Vorkorrekturmodul und Korrekturmodul) die Bewerter-Eigenschaften (s.u.) identisch ein!

Nach Auswählen des (Vor-)Korrekturmoduls in der erweiterten Aufgabenerstellung wird ein Button »Eigenschaften…« eingeblendet, über welchen Sie nun die Konfiguration des Moduls bearbeiten können.

Eine solche „Eigenschaften"-Konfiguration kann z.B. wie folgt aussehen:

```
Schablonenoption:
 Musterloesung
 Gesamtbewertung
Init:
 Aufgabentyp: XausN
 Vorgabe: A,B+,C; A,D
 Punkte: 8
 Gewichtung: 1
 N: 4
 Name: Multiple Choice
 Kommentar: Kommentar1
Init:
 Aufgabentyp: BegriffeIC
 Name: Begriffe
 Kommentar: Die FernUni liegt in der Stadt Hagen im Bundesland Nordrhein-
Westfalen (NRW)
 Punkte: 20
 Gewichtung: 1
 Felder: 2
 Uebereinstimmung: 100
 Wortliste:  Hagen|Hagen, Westfalen|Hagen (Westfalen)
 Wortliste:  Nordrhein-Westfalen|NRW|Westfalen
Init:
 Aufgabentyp: Zahlen
 Name: Zahlen
 Punkte: 10
 Gewichtung: 1
 Vorgabe: 1.1 / 1.2

Intervall: 12-15
Textbaustein:
Sehr gut, Sie haben den Stoff verstanden.

Intervall: 8-11
Textbaustein:
Gut!

Intervall: 0-7
Textbaustein:
Übung macht den Meister.
```

Der oberste Bereich `Schablonenoption:` ist optional. Die dahinter angegebenen Optionen steuern den genaueren Aufbau der Seitenvorlage (Template, „Schablone“) für die Textausgaben des (Vor-)Korrekturmoduls. Die Einträge in obigem Beispiel bedeuten, dass die richtige Lösung verraten und im Falle von mehreren Init-Blöcken neben den Einzelwertungen auch eine Gesamtwertung erzeugt werden soll. Darauf geht der nächste Unterabschnitt genauer ein.

Optional kann weiterhin (vor oder hinter dem Bereich `Schablonenoption:`) die Boolean-Property `NurBearbeiteteTeilaufgaben: true` angegeben werden. Sie kann alternativ auch den Wert false annehmen, was auch der Default-Wert bei Nicht-Angabe der Option ist. Diese Option bewirkt, dass für die Teilaufgaben, die ein Student nicht bearbeitet hat (zu deren Feldern keine Einsendungen vorliegen), keine Korrekturausgaben erzeugt werden und somit auch z.B. die korrekten Lösungen nicht verraten werden, sondern eben nur eine Korrektur für die bearbeiteten Teilaufgaben erzeugt werden soll. Diese Option sollte insbesondere dann angegeben werden, wenn Sie die Fragen-Randomisierung <https://online-uebungssystem.fernuni-

hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#fragenrandom> mit zufälliger Fragenauswahl verwenden, damit ein Student zu denjenigen Fragen, die ihm in seiner persönlichen Zufallsauswahl gar nicht gestellt wurden, keine Korrekturen der Art „Sie haben zur Frage x keine Eingabe eingesendet, daher 0 Punkte erreicht, richtig wäre Antwort B gewesen" zu den Fragen bekommt, die er gar nicht sehen und bearbeiten konnte. (Hinweis: Auch nach Einführung der Selektionssubmit-Speicherung ist diese Option weiterhin nötig. Auch wenn eine Extra-Information über die eingesendete Selektion – also welche Teilaufgaben in welcher Reihenfolge in seinem individuellen Aufgabenformular vorkamen – übermittelt wird, wird diese nur zur Sortierung der Korrekturausgaben verwendet und nicht zur Ausfilterung von Einsendungen. Das wurde der Fehlertoleranz halber so beibehalten.)

Zusätzlich zu `NurBearbeiteteTeilaufgaben: true` kann die Property `PunktzahlUnbearbeitet:` gefolgt von einer Zahl angegeben werden. Diese legt fest, welche Punktzahl für „Teilnehmer" vergeben werden soll, die die Aufgabe gar nicht bearbeitet (also zu gar keiner Teilaufgabe etwas eingesendet) haben. Defaultwert ist Null (0).
Diese Einstellung wird jedoch nur in wenigen Ausnahmefällen benötigt. Zunächst gelten im Regelfall nur diejenigen als „Teilnehmer", die die Aufgabe bearbeitet haben. Lediglich falls die Korrekturart auf »automatisch, auch unbearbeitete (Teil-)aufgabe(n) bewerten« eingestellt ist, dann gelten alle Studierenden als „Teilnehmer", die das *Aufgabenheft* bearbeitet haben. Dann soll also auch jede unbearbeitete Aufgabe des Hefts die automatische Bewertung durchlaufen. Im Normalfall gibt es dafür dann immer 0 Punkte. Wenn Sie aber nun eine Aufgabe, z.B. wegen missverständlicher Aufgabenstellung, rückwirkend komplett aus der Wertung nehmen möchten (siehe auch unten folgenden Abschnitt Fragen und Aufgaben aus der Wertung nehmen) und alle Heftteilnehmer die volle Punktzahl bekommen sollen – auch diejenigen, die die Aufgabe nicht bearbeitet haben–, dann stellen Sie besagte Korrekturart ein, konfigurieren Sie die einzelnen Bewerter in ihren `Init:`-Blöcken (s.u.) per `Vorgabe:`-Einstellung so, dass sie für jede Antwort immer volle Punktzahl vergeben. Und falls diese Aufgabe randomisiert war und Sie die oben beschriebene `NurBearbeiteteTeilaufgaben: true`-Einstellung eingefügt hatten, dann ergänzen Sie diese eben um `PunktzahlUnbearbeitet:` und geben hinter dem Doppelpunkt die erreichbare Punktzahl an. (Diese ist bei randomisierten Aufgaben nicht aus den Einzelpunktzahlen in den `Init:`-Blöcken (s.u.) einzeln ableitbar, da die Gesamtpunktzahl noch von der Selektionsgröße, also der Anzahl der den Teilnehmern zufällig ausgewählten Random-Blöcke (Fragen) abhängt.) Für nicht randomisierte Aufgaben ohne die `NurBearbeiteteTeilaufgaben`-Einstellung wird der Bewerter ohnehin auch für jede unbearbeitete Teilaufgabe ausgeführt und vergibt dann bereits volle Punktzahl. Da wird die Einstellung daher nicht benötigt.

Auf obige globale (die gesamte Aufgabe betreffende) Einstellungen folgen dann jeweils durch eine Zeile `Init:` eingeleitete Fragen-Properties: Für jede einzelne Frage in der Teilaufgabe ist genau ein solcher Init-Block zu deren Konfiguration anzugeben. Die Zuordnung der Konfigurationen zur Frage in der Aufgabenseite erfolgt nach der Reihenfolge des Auftretens: Der erste Init-Block konfiguriert die erste Frage der (Teil-)Aufgabe, der darauf folgende Init-Block die zweite Frage u.s.w..

Abschließend finden sich Sequenzen aus je einer `Intervall`-Property (Angabe eines Punkteintervalls) gefolgt von einer `Textbaustein`-Property mit ggf. mehrzeiligem Textkommentar. Diese Angaben entsprechen der Rubrik **»Kommentar zur Gesamtwertung«** des Aufgabenerstellungsassistenten. Unter `Intervall:` ist jeweils ein Punkteintervall in der Syntax `10-20` einzutragen, also eine kleinere (ganze) Zahl, gefolgt von einem Minus-Symbol („bis") und einer größeren (ganzen) Zahl. Wenn dann die von einem/-r Teilnehmer/-in in der Aufgabe insgesamt erreichte Punktzahl in diesem Intervall liegt, wird der hinter dem Intervall unter `Textbaustein:` festgelegte Text hinter der Gesamtwertung (in ganzen Punkten und Prozentpunkten) angezeigt.

Hinweis: Falls Sie in den Kursparametern ein *Punkteformat mit Nachkommastellen* aktiviert haben sollten, so wird immer nur der ganzzahlige Punkteanteil mit diesen Intervallgrenzen vergleichen (und

die Intervallgrenzen selbst müssen immer ganze Zahlen sein).
Beispiel: Angenommen, Sie haben ein Punkteintervall `0-9` sowie ein zweites Intervall `10-20` definiert, und jemand erreicht aber nun 9,5 Punkte. Dann werden diese 9,5 Punkte eben nicht als „zwischen diesen Intervallen" liegend interpretiert, sondern die Nachkommastelle wird ignoriert, und somit wird der Textbausteim zum Intervall `0-9` auch für 9,5 Punkte angezeigt.

Noch ein abschließender Hinweis zur *Zufallsreihenfolge bei Randomisierung*: Falls die randomisierte Aufgabenseite einen Selektionssubmit mit Speicherung der Selektion verwendet, also dem AufgabenBewerter neben den eigentlichen Feldeinsendungen der Studierenden auch jeweils eine Einsendung zur „Pseudo-Teilaufgabe" `$`, Feld Nr. 1, übermittelt wird, dann wird der Aufgabenbewerter seine Korrekturen intern umsortieren. Die Korrektur zum ersten `Init:`-Block wird dazu intern mit dem Buchstaben `A` versehen, die zum zweiten `Init:`-Block mit `B` etc, und in der Autokorrektur-Ausgabe werden dann immer zunächst diejenigen Autokorrektur-Blöcke in der Reihenfolge ausgegeben, wie deren Buchstaben in der „Selektionseinsendung" stehen. Sollte es noch zu Init-Blöcken, deren so ermittelter Buchstabe nicht in der Selektion vorkommt, Einsendungen geben (bzw. nicht-leere Einsendungen bei Option `NurBearbeiteteTeilaufgaben: true`, s.o.), dann werden diese Autokorrekturen dann im Anschluss noch angehängt.

Schablonenoptionen

Hinter dem Schlüsselwort `Schablonenoption:` können Sie eine Liste von Optionen angeben, getrennt durch Leerzeichen und/oder Zeilenumbrüche:

- Die Option `Musterloesung` (oder eigentlich jedes Wort, in dem `Muster` vorkommt, ohne Berücksichtigung der Groß-/Kleinschreibung, also z.B. auch einfach nur `MUSTER`) legt fest, dass in den Einzelbewertungen auch die korrekten Lösungen verraten werden. *Wenn Sie diesen Punkt nicht angeben (aus obigem Beispiel also z.B. entfernen), entfernen Sie also die Anzeige der korrekten Lösung aus den Autokorrekturen dieser Aufgabe.* Dann hat jede Autokorrektur nur noch die Form »Dies war Ihre Antwort:…, und dafür bekommen Sie die folgende Punktzahl.«, ohne genauer zu begründen, was an der Antwort korrekt oder falsch war. Damit wird allerdings auch nicht genauer nachvollziehbar, wie die vergebene Punktzahl zustande kommt. Bei vielen Fragetypen wie Einfachauswahl oder Zahlen gibt es in der Regel ohnehin nur 0 Punkte oder die volle Punktzahl, da ist dann bei Vergabe von 0 Punkten zumindest klar, dass die gegebene Antwort falsch war (ohne dass verraten wird, welche Antwort richtig gewesen wäre). Bei Multiple Choice z.B. wird dem Studenten dann angezeigt, welche Optionen er angekreuzt hat, und wieviele Punkte er für seine Antwort bekommt, jedoch kann man an einer Teilpunktzahl nicht ablesen, welche Teile seiner Antwort richtig und welche falsch waren.
- Die Option `Gesamtbewertung` (oder auch jedes Wort, in dem `gesamt` unabhängig von der Groß-/Kleinschreibung vorkommt, z.B. auch `Gesamtwertung` oder nur `GESAMT`) steuert für Aufgaben mit mehr als einem `Init`-Block, dass nicht nur pro Einzelfrage (`Init`-Block) eine Einzelwertung erzeugt werden soll, sondern abschließend auch noch eine Gesamtbewertung (Gesamtpunkte von gesamt erreichbaren Punkten, Prozentpunkte, optional auch noch von der Gesamtpunktzahl abhängig gewählte Textblöcke, festgelegt über `Intervall`- und `Textbaustein`-Paare, s.o.) ausgegeben werden soll.
- Eine dritte mögliche Option ist `Ueberschrift:` gefolgt von einem Text.
    - Normalerweise hat eine Autokorrektur eine Standardüberschrift wie folgt:
        - Besteht sie aus genau einer Einzelwertung (ein `Init`-Block in den Korrektureinstlelungen, s.u.), dann wird der Titel der Einzelwertung/Frage als Überschrift darüber gesetzt.

        - Bei Korrekturen für mehrere Einzelwertungen (mehrere `Init`-Blöcke) wird eine Tabelle erstellt, in der der Titel der jeweiligen Einzelwertung in der linken Spalte vor der Wertung steht, und es wird eine Standardüberschrift über der gesamten Autokorrektur ausgegeben. Stand Januar 2022 lautet diese Standardüberschrift: »Die Bewertung im Einzelnen:«, Änderungen vorbehalten.
    - Durch Angabe von `Ueberschrift:` überstimmen Sie die jeweilige Standardüberschrift und legen den Überschriftentext selbst fest. Dabei zählt alles, was hinter `Ueberschrift:` bis hin zum Zeilenende steht, als der neue Überschriftentext. (D.h. die anderen Optionen wie `Gesamtbewertung` dürfen vor `Ueberschrift:` oder in einer nachfolgenden Zeile stehen, aber nicht in derselben Zeile hinter `Ueberschrift:`, dann würde das Wort als Teil der Überschrift interpretiert.)
    - Wenn Sie *keinen* Text hinter `Ueberschrift:` angeben, sondern direkt danach eine neue Zeile beginnen (Leerzeichen zwischen Doppelpunkt und Zeilenumbruch sind erlaubt und werden ignoriert), *dann schalten Sie damit die Ausgabe einer Überschrift komplett ab.*

Falls Sie die Optionale Rubrik `Schablonenoption:` aus den Bewertereinstellungen *ganz weglassen*, treten die Defaulteinstellungen in Kraft: Anzeige von Musterlösungen/richtigen Antworten, der Gesamtwertung (bei mehr als einer Frage) und der Standardüberschrift (bei mehr als einer Frage).

Falls Sie *weder Musterlösung noch Gesamtwertung* anzeigen möchten noch eine eigene Überschrift festlegen möchten, geben Sie `Schablonenoption:` an, ohne eine der drei Optionen dahinter zu schreiben.

Das Festlegen einer eigenen Überschrift kann sinnvoll sein, falls …

- Sie Ihre Aufgabenseite randomisieren und dazu den Selektionssubmit nutzen,
- wobei es Random-Blöcke gibt, die aus mehr als einer autokorrigierten Frage bestehen,
- und Sie dabei die Vorkorrektur (Sofortfeedback) nutzen. Dann wurde für jeden Random-Block eine eigene Quittungsseite erzeugt, und nach Einsendung werden die Einzelquittungen pro ausgewerteter Teilaufgabe zu einer Gesamtquittungsseite zusammengefügt. Wenn es in dieser Gesamtquittungsseite dann mehrere Teilabschnitte mit der Standard-Überschrift »Die Bewertung im Einzelnen:« gäbe, in denen eben nur die Autokorrekturen zu den Fragen aus dem betreffenden einzelnen Random-Block folgten, wäre das sehr unübersichtlich. In diesem Fall sollten Sie die `Ueberschrift:`-Option in den Einstellungen des Vorkorrekturmoduls zu einer Teilaufgabe aus mehreren Einzelfragen nutzen, um dort eine sinnvolle Überschrift (z.B. einen Abschnittstitel) einzutragen, analog zum Aufbau in der Aufgabenseite.

Init-Blöcke allgemein

Im Folgenden soll genauer auf die `Init`-Blöcke eingegangen werden. Unabhängig vom Typ der zu bewertenden Frage gibt es einige „globale“ Properties, die immer anzugeben sind:

Property	Mögliche Werte	Bedeutung
`Aufgabentyp`	`1ausN`, `XausN`, `XausNv2`, `XausNv2.1`, `XausNv2.2`, `XausN+`, `XausN3A`, `XausN3Av2`, `XausN3Av2.1`, `XausN3Av2.2`, `XausN3A+`, `1-XausN`, `1-XausNv2`, `1-XausNv2.1`, `1-XausNv2.2`, `1-XausN+`, `Zuordnung`, `Zahlen`, `Begriffe`, `BegriffeIC`, `BegriffeSmart`, `BegriffeSmartIC`, `Begriffsfolge`, `BegriffsfolgeIC`, `BegriffsfolgeSmart` oder `BegriffsfolgeSmartIC`	Bewertermodus, d.h. um was für eine Art von Frage bzw. Art der automatischen Auswertung handelt es sich? (Groß-/Kleinschreibung der Bewertername ist beliebig.)
`Name`	String	Der Fragetitel. Dieser String wird der Auswertung einer Frage vorangestellt und sollte mit einer entsprechenden Überschrift im Aufgabenformular übereinstimmen, um den Bezug der Auswertung zur Frage herzustellen.
`Kommentar`	String	Optionaler Erläuterungstext, der unabhängig vom erreichten Ergebnis unter der Auswertung zu einer Frage angezeigt wird.
`Punkte`	Natürliche Zahl	Die zu vergebende Punktzahl bei korrekter Antwort, bzw. Maximalpunktzahl bei Fragen wie Multiple Choice. Ggf. auch mehrere Punktzahlen durch Semikolon getrennt (maximal soviele wie Vorgaben, unterstützt bei `1AusN`, `XausN`... und `Zahlen`.)
`Gewichtung`	Natürliche Zahl	Sollte normalerweise immer 1 sein. Die erreichte Punktzahl wird mit diesem Faktor multipliziert, wodurch mehr als 100% der Punkte erreicht werden können. Ab sofort (seit November 2023) ist diese Angabe optional, d.h. der Defaultwert ist 1.

Die weiteren Properties sind abhängig vom gewählten Fragentyp (Property `Aufgabentyp`), darauf werden die nachfolgenden Unterabschnitte eingehen.

Typen `1ausN`, `XausN`, `XausNv2`, `XausNv2.1`, `XausNv2.2`, `XausN+`

`1ausN` steht für eine Einfachauswahl aus mehreren Alternativen, z.B. per Radiobutton, bewertet mit voller Punktzahl, wenn der Student die richtige Auswahl getroffen hat, andernfalls mit 0 Punkten.

Technische Seite: Die Einsendung eines Studenten ist ein String, und die Musterlösung (Property `Vorgabe`, s.u.) ist ebenfalls ein String. Stimmen beide überein, wird die Einsendung mit voller Punktzahl bewertet, sonst mit 0 Punkten. Eine typische Realisierung besteht darin, in der Aufgabenseite Radiobuttons oder eine Selectbox einzubinden, über welche ein Student einen aus N vorgegebenen Strings auswählen kann, der dann bei der korrekten Auswahl mit dem Lösungsstring

übereinstimmt. Typische Strings sind einfach nur Buchstaben `A`, `B`, `C` u.s.w., mit denen die Alternativen aufgezählt werden, aber prinzipiell sind beliebige Strings möglich.

Bei den Bewertern `XausN`, `XausNv2`, `XausNv2.1`, `XausNv2.2` und `XausN+` soll ein Student ebenfalls aus N vorgegebenen Lösungsstrings auswählen, aber nicht genau einen, sondern eine beliebige Anzahl (Multiple Choice / Mehrfachauswahl). D.h. es können beliebig viele der Antwortalternativen richtig oder falsch sein, insbesondere auch alle oder gar keine. (Siehe die im Folgenden noch beschriebenen `1-XausN...`-Typen als Variante für Aufgaben, in denen immer mindestens eine der Antwortalternativen richtig ist.)

Die Musterlösung (Property `Vorgabe`, s.u.) zählt dazu durch Kommas getrennt alle korrekten Lösungsstrings auf (darf auch leer sein, wenn keine Antwort richtig ist), und auch eine studentische Einsendung soll eine solche Aufzählung von Strings sein (darf ebenfalls leer sein). Das HTML-Formular sendet entweder in einem Formularfeld direkt eine (Komma- oder Leerzeichen-separierte) Liste von Strings ein, oder es sendet in mehreren Formularelementen gleichen Namens jeweils einen ausgewählten String ein. Ein typisches Beispiel sind Checkboxen, die alle den gleichen Namen (Attribut `name`) haben, aber einen unterschiedlichen String (Attribut `value`) einsenden, wenn sie markiert sind. Aus mehreren unter dem gleichen Namen eingesendeten Strings macht das Übungssystem automatisch eine kommaseparierte Aufzählung, die es dann dem Bewertermodul übergibt.

Die verschiendenen `XausN...`-Bewerter unterscheiden sich nun in der Art der Punktevergabe (bei Teilübereinstimmungen): Alle diese Bewerter vergeben natürlich die volle Punktzahl, wenn eine Einsendung zur Frage zu 100% korrekt ist, sowie 0 Punkte, wenn die Einsendung komplett falsch ist. Aber bei Multiple-Choice-Fragen mit mehrere einzelnen Teilfragen bzw. Antwortalternativen (z.B. Checkboxen), können ja auch einzelne (Teil-)Antworten (Checkbox-Markierungen) korrekt, andere jedoch falsch sein, woraufhin nur eine Teilpunktzahl vergeben wird. Dabei vergibt `XausN+` nur Positivpunkte für die korrekten Antworten, 0 Punkte für falsche. Die anderen Varianten verwenden eine Ratekorrektur: `XausN` verwendet die klassische Lotse-Ratekorrektur, bei der die Anzahl falscher Antworten von der Anzahl richtiger Antworten abgezogen wird, und `XausNv2` steht für eine neuere Bewertung (Ratekorrektur Version 2), die nicht mit Punktabzügen arbeitet, sondern sicherstellt, dass bereits für nur *eine* richtige (Teil-)Antwort anteilig Punkte vergeben werden (wenn auch nur wenig, i.d.R. 1%) – sofern die erreichbare Maximalpunktzahl groß genug ist, dass dieser Wert nicht zu 0 abgerundet werden muss. Von dieser Ratekorrektur V2 gibt es inzwischen drei Versionen (2.0, 2.1 und 2.2), realisiert durch die Bewertertypen `XausNv2` (Version 2.0), `XausNv2.1` sowie `XausNv2.2`. Wenn Sie die Ratekorrektur V2 verwenden möchten, wird heute die Version 2.2 empfohlen, die beiden älteren Versionen werden nur zur Abwärtskompatibilität noch angeboten.

Diese unterschiedlichen X-aus-N-Bewertungen werden in einem separaten Handbuch »Bewertung von Mehrfachauswahlaufgaben« <https://online-uebungssystem.fernuni-hagen.de/download/XausNBewertung/XausNBewertung.html> ausführlich beschrieben, daher gehen wir an dieser Stelle nicht ausführlicher darauf ein, sondern fahren direkt fort mit der Konfiguration der Bewerter:

Für alle diese Bewerter (`1ausN`, `XausN`, `XausNv2`.* und `XausN+`) sind folgende Properties zusätzlich anzugeben:

Property	Mögliche Werte	Bedeutung
`N`	Natürliche Zahl	Anzahl der Antwortalternativen. Bei Multiple Choice (`XausN`...) sollte die Gesamtpunktzahl `Punkte` durch `N` teilbar sein – zumindest bei `XausN` und `XausN+` sind dann alle möglichen Bewertungen ganze Zahlen. Sollte sich eine nicht-ganze Punktzahl für einen Studenten errechnen, wird diese *immer abgerundet* (die Nachkommastellen also abgeschnitten)! Ausnahme: Bei „V2-Bewertung“ (`XausNv2`, `XausNv2.1` oder `XausNv2.2`), bei denen selbst wenn `Punkte` durch `N` teilbar ist, nicht-ganzzahlige Ergebnisse errechnet werden können, werden diese gerundet. (Falls Sie in den Kursparametern ein Punkteformat mit 1 oder 2 Nachkommastellen aktiviert haben, sollten zumindest die gesamten Zehntel- bzw. Hunderstelpunkte, also das zehn- bzw. hundertfache der `Punkte` durch `N` teilbar sein. So wäre z.B. auch `Punkte:1, N:5` bei ein oder zwei Nachkommastellen OK, da 10 durch 5 teilbar ist und jede Antwortalternative also 0,2 Punkte „wert wäre“.)
`Vorgabe`	String	Korrekte Lösung. Bei Einfachauswahl ein einfacher String, der mit dem vom Formular eingesendeten Formularwert (z.B. `value`-Attribut eines Radiobuttons) übereinstimmen soll, bzw. eine Semikolon-separierte Aufzählung mehrerer alternativer Lösungen, die alle mit der vollen Punktzahl bewertet werden sollen. Bei Mehrfachauswahl eine Komma-separierte Liste der richtigen Antworten, die zusammen die Lösung bilden. Ein `+`-Suffix an einer Antwort nimmt diese aus der Wertung[20], der Student bekommt dann für diese Antwortalternative immer die Punkte, unabhängig davon, ob er die Antwort gegeben hat oder nicht. Auch bei Mehrfachauswahl können mehrere alternative Musterlösungskombinationen angegeben werden, indem mehrere dieser Komma-separierten Lösungskombinationen jeweils durch Semikolon getrennt aufgezählt werden. Alternativ `*` für „außer Wertung“, s.u.

Angabe mehrerer alternativer Musterlösungen

Sie können unter `Vorgabe` mehrere Lösungsstrings (bei Einfachauswahl also richtige Lösungen, bei Mehrfachauswahl kommaseparierte Aufzählung der richtigen Antworten) angeben, jeweils getrennt durch Semikolon (`;`).

Für die Bewertung wird dann jeweils diejenige der alternativen Lösungen herangezogen, für die sich die maximale/bestmögliche Gesamtpunktzahl für errechnet, bezüglich der also der Student die meisten richtigen Antworten gegeben hat.

Hier nochmal ein entsprechender Ausschnitt aus dem obigen größeren Beispiel:

```
Init:
 Aufgabentyp: XausN
 Vorgabe: A,B+,C; A,D
 Punkte: 8
 N: 4
 …
```

Dies Konfiguration geht davon aus, dass die erste Frage im Aufgabenformular eine Mehrfachauswahl-Frage mit vier Antwortalternativen ist, deren `value`s die Großbuchstaben `A`, `B`, `C` und `D` sind. Ein Student soll jeweils die volle Punktzahl bekommen, wenn er nur die Antworten `A` und `D` gegeben hat oder wenn er `A` und `C` und optional noch `B`, nicht jedoch `D` geantwortet hat. Antwortet ein Student also z.B. `B,C`, so bekommt er noch 60% der erreichbaren Punkte: Bezüglich Vorgabe `A,D` hätte er 0 richtige Antworten gegeben, bezüglich Vorgabe `A,B+,C` dagegen 3

Richtige (das Nicht-Nennen von A wäre hier der einzige Fehler), also wird gegen diese Vorgabe gewertet.

Angabe mehrerer Punktzahlen

Falls Sie, wie gerade beschrieben, in `Vorgabe` mehrere alternative Musterlösungen (durch `;` getrennt) vorgeben, können Sie auch in der Property `Punkte` mehrere Maximalpunktzahlen (ebenfalls durch `;` getrennt) eintragen:

- Ist nur *eine* Maximalpunktzahl festgelegt (wie im vorigen Beispiel), gilt sie für alle alternativen Lösungen gleichermaßen.
- Sind genauso viele Maximalpunktzahlen aufgezählt wie alternative Vorgaben, gilt eine 1:1-Zuordnung: Stimmt die Einsendung des Studenten mit der ersten möglichen Musterlösung überein, bekommt er die erstgenannte Punktzahl gutgeschrieben, stimmt sie mit der zweiten Lösung überein, bekommt er die zweite Punktzahl gutgeschrieben etc. Stimmt die Eingabe des Studenten mit keiner der Musterlösungen *vollständig* überein, wird für jede Lösung separat die bei Teilübereinstimmungen ggf. erreichbare Teilpunktzahl (von der der jeweiligen Lösung zugeordneten Maximalpunktzahl) errechnet und die bestmögliche Bewertung wird dem Studenten gutgeschrieben.
- Mischform: Falls zwar mehrere Punkte angegeben wurden, jedoch weniger als Lösungen, so wird jeder „überschüssigen" Lösung die erstgenannte erreichbare Punktzahl zugeordnet. Präzise Definition: Sei *x* die Anzahl der unter `Punkte` angegebenen Maximalpunktzahlen und *y* die Anzahl der unter `Vorgabe` angegebenen Alternativlösungen ($1 \leq x \leq y$), und bezeichne p_i die *i*-te angegebene Maximalpunktzahl (für $1 \leq i \leq x$), so wird jeweils der *i*-ten Vorgabe ($1 \leq i \leq y$) die maximal erreichbare Punktzahl *P(i)* zugeordnet:

$$P(i) = \begin{cases} p_i & \text{, falls } i \leq x \\ p_1 & \text{, falls } i > x \end{cases}$$

Beispiel 1: Betrachtet sei folgender Init-Block für eine Einfachauswahlfrage:

```
Init:
      Aufgabentyp: 1ausN
      Vorgabe: A; C; D
      Punkte: 10; 20; 15
      Gewichtung: 1
      N: 5
      Name: Einfachauswahl-Demo
```

In dieser Aufgabe sind maximal 20 Punkte erreichbar, die ein Student genau dann erreicht, wenn er mit `C` antwortet. Antwortet ein Student mit `A`, bekommt er dafür noch 10 Punkte (halbe Punktzahl), für Antwort `D` bekommt er noch 15 Punkte (drei Viertel). Antworten `B` und `E` gelten als „komplett" falsch und werden mit 0 Punkten bewertet.

Beispiel 2: Betrachtet sei folgender Init-Block für eine Mehrfachauswahlfrage:

```
Init:
      Aufgabentyp: XausN+
      Vorgabe: A,B+,C; E; B,C
      Punkte: 100; 50
      Gewichtung: 1
      N: 5
      Name: Mehrfachauswahl-Demo
```

Sendet ein Student nun z.B. eine der Antworten `A,B,C` oder `A,C` oder `B,C` ein, erhält er dafür jeweils 100 Punkte gutgeschrieben. Auch die Antwort `E` wird zwar als volle Übereinstimmung mit einer der drei Lösungen akzeptiert, jedoch als „minderwertige" Lösung nur mit 50 statt 100 Punkten bewertet.
Die Antwort `C,E` dagegen stimmt mit keiner der Lösungen *vollständig* überein, aber stellt eine *Teilübereinstimmung* mit allen drei Lösungen dar. Verglichen mit Lösung `E` liegt nur ein Fehler vor, was der `XausN+`-Bewerter mit 80% von 50 Punkten = 40 Punkten bewerten würde. Verglichen mit Lösung `B,C` liegen zwar 2 Fehler vor, was der `XausN+`-Bewerter jedoch mit 60% von 100 Punkten, also 60 Punkten bewertet. Der Student erhielte bei dieser Konfiguration daher 60 Punkte auf die Antwort `C,E`, da nicht die Antwort mit den wenigsten Fehlern zählt, sondern die, mit der er die bestmögliche Punktzahl erreicht.

Auswahlfrage aus der Wertung nehmen

Falls Sie nach einer Auswertung feststellen sollten, dass eine Auswahlfrage (egal ob Einfach- oder Mehrfachauswahl) missverständlich oder falsch gestellt war, können Sie unter Vorgabe (statt einer oder mehrerer Lösungsvorgaben) einfach ein Sternchen `*` eintragen und die Aufgaben neu auswerten. Die Auswirkung ist, dass jede beliebige Eingabe akzeptiert und mit voller Punktzahl bewertet wird. Die Frage ist damit dann effektiv aus der Wertung.

Man kann alternativ auch einfach alle möglichen Lösungen als alternative Vorgaben aufzählen, z.B. `A+,B+,C+,D+,E+` für eine X-aus-5-Frage oder `A;B;C;D;E` für eine 1-aus-5-Frage), aber die Vorgabe `*` ist deutlich kürzer und unabhänigig von der Menge der Antwortalternativen. Zudem gibt es einen erwähnenswerten Unterschied, falls das Eingabeformular zu einer Auswahlaufgabe auch ungültige Antworten ermöglicht (z.B. ein Textfeld enthält, in das Antworten wie `A, B` eingetippt werden sollen, aber eben auch andere Texte wie `Hallo` eingetragen werden können, oder Checkboxen zu einer Einfachauswahl-Frage, die, obwohl nur eine Antwort richtig sein kann und gegeben werden darf, dennoch mehrere Antworten ermöglichen – was grundsätzlich als Fehler gilt und mit 0 Punkten bewertet wird). In diesen Fällen bewirkt eine `*`-Vorgabe, dass wirklich *jeder* Einsender die volle Punktzahl bekommt, während in den Fällen, in denen die Vorgabe nur sämtliche *gültigen* Antwortmöglichkeiten aufzählt, ungültige Antworten weiterhin mit 0 Punkten bewertet würden.

Typen `1-XausN`, `1-XausNv2`, `1-XausNv2.1`, `1-XausNv2.2`, `1-XausN+`

Diese Typen stehen für Mehrfachauswahlaufgaben, bei denen eine bis X aus N verschiedenen Antwortalternativen korrekt sind. Sie stellen jeweils eine Variante der oben genannten Typen `XausN`, `XausNv2`, `XausNv2.1`, `XausNv2.2` und `XausN+` dar, mit folgenden Besonderheiten:

- Sie sind nur für Fragen anwendbar, für die gilt, dass *mindestens eine Antwortalternative korrekt ist*, also der Vorgabe-String nicht leer ist – während die `XausN…`-Bewerter auch zulassen, dass alle Antwortalternativen falsch sind.
- Vom Studenten wird erwartet, dass er mindestens eine Alternative ankreuzt (bzw. der eingesendete Lösungsstring nicht leer ist)! *Leere Antworten werden grundsätzlich mit 0*

Punkten bewertet, sie werden also als „Aufgabe wurde nicht bearbeitet“ interpretiert. (Im Gegensatz dazu werden bei den `XausN…`-Bewertern grundsätzlich auch leere Einsendungen interpretiert: Sie könnten ja tatsächlich die korrekte Lösung sein oder zumindest Teilpunkte für nicht angekreuzte falsche Alternativen bekommen.)

Für die Bewertung gilt also: Der Bewerter `1-XausN…` vergibt prinzipiell dieselbe Punktzahl wie der `XausN…` mit gleichem Suffix (ohne das Präfix `1-`), außer falls der Student gar keine Antwort gegeben hat (keine Antwortalternative als richtig markiert hat), dann werden grundsätzlich 0 Punkte vergeben, selbst wenn der entsprechende `XausN…`-Bewerter dafür noch Teilpunkte vergeben hätte.

Beispiel: Bei einer Frage mit fünf Antwortalternativen `A` bis `F` sei nur Alternative `A` korrekt (Konfiguration: `Vorgabe: A`). Sendet ein Student nun eine leere Antwort ein, so bekäme er z.B. vom `XausN`- und `XausNV2`-Bewerter noch jeweils $\frac{3}{5}$ der erreichbaren Punkte, vom `XausN+`-Bewerter sogar noch $\frac{4}{5}$, weil er vier richtige und nur eine falsche Antwort gegeben hat. Die `1-XausN…`-Bewerter vergeben dagegen jeweils 0 Punkte, da die Aufgabe ohne Antwort als nicht beantwortet / nicht bearbeitet angesehen wird.

Der `1-XausN…`-Aufgabentyp „zwingt“ die Studenten also dazu, mindestens eine Antwort zu geben. Rät der Student aus obigem Beispiel nun aus diesem Grund (statt gar nichts zu antworten) z.B. Antwort `B`, so erhält er vom `1-XausN`-Bewerter (genau wie vom `XausN`-Bewerter bei gleicher Antwort) noch $\frac{1}{5}$, von `1-XausNv2` (.*) noch 30% und von `1-XausN+` (ebenso wie von `XausN+`) noch $\frac{3}{5}$ der erreichbaren Punkte – in jedem Fall also weniger, als er bei leerer Antwort vom jeweiligen `XausN…`-Bewerter erhalten hätte.

Typen `XausN3A`, `XausN3Av2`, `XausN3Av2.1`, `XausN3Av2.2` und `XausN3A+`

Diese Aufgabentypen stellen eine besondere Alternative zu den oben genannten `XausN`, `XausNv2`, `XausNv2.1`, `XausNv2.2` und `XausN+` dar. Der Grund für deren Unterschiede war ja die Behandlung des Einflusses von Raten auf das Gesamtergebnis: Bei `XausN+` ohne Ratekorrektur sind bereits im Schnitt durch Raten 50% der Punkte erreichbar, erst Ergebnisse ab etwa 75% sind überhaupt als ausreichend zu betrachten. Die beiden anderen Bewerter gehen dagegen mit einer Ratekorrektur vor.

Alle obigen Aufgabentypen haben dabei gemeinsam, dass ein Student zu jeder Antwortalternative eine von zwei möglichen Antworten geben *muss*: „richtig“ oder „falsch“ (bei Checkboxen: angekreuzt oder nicht angekreuzt). Ist er sich bei einer Antwort nicht sicher, *muss* er raten.

Dieser Modus dagegen bietet einem Studenten zu jeder Antwort *drei* Antwortmöglichkeiten: „korrekt“, „falsch“, „keine Antwort / weiß ich nicht“ (daher das Suffix `3A` für „3 Antwortmöglichkeiten“). Dies ermöglicht eine differenziertere Ratekorrektur: Der Student kann bei Antwortalternativen, bei denen er sich nicht sicher ist, selbst entscheiden, ob er die Alternative unbeantwortet lassen will (dann erhält er auf diese nicht beantwortete Alternative anteilig 0 Punkte) oder ob er raten will – mit der Chance auf mehr Punkte bei korrekter Antwort, aber auf weniger Punkte bei falscher Antwort.

Sei *N* die Anzahl der Antwortalternativen und *P* die maximal erreichbare Gesamtpunktzahl, dann ist jede Antwortalternative $\frac{1}{N}\,P$ Punkte „wert“.

Gibt nun ein Student nur zu *m* dieser *N* Alternativen eine Antwort (»richtig« oder »falsch«), so kann er damit entsprechend maximal $P' := \frac{m}{N}\,P$ Punkte erzielen. Die Bewertung der *m* beantworteten Fragen erfolgt dann im Prinzip wie mit dem `XausN`- bzw. `XausNv2`-, `XausNv2.1`- oder

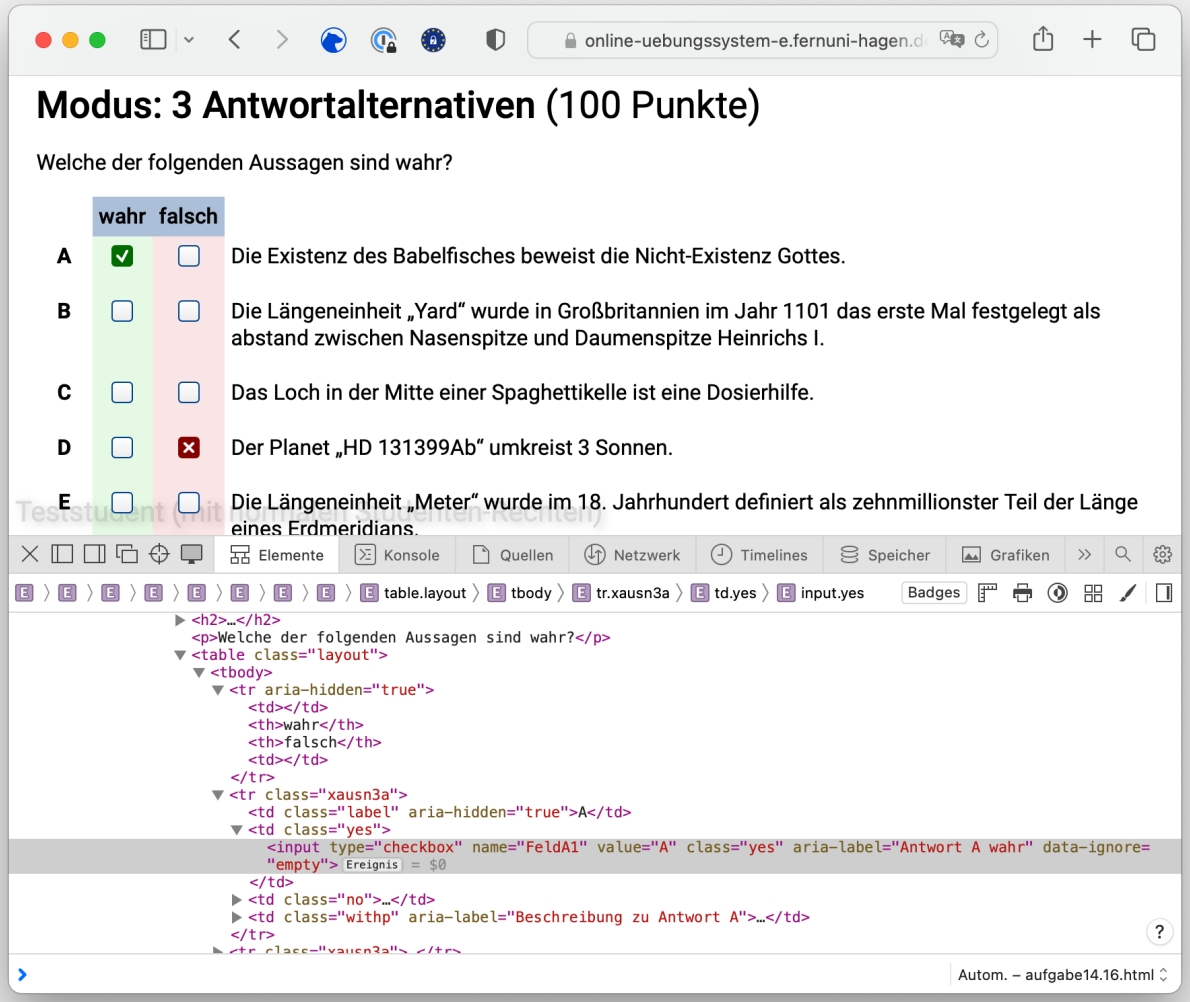
`XausNv2.2`-Bewerter, eben eingeschränkt auf die beantworteten Fragen, d.h. für *m* Alternativen mit erreichbarer Maximalpunktzahl *P'*. Die Bewerter unterscheiden sich dabei wieder in der Ratekorrektur. Der Vollständigkeit halber kann auch dieser 3-Antwort-Modus alternativ ganz ohne Ratekorrektur eingesetzt werden (`XausN3A+`): Dabei macht es dann effektiv keinen Unterschied mehr, ob ein Teilnehmer eine Frage falsch beantwortet oder gar nicht[21].

Auch zu diesen Bewertermodi ist die Bewertung jeweils im Handbuch »Bewertung von Mehrfachauswahlaufgaben« <https://online-uebungssystem.fernuni-hagen.de/download/XausNBewertung/XausNBewertung.html> ausführlich beschrieben.

Die **Konfiguration des Bewertermoduls** erfolgt für alle drei Varianten exakt wie bei den obigen Bewertern auch. Insbesondere wird in der Property `Vorgabe` jede richtige Lösung (in Großbuchstaben) aufgezählt.

Anders ist der **Aufbau der vom Aufgabenformular einzusendenden Strings**, die das Bewertermodul verarbeiten muss. Bei den obigen Bewertern für nur zwei Antwortmöglichkeiten musste der String eine Aufzählung aller vom Student genannten Antworten enthalten. Das genügt bei drei Antwortmöglichkeiten nicht mehr. Vielmehr erwarten diese drei Bewerter eine Aufzählung aller Ja- und Nein-Antworten des Studenten, wobei eine Ja-Antwort nur aus Großbuchstaben bestehen darf, eine Nein-Antwort nur aus Kleinbuchstaben. Eine Antwort, die in der Einsendung gar nicht genannt wird (weder in Groß- noch in Kleinschreibung), gilt als „keine Antwort" (also 0 Punkte).

Beispiel: Angenommen, eine Aufgabe bestehe aus fünf Antwortalternativen A bis E, und die Musterlösung (Property `Vorgabe`) laute `A, C`. Eine Einsendung eines Studenten könnte nun lauten: `A, b, d, E`, d.h. der Student hätte A und E als richtig eingestuft, B und D als falsch, und zu C hätte er keine Antwort gegeben. Damit hätte er 4 von 5 Antworten gegeben, davon $r = 3$ richtige (Einstufung von A als korrekt und von B und D als falsch) und $f = 1$ falsche (E), d.h. er bekäme $r - f = 3 - 1 = 2$ Fünftel der erreichbaren Punkte (im Modus `XausN3A`).



Realisierung des Aufgabenformulars durch den Aufgabenerstellungsassistenten

Ein Aufgabenformular dazu kann man wieder mit Checkboxen realisieren, allerdings mit zwei Checkboxen zu jeder Antwortalternative (jeweils mit groß- und kleingeschriebenem Lösungsstring als `value`). Vgl. obige Abbildung, die zeigt, wie der Aufgabenerstellungsassistenten <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#ab_assi> ein solches Formular aufbaut. Wenn solche zwei Checkboxen verwendet werden, ist eine Skriptsteuerung sinnvoll, die gleichzeitiges Ankreuzen beider Antworten (richtig & falsch) zur selben Alternative vermeidet. Fehlt eine solche Maßnahme und markiert tatsächlich ein Student eine Antwort gleichzeitig als richtig und als falsch, so erhält er darauf sowohl Plus- als auch Minuspunkte, die sich effektiv auslöschen, d.h. die Antwort wird im Endeffekt genauso mit 0 Punkten bewertet, als hätte er sie gar nicht beantwortet.

Aus der Wertung genommene Alternativen

Wie eingangs schon erwähnt, können (für alle Mutliple-Choice-Bewerter) einzelne Alternativen aus der Wertung genommen werden, indem in der Property `Vorgabe` die Alternative mit einem `+` markiert wird. (Im Aufgabenerstellungsassistenten kann dazu in der erweiterten Ansicht eine „Egal“-Checkbox markiert werden.)

Für die „3A-Bewerter“ gilt dabei: Nicht nur Teilnehmer, die zu dieser Alternative eine Antwort gegeben haben (also „richtig“ oder „falsch“ markiert haben), sondern jeder, der die Aufgabe bearbeitet hat, d.h. auch jeder, der hier gar nichts markiert hat, erhält die anteiligen Punkte auf diese

Alternative, so als hätte er sie korrekt markiert.

Beispiel: Angenommen sei eine X-aus-5-Aufgabe (3A) mit `Vorgabe: A,B,C+`. Dann erhielten z.B. die Einsendungen `A,B,C,d,e`, `A,B,c,d,e`, aber auch `A,B,d,e` jeweils die volle Punktzahl, und `A` würde z.B. nicht nur wie *eine* korrekte Antwort gewertet, sondern wie *zwei* (nämlich Alternativen A und C korrekt beantwortet).

Analog zu den anderen Multiple-Choice-Fragen können Sie natürlich auch hier mittels `Vorgabe: *` die komplette Frage aus der Wertung nehmen (und jedem Einsender volle Punkzahl geben).

Sonderoption für alle obigen Bewerter bei Alternativen-Randomisierung

Für alle Einfach- und Mehrfachauswahl-Bewerter (wie z.B. `1ausN`, `XausN`, `1-XausN` oder auch `XausN3a`) gilt im Normalfall, dass sie genau eine Eingabe aus einem Eingabefeld entgegennehmen und bewerten. (Bei Einfachauswahl enthält das Eingabefeld genau eine Antwort, bei den anderen eine kommaseparierte Aufzählung von Antworten, wobei die Antworten in der Regel durch einfache Buchstaben A, B, C, … dargestellt werden.)

Inzwischen ist es auch möglich, die Antwortalternativen zu randomisieren, also z.B. in eine besondere Reihenfolge zu bringen oder jedem Teilnehmer eine zufällige Teilmenge von Antwortalternativen aus der Aufgabenstellung zu präsentieren, siehe Abschnitt Blocklokale Randomisierung. Wenn dabei die Zuordnung der Buchstaben zu den Antwortalternativen nicht beibehalten werden soll, sondern vielmehr die Liste der Antwortalternativen *nach Randomisierung* mit A beginnend „durchbuchstabiert" werden soll, also für jeden Teilnehmer der jeweilige Buchstabe eine andere, individuelle Zuordnung zu einer ausgewählten Antwortalternative haben kann, dann muss das Korrekturmodul diese Zuordnung kennen. Dafür ist folgender Mechanismus vorgesehen:

1. Die Inputs zur Einsendung (wie RadioButtons bei Einfachauswahl oder Checkboxen bei Mehrfachauswahl) sollten immer dieselbe, konstante Antwortkennung als `value` verwenden, damit die eigentliche, im Online-Übungssystem gespeicherte Antwort am Ende unabhängig von der individuellen Randomisierung ist. Dazu können auch einfache Buchstaben verwendet werden, der Aufgabenerstellungsassistent z.B. stellt diesen intern verwendeten Kennungen (die den Studierenden nicht angezeigt werden) aber zur besseren Abgrenzung von den individuell anzuzeigenden Buchstaben einen Unterstrich voran.
2. Über ein zweites Eingabefeld, realisiert durch `hidden`-Inputs, wird für jede einem/-r Teilnehmer/-in präsentierte Antwortoption eine Zuordnung des (per `$Counter` erzeugten) individuellen Buchstabens zur internen Kennung erzeugt.
3. Dem Bewerter wird in den Properties über eine bestimmte Option (s.u.) mitgeteilt, dass es nicht nur ein Eingabefeld auswerten soll, sondern zwei, von denen das erste die eigentliche Einsendung (aus internen Kennungen bestehend) und das zweite eine Abbildung von internen auf teilnehmerindividuelle Kennungen enthält. Die Auswertung erfolgt dann allein anhand der internen Kennung, aber in der Ausgabe/Anzeige der Autokorrektur werden alle internen Kennungen entsprechend dieser individuellen Abbildung durch die jeweiligen individuellen Kennungen des Teilnehmers ersetzt.
4. Optional kann noch eingestellt werden, ob diese Abbildung interner auf individuelle Kennungen in der Autokorrektur (als Tabelle) mit angezeigt werden soll oder nicht. Das ergibt nur Sinn, wenn die internen Kennungen nicht „intern bleiben", sondern z.B. ein Aufgaben- oder Lösungs-PDF veröffentlicht wird, das für alle Teilnehmer gleich ist (also nicht individualisiert / randomisiert) und daher die internen Kennungen verwendet. Dann kann jede/-r Teilnehmer/-in anhand dieser Zuordnungstabelle später leichter nachvollziehen, welche der ihm im Online-Übungssystem angezeigten Antwortoptionen welcher Angabe im PDF entspricht.

Im HTML-Formular könnte also z.B. etwas der folgenden Art stehen:

```
$Fixed1
$Randomize(3, Shuffle)
$SetCounter.1(A)
$Random.1
<label for="mcA1A">$Counter.1</label>
<input  id="mcA1A" type="checkbox" name="FeldA1" value="_A" data-
ignore="empty">
<input type="hidden" name="FeldA2" value="_A/$Counter.1" class="no-restore"
data-ignore="always">
…Antworttext…
$/Random.1

$Random.2
<label for="mcA1B">$Counter.1</label>
<input  id="mcA1B" type="checkbox" name="FeldA1" value="_B">
<input type="hidden" name="FeldA2" value="_B/$Counter.1" class="no-restore">
…Antworttext…
$/Random.2

…u.s.w.
```

Für einen Teilnehmer könnte dann der Block `Random.2` (mit Value `_B`) als erste Antwortmöglichkeit angezeigt werden, per `$Counter.1`-Variable beschriftet mit `"A"`. Nehmen wir weiter an, dass ein fünfter Block (mit Option `_E`) als zweite Option (`"B"`) präsentiert wurde, und der erste Block (`_A`) als dritte Option (`"C"`). Und der Teilnehmer würde von diesen drei ihm präsentierten Optionen (A, B, C) nur A und C ankreuzen.

Dann sähe die Einsendung wie folgt aus:

- `FeldA1` = `_B,_A`
- `FeldA2` = `_B/A,_E/B,_A/C`

Genau dieses Format also eine Roheinsendung (kommaseparierte Aufzählung von „internen“ Werten) in einem ersten Eingabefeld sowie Mapping (kommaseparierte Aufzählung von Paaren, ein Paar pro dem Studenten präsentierter Antwortoption, jeweils bestehend aus erstens der internen Antwortkennung und zweitens – getrennt durch Querstrich – der dem Studenten dazu angezeigten individuellen Kennung) im nachfolgenden Eingabefeld erwartet der Bewerter, falls in seinen Properties die folgende Einstellung angegeben wird:

Property	Mögliche Werte	Bedeutung
`AnzeigeAbbFolgeFeld:`	leer oder `zeigen`	Unabhängig vom Wert steuert die reine Existenz dieser Property, dass das Korrekturmodul nicht ein, sondern zwei Eingabefelder in Folge auswertet, wobei es im ersten die eigentliche Einsendung erwartet, im zweiten eine Anzeigeabbildung als kommaseparierte Liste von Paaren, wobei jedes Paar aus einem Wert besteht, wie er in der Einsendung vorkommen kann, gefolgt von einem Querstrich `/` als Trennzeichen und einem Wert, der in der Autokorrektur an Stelle des erstgenannten „Rohwertes" jeweils angezeigt werden soll. Wenn der Property der Text `zeigen` als Wert zugeordnet wird, so wird diese Anzeige-Abbildung (Mapping) in der Autokorrektur noch extra in Tabellenform mit ausgegeben.
`N:`	natürliche Zahl	Bei Randomisierung von Antwortalternativen ist hier an Stelle der Gesamtzahl aller Alternativen die jeweils präsentierte / von jedem/-r Teilnehmer/-in zu beantwortende Anzahl von Fragen anzugeben (also identisch mit der Auswahlgröße in der Randomize-Variablen).

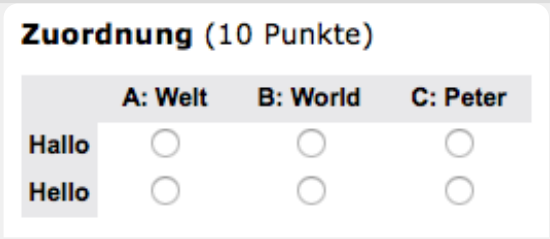
Typ `Zuordnung`

Eine Zuordnungsaufgabe ist eine Sammlung von mehreren Einfachauswahl-Subfragen in einer Frage, wobei die Menge der Antwortalternativen zu jeder Teilfrage identisch ist. Die typische Darstellung ist daher eine Art Matrix mit den Teilfragen als Zeilen und den Antwortalternativen als Spalten (vgl. Abschnitt zum Aufgabenerstellungsassistenten <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#ab_assi>).

Die Konfiguration erfolgt daher fast genauso wie beim `1ausN`-Bewerter:

Property	Mögliche Werte	Bedeutung
`N`	Natürliche Zahl	Anzahl der Antwortalternativen (für alle Teilfragen gleich)
`Felder`	Natürliche Zahl	Anzahl der Teilfragen (= Anzahl der Eingabefelder für die Antworten)
`Vorgabe`	String	Durch Pipe ('|') separierte Liste von Vorgabe-Strings für den 1-aus-N-Bewerter (siehe vorhergehenden Abschnitt). Bei 5 Teilfragen also erst die Vorgabe für die erste Teilfrage (erwartete Antwort oder Semikolon-separierte Liste alternativer Antworten), dann ein Pipe, gefolgt von der Vorgabe für die zweite Teilfrage etc.
`Fragen`	String	Optional: Durch Pipe ('|') separierte Liste der Teilfragen (Fragestellungen oder Namen der Teilfragen). Wenn angegeben, wird in der Auswertung dieser String jeweils der Auswertung der jeweiligen Teilfrage vorangestellt, um den Bezug der Auswertung zur Teilfrage herzustellen. Wird diese Property nicht angegeben, findet sich in der Auswertung einfach nur eine Liste von Einfachauswahl-Auswertungen, die alle mit dem Titel der Zuordnungsfrage allein beschriftet sind.

Nehmen wir als Beispiel die folgende Zuordnungsmatrix an:



Beispiel für eine Zuordungsfrage mit alternativen Antwortmöglichkeiten

Dem deutschen Wort »Hallo« soll der Student wahlweise das deutsche »Welt« oder den englisch wie deutsch gleich geschriebenen Namen »Peter« zuordnen dürfen, nicht jedoch das englische Wort »World«. Entsprechend darf dem englischen »Hello« alternativ »World« oder auch »Peter« zugeordnet werden. Die Konfiguration (unter der Annahme, dass die einsendbaren Values die Buchstaben `A`, `B` oder `C` sind wie über der Tabelle angegeben) könnte dann wie folgt aussehen:

```
Init:
      Aufgabentyp: ZUORDNUNG
      Name:  Zuordnung
      Kommentar:
      Punkte: 10
      Gewichtung: 1
      N:  3
      Felder: 2
      Vorgabe: A;C|B;C
      Fragen: Hallo|Hello
```

Spezielle Optionen bei Randomisierung

Bei Zuordnungsaufgaben besteht prinzipiell die Möglichkeit, die Antwortmöglichkeiten (typischerweise Spalten einer Tabelle) oder die einzelnen Fragen (typischerweise Zeilen einer Tabelle) zu randomisieren (siehe Beispiel 3 in Abschnitt Blocklokale Randomisierung).

Beim Randomisieren von Antwortoptionen, die analog zu Einfachauswahlaufgaben oben mit Buchstaben als Antwortkennungen versehen werden, kann dasselbe Verfahren angewendet werden wie bei obigen Bewertern auch (siehe Sonderoption für alle obigen Bewerter bei Alternativen-Randomisierung). Die entsprechende Bewerterproperty `AnzeigeAbbFolgeFeld` steht daher auch bei Aufgabentyp `ZUORDNUG` zur Verfügung.

Beim Randomisieren von Fragen dagegen sollte dem Korrekturmodul noch separat mitgeteilt werden, welche der Fragen (im Falle einer Zufallsselektion) dem/der jeweiligen Studenten/-in tatsächlich gestellt wurden (damit zu den nicht gestellten/ausgeblendeten Fragen auch keine Autokorrektur erzeugt wird) und/oder (im Falle einer Zufallsreihenfolge) in welcher Reihenfolge, damit auch die Autokorrekturausgaben in derselben Reihenfolge erfolgen.

Zu letzterem Zweck ist den eigentlichen Eingabefeldern (eines pro Frage) ein weiteres Eingabefeld voranzustellen, das eine kommaseparierte Liste von Fragennummern enthält (beginnend bei 1 für die erste Frage der Zuordnungsaufgabe), und die Existenz eines solchen Feldes ist über die spezielle Property `RandomFeld1:` anzuzeigen.

Property	Mögliche Werte	Bedeutung
`AnzeigeAbbFolgeFeld:`	leer oder `zeigen`	Unabhängig vom Wert steuert die reine Existenz dieser Property, dass das Korrekturmodul pro Frage nicht ein, sondern jeweils zwei Eingabefelder in Folge auswertet, wobei es im ersten die eigentliche Einsendung erwartet, im zweiten eine Anzeigeabbildung analog zur obigen Beschreibung bei Auswahlaufgaben.
`RandomFeld1:`	leer	Die Existenz dieser Property zeigt an, dass das erste Eingabefeld der Zuordnungs-Einsendung eine Aufzählung von Fragennummern enthält. Nur diejenigen Fragen, die in dieser Liste vorkommen, sollen überhaupt ausgewertet werden, und die Ausgabe der Autokorrekturen sollen in der Reihenfolge gemäß dieser Liste erfolgen.
`Felder:`	Natürliche Zahl	Bei Randomisierung ist hier nicht die Gesamtzahl der Fragen und erst recht nicht die Anzahl aller Eingabefelder (einschließlich der versteckten Abbildungsfelder, s.o.) anzugeben, *sondern genau die Anzahl der jedem/-r Teilnehmer/-in zu stellenden Fragen*, also dieselbe Zahl wie in der `$Randomize`-Variablen.
`Punkte:`	Natürliche Zahl	Die hier angegebene Punktzahl muss bei Randomisierung durch die Anzahl der jeweils gestellten Fragen (also durch den Wert von `Felder:`) teilbar sein! (Falls Sie in den Kursparametern ein Punkteformat mit 1 oder 2 Nachkommastellen aktiviert haben, genügt es auch, wenn das 10- bzw. 100-fache der Punktzahl durch `Felder` teilbar ist. Falls Sie z.B. zehn Teilfragen stellen und insgesamt 2 Punkte vergeben, so wäre das 10-fache von 2 = 20 durch 10 teilbar, und für jede korrekte Antwort würden 2 Zehntelpunkte = 0,2 Punkte vergeben.)

Typ `Zahlen`

Gefragt wird nach einer reellen Zahl, die in einem bestimmten Lösungsintervall liegen muss. Jede Antwort in diesem Intervall erhält die volle Punktzahl, Antworten außerhalb des Intervalls 0 Punkte.

Property	Mögliche Werte	Bedeutung
`Vorgabe`	Zwei Fließkommazahlen	Es sind zwei Zahlen anzugeben, um ein Intervall festzulegen, in welchem die eingegebene Lösung des Studenten liegen muss. Ein Vorzeichen ist optional, als Dezimaltrenner darf wahlweise Punkt oder Komma verwendet werden, Tausendertrenner sind daher nicht erlaubt. Das Trennzeichen zwischen den beiden Intervallgrenzen ist nicht vorgegeben, ein Komma eignet sich aber nicht, da es als Dezimaltrenner gelesen werden könnte, und das Semikolon eignet sich ebenfalls nicht, da es zur Aufzählung alternativer Lösungsintervalle dient (s.u.). Der Aufgabenerstellungsassistent trennt die beiden Angaben z.B. per `/`. Optional können eckige Klammern als Intervallklammern notiert werden, die explizit signalisieren, ob die jeweilige Intervallgrenze Teil des Lösungsintervalls sein soll oder nicht. Außerdem ist Vorgabe `*` für „außer Wertung“ möglich, s.u..
`InputType`	`number` oder `text`	Optionale Angabe, Default ist `text`. Der Wert `number` sollte genau dann übergeben werden, wenn die Zahleneingabe im Aufgabenfeld in einem `<input type="number"...>` erfolgt. Dies bewirkt dann zum Ersten eine abweichende Darstellung der Eingabe in der Autokorrektur unter »Ihre Einsendung«: Da Number-Felder z.B. bei Fließkommazahlen immer einen Punkt als Dezimaltrenner einsenden (selbst wenn dem Benutzer je nach Landeseinstellung vom Browser ein Komma angezeigt wird), ersetzt der Bewerter hier in der Ausgabe den Punkt wieder durch ein Komma. Zum Zweiten kann auch das Parsen der Zahlen für die Bewertung abweichen: Eingabe wie `"1.000"` in Texteingabefeldern werden heuristisch als 1000 statt 1,0 gewertet. Bei Zahlen-Inputs dagegen, wo selbst eine Eingabe von `"1,000"` als `"1.000"` eingesendet wird, weil der Browser immer einen Punkt als Dezimaltrenner verwendet und keine Tausendertrenner unterstützt, wird der Bewerter dies auch bei `InputType: number` als 1 und nicht mehr (wie sonst) als 1000 bewerten! (Siehe auch Ausführungen zu Quittungsvariablen `$FeldA1NUM` vs. `$FeldA1DECIMAL`.)

Beispiel für die Angabe von Intervallklammern: Bei Angabe von `Vorgabe: ]0.0 / 42.0]` gehört die Zahl 0 selbst nicht mehr zum Intervall, für eine korrekte Lösung muss gelten: $0 < \text{Lösung} \leq 42$. (D.h. Eingaben wie 0.001 oder 0.0000003 gelten noch als korrekt, 0.0 jedoch schon als falsch.)

Wie gesagt sind die Klammern optional, lässt man die Klammern weg, gelten die Intervallgrenzen implizit als eingeschlossen. Die Vorgabe `0,0 / 10,0` ist z.B. gleichbedeutend mit `[0,0 / 10,0]`.

Werden unter `Vorgabe` mehr als zwei Zahlen angegeben, so werden alle ab der dritten ignoriert. Wird nur eine Zahl angegeben, so muss die studentische Eingabe genau dieser Zahl entsprechen, das Lösungsintervall besteht also nur aus genau dieser Zahl als oberer und unterer Grenze.

Wie bei den obigen Bewertern gilt auch hier: Es können mehrere alternative Vorgaben durch Semikolon getrennt aufgezählt werden.

Angabe mehrerer Punktzahlen

Falls Sie unter `Vorgabe` mehrere alternative Antwortmöglichkeiten aufzählen, gilt jede davon als richtig und wird mit der volle Punktzahl bewertet. Außerdem haben Sie auch hier wieder die Möglichkeit, jeder möglichen richtigen Antwort, die Sie unter `Vorgabe` aufzählen, unterschiedliche erreichbare Punktzahlen zuzuordnen, indem Sie in der `Punkte`-Eigenschaft mehrere Werte durch `;` getrennt aufzählen (wie schon oben zu den `1ausN`- und `XausN*`-Bewertern beschrieben).

Beispiel: Angenommen sei folgende Bewerterkonfiguration für eine Zahlen-Frage:

```
Init:
      Aufgabentyp: Zahlen
      Name: Zahlen
      Punkte: 10; 20
      Gewichtung:1
      Vorgabe: -42 / -40; 40 / 42
```

Bei dieser Konfiguration wird jede positive Eingabe zwischen 40 und 42 mit 20 Punkten bewertet, bei Minus-Vorzeichen (Zahl zwischen -42 und -40) werden noch 10 Punkte vergeben, alle anderen Eingaben werden mit 0 Punkten bewertet.

Zahlen-Frage aus der Wertung nehmen

Falls Sie nach einer Auswertung feststellen sollten, dass eine Zahlenfrage missverständlich oder falsch gestellt war, so dass weder durch die ursprünglich eingestellten Lösungsintervalle eine gerechte Bewertung möglich ist, noch sich die Musterlösung geeignet anpassen lässt, können Sie unter Vorgabe (statt eines Lösungsintervalls) einfach ein Sternchen `*` eintragen und die Aufgaben neu auswerten. Die Auswirkung ist, dass jede beliebige Eingabe akzeptiert und mit voller Punktzahl bewertet wird. Die Frage ist damit dann effektiv aus der Wertung.

Typen `Begriffe`, `BegriffeIC`, `BegriffeSmart` und `BegriffeSmartIC`

Diese Bewerter gehören zu Fragen, bei denen ein Student einen oder mehrere Begriffe eingeben soll. Dazu wird eine Menge korrekter Begriffe als Musterlösung festgelegt, aus denen der Student alle oder eine Teilmenge nennen muss. Zu jedem Begriff können optional Synonyme definiert werden.

Die beiden Bewerter `Begriffe` und `BegriffeIC` verhalten sich fast identisch, der einzige Unterschied ist die Behandlung der Groß-/Kleinschreibung in den studentischen Antworten: `BegriffeIC` ignoriert die Groß-/Kleinschreibung – „IC“ steht hier für „ignore case“ –, `Begriffe` dagegen besteht auf korrekte Groß-/Kleinschreibung.

Der Zusatz `Smart` im Namen aktiviert den unten näher beschriebenen *Smart-Modus*. Smart-Modus und Ignorieren der Groß-/Kleinschreibung lassen sich natürlich auch kombinieren (`BegriffeSmartIC`).

Property	Mögliche Werte	Bedeutung
`Felder`	Natürliche Zahl	Anzahl der vom Studenten einzugebenden Begriffe, d.h. der Eingabefelder.
`Uebereinstimmung`	Prozentzahl (1..100) oder `RegEx`*	Bei Wert `100` muss eine studentische Eingabe exakt mit einem Vorgabewort übereinstimmen, um gewertet zu werden, bei kleineren Werten dürfen Abweichungen in entsprechend vielen Zeichen bestehen, um fehlertoleranter bei Tippfehlern zu bewerten, siehe unten. Falls keine Prozentzahl angegeben wird, sondern der String `RegEx` (auch alternative Schreibweisen wie `rex` oder `regexp` oder `regular expression` sind möglich, Groß-/Kleinschreibung jeweils beliebig), so werden die Vorgabestrings aus `Wortliste` (s.u.) nicht mehr als String-Literale interpretiert, sondern als reguläre Ausdrücke (Textmuster, die den Aufbau korrekter Lösungseingaben beschreiben).
		Die Property `Wortliste` darf mehrfach vorkommen, es ist je eine Wortliste für jeden abgefragten Begriff anzugeben. (Die Anzahl der

Property	Mögliche Werte	Bedeutung
		`Wortliste`-Properties muss ≥ dem Wert der `Felder`-Property sein: Bei Gleichheit wird eine Eingabe aus jeder Wortliste verlangt, bei mehr Wortlisten als Feldern eben nur eine Teilmenge davon.)
`Wortliste`	Strings	Innerhalb jeder Wortliste können, durch das Pipe-Symbol ('|') getrennt, verschiedene Synonyme aufgezählt werden. Leerzeichen vor und nach dem Pipe-Symbol sind erlaubt, aber nicht nötig: Sie werden ignoriert. Das heißt auch, dass Begriffe in der Wortliste zwar Leerzeichen *enthalten* können, aber nicht mit Leerzeichen beginnen oder enden können. (Sollte in einem Begriff selbst das Pipe-Symbol als sichtbares Zeichen vorkommen, stellen Sie diesen Vorkommen, die kein neues Synonym einleiten sollen, einen Backslash `\` voran. In Begriffen der Wortliste vorkommende Backslashes selbst sind *nicht* zu escapen, insb. also nicht doppelt zu schreiben. Sollte das letzte Zeichen eines Wortes ein Backslash sein und darauf ein weiteres Wort in der Liste folgen, schreiben Sie mindestens ein Leerzeichen zwischen den Backslash und das Pipe-Symbol, damit letzteres als Worttrenner und nicht Wortbestandteil gelesen wird.)
`Wortliste-Separator`	Zeichen	Trennzeichen für den Split-Modus (s.u.): Falls die einzelnen Begriffe einer Wortliste eine Aufzählung von Teilbegriffen darstellen, ist hier das Aufzählungszeichen dazu anzugeben. Diese Property ist nur einmalig anzugeben, d.h. in jeder Wortliste ist dasselbe Trennzeichen zu verwenden.
`Teilwortanzahl`	Zahl	Kann direkt auf eine `Wortliste` folgen, falls ein `Wortliste-Separator` verwendet wird, die Begriffe in der `Wortliste` also jeweils aus mehreren Teilbegriffen bestehen, und von der Menge dieser Teilbegriffe jedes Begriffs jeweils genau diese Zahl an Begriffen in beliebiger Reihenfolge abgefragt werden soll (Split-Modus für Teilmengen von Teilbegriffen, s.u.). Wenn dieser Modus genutzt werden soll, muss immer *direkt hinter jeder Wortliste* diese Teilwortanzahl-Property stehen und zu jeder Wortliste denselben Wert haben (nur beim Begriffe-Bewerter, nicht beim Begriffsfolge-Bewerter).
`InputType`	`number` oder `text` (ggf. Aufz.)	Optionale Angabe, Default ist `text`. Ist auf `number` zu setzen, falls der Smart-Modus verwendet wird und ausschließlich Zahlen eingegeben werden sollen, kein Text, und zu diesem Zweck im HTML-Formular sämtliche Begriffseingabefelder als `<input type="number"…>` realisiert wurden. Siehe obige Ausführungen zu Typ `Zahlen` zur Auswirkung. Im Split-Modus für eine *Teilwortsequenz* ist hier eine kommaseparierte Aufzählung anzugeben, mit je einem Wert pro Input für ein Teilwort. Soll ein Begriff also z.B. aus zwei Teilworten bestehen, von denen das erste ein Text, das zweite eine Zahl ist, und für die zweite Eingabe ein `<input type="number">` verwendet wird, so ist `InputType: text, number` anzugeben.

Betrachten Sie die Konfiguration zur Begriffs-Frage im obigen Beispiellisting (in der Einleitung des Abschnitts 'Konfiguration der internen Module'): Dort wird nach zwei Begriffen gefragt und es werden zwei Wortlisten angegeben, d.h. der Student muss aus jeder der Wortlisten eines der Synonyme nennen. (Es kann auch mehr Wortlisten als verlangte Begriffe geben, dann wird eben nur nach einer Teilmenge der akzeptierten Begriffe gefragt.) Eine zugehörige Frage zu dieser Bewerterkonfiguration könnte lauten: „Nennen Sie die Stadt und das Bundesland, in denen die FernUniversität ihren Sitz hat." Der Student erhält zwei Eingabefelder und soll die Begriffe „Hagen" und „Nordrhein-Westfalen" angeben. Jeder dieser beiden Begriffe wird als `Wortliste`-Property festlegt, und zu jedem wurden mehrere Synonyme (alternative Schreibweisen) festgelegt. Die Übereinstimmung wurde auf 100% belassen, was jedoch durch den Bewerter `BegriffeIC` insofern abgeschwächt wurde, als dass bei der Bewertung die Groß-/Kleinschreibung nicht

berücksichtigt wird.

Es ist wichtig, dass im Beispiel „NRW“ und „Nordrhein-Westfalen“ Synonyme sind, also in derselben Wortliste stehen: Bei Eingabe von „NRW“ in eines der beiden Eingabefelder und von „Nordrhein-Westfalen“ ins zweite wird so nur die halbe Punktzahl vergeben, da dies als Doppelnennung desselben Begriffs (nur in verschiedenen Schreibweisen) erkannt wird. Hätte man beide Schreibweise als separate Wortlisten definiert, würden beide Schreibweisen als unabhängige Begriffe gewertet, d.h. diese Eingabe erhielte die volle Punktzahl.

Übereinstimmungs- / Toleranzeinstellung für Literalvergleich

Im Normalfall werden Ihre Musterbegriffe aus den Wortlisten als String-Literale (Textkonstanten) interpretiert, und mit der Einstellung `Uebereinstimmung` legen Sie fest, wie exakt Eingaben mit den von Ihnen vorgegebenen Lösungsbegriffen übereinstimmen müssen, um als korrekte Begriffsnennung gewertet zu werden. 100 steht für hundertprozentige, also exakte Übereinstimmung, jeder Tippfehler führt dazu, dass der Begriff nicht erkannt wird[22].

Kleinere Werte erhöhen die Toleranz, d.h. je kleiner diese Zahl, desto mehr dürfen studentische Eingaben von der vorgegebenen Schreibweise abweichen und werden dennoch als korrekt anerkannt. Ist z.B. ein vorgegebener Begriff 10 Zeichen lang und ein Teilnehmer gibt diesen Begriff mit einem abweichenden Buchstaben oder einem Buchstaben zu viel oder zu wenig ein, sind also 9 von 10 Buchstaben korrekt, so wird das als 90-prozentige Übereinstimmung gewertet. Dasselbe gilt für genau einen „Buchstabendreher“ in der Eingabe (z.B. „Buchtsaben“ statt „Buchstaben“). Bei einer Einstellung von 90% würden diese Eingaben also noch akzeptiert, bei größerem Wert dagegen als falsch gewertet. Mit steigender Toleranz (kleineren Werten für »Übereinstimmung«) werden also mehr Tippfehler akzeptiert, es steigt aber auch die Wahrscheinlichkeit, dass falsche Antworten als korrekt gewertet werden (sog. False Positives).

Reguläre Ausdrücke

Textvergleiche (also die Prüfung aller Eingaben mit Ausnahme der Vergleiche von Texteingaben mit numerischen Vorgaben im Smart-Modus, s.u.) erfolgen normalerweise als Literalvergleich mit Übereinstimmungsmaß, wie oben gerade beschrieben.

Alternativ können Sie aber auch einen Vergleich über *Reguläre Ausdrücke* (Ausführliche Einführung und Referenz, englisch <https://www.regular-expressions.info> , Wikipedia <https://de.wikipedia.org/wiki/Regulärer_Ausdruck>) aktivieren. Diesen Modus aktivieren Sie, wenn Sie in der Bewerterkonfiguration unter Key `Uebereinstimmung:` keine Zahl, sondern den Text `RegEx` (oder auch `rex`, `RegExP`, `regular expression`, Groß-/Kleinschreibung egal) notieren.

In dem Fall werden alle Ihre Vorgabestrings aus den `Wortliste:`-Properties als reguläre Ausdrücke interpretiert – sofern möglich.

Hinweise dazu:

- Auch wenn die Sprache der regulären Ausdrücke bis zu einem gewissen Grad einheitlich ist, gibt es doch verschiedene „Dialekte“ je nach Zielumgebung, die sich insbesondere in der Unterstützung einiger meist seltener gebrauchter Features/Erweiterungen unterscheiden können. Das Online-Übungssystem basiert derzeit auf Java 8, unterstützt also den entsprechenden „Java-8-Dialekt“ für reguläre Ausdrücke.
- Es wird empfohlen, zumindest komplexere reguläre Ausdrücke vor der Verwendung in der Aufgabe in einem geeigneten Tool zu testen.
    - Sehr gut, aber auch kostenpflichtig, ist z.B. RegexBuddy <https://www.regular-

expressions.info/regexbuddy.html> . Dort kann man insbesondere die Zielumgebung (den „Dialekt"), hier Java 8, einstellen, und das Tool umfasst sogar einen Debugger, um genauer analysieren zu können, warum bestimmte Eingaben von einem regulären Ausdruck (nicht) akzeptiert werden.
    - Es gibt auch verschiedene kostenlose Tools im Web, wie z.B. https://regexr.com <https://regexr.com> . Letzterer unterstützt nicht spezifisch den Java-„Dialekt" von regulären Ausdrücken, aber für typische, einfachere Ausdrücke, die keine „exotischen", nur in bestimmten „Dialekten" definierten Sprachelemente nutzen, ist das kein Problem.
- Sollte ein String aus Ihrer Wortliste *kein* syntaktisch korrekter regulärer Ausdruck sein, wird er als String-Literal interpretiert.
- Sollten Sie in Wortlisten reguläre Ausdrücke mit String-Literalen (einfach wie im Standard-Modus wörtlich zu vergleichenden Strings) mischen wollen, sind diese String-Literale in entsprechend „maskierte" reguläre Ausdrücke umzuwandeln.
    - Dazu können Sie *entweder* jedem Sonderzeichen, das in regulären Ausdrücken eine besondere Bedeutung hat (wie runde Klammern, Fragezeichen, Plus, Sternchen, Punkt, Dollar, Zirkumflex, Backslash etc.) jeweils einen Backslash (\) voranstellen. Z.B. der reguläre Ausdruck `Warum?` würde ein Muster beschreiben, auf das die beiden Eingaben `Waru` und `Warum` passen, denn das Fragezeichen bedeutet, dass das vorhergehende Zeichen optional ist, während `Warum\?` als regulärer Ausdruck genau die Eingabe `Warum?` akzeptiert.
    - Alternativ kann dem String-Literal ein `\Q` (wie Quote) vorangestellt und ein `\E` angehängt werden: Text zwischen `\Q` und `\E` wird als „Zitat" und nicht als regulärer Ausdruck interpretiert.
- **Reguläre Ausdrücke mit zusätzlichem Musterlösungs-Anzeigetext**: In der Regel (abhängig von den Schablonenoptionen, s.o.) werden die Einträge aus den `Wortliste:`-Angaben nicht nur als Vorgabe für die automatische Korrektur verwendet, sondern den Studierenden auch als Musterlösung angezeigt. Wenn Sie hier also reguläre Ausdrücke eintragen, werden im Standardfall auch die Studierenden diese regulären Ausdrücke als Musterlösung angezeigt bekommen. Bei einfachen Ausdrücken ist das vielleicht akzeptabel, komplexere reguläre Ausdrücke aber sind oft nicht gerade gut lesbar für Menschen. Deshalb können Sie optional zusätzlich zum regulären Ausdruck, der *alle* zu akzeptierenden Eingaben beschreibt, einen einfachen Musterlösungsstring zur Anzeige in der Autokorrektur festlegen, der vielleicht nicht alle akzeptierten Eingaben, sondern eine typische Musterschreibweise davon darstellt.
  Um beides (einen regulären Ausdruck und einen anzuzeigenden Musterlösungstext) in einer Wortliste zu notieren, fassen Sie den regulären Ausdruck zwischen zwei Querstriche ein (in Anlehnung an Sprachen wie JavaScript oder Perl, die Querstriche statt Anführungszeichen verwenden, um zwischen String-Literalen und Literalen für reguläre Ausdrücke zu unterscheiden) und notieren Sie dahinter die Beschriftung eingefasst in spitze Klammern (Kleiner-/Größer-Zeichen, angelehnt an die Syntax zum Anfügen eines Anzeigenamens hinter eine E-Mail-Adresse):
  `/` *regulärer Ausdruck* `/ <` *Anzeigetext* `>`. Es dürfen dabei beliebig viele Leerzeichen zwischen dem regulären Ausdruck und dem Anzeigetext (also zwischen `/` und `<`) stehen.
  Beispiel: `/(?i)write(ln)?\s*\(\s*(?-i)'Hello World!?'\s*\)(\s*;)?/ <write('Hello World!')>`
  Dieses Beispiel zeigt einen regulären Ausdruck, der eine Pascal-Anweisung zur Ausgabe des Textes »Hello World« oder »Hello World!« beschreibt, und zwar mit oder ohne anschließenden Zeilenvorschub (also wahlweise per `write`- oder `writeln`-Anweisung), mit oder ohne den Anweisungstrenner Semikolon nach der Anweisung sowie mit beliebig vielen Leerzeichen, Tabulatoren oder anderen „white spaces" zwischen der Anweisung, den Klammern, dem

String-Literal und dem Semikolon. Außerdem legt dieser reguläre Ausdruck auch fest, dass die Groß-/Kleinschreibung des Schlüsselworts `write` bzw `writeln` beliebig ist, die des String-Literals (`'Hello World'`) dagegen nicht, siehe folgenden Punkt:

- Sofern Ihr regulärer Ausdruck keine eigenen Steuerungen zur Groß-/Kleinschreibung trifft, gilt auch hier die durch den Bewertertyp gewählte Regelung, d.h. bei Verwendung von `Begriffe` z.B. wird die Groß-/Kleinschreibung auch beim Abgleich von Eingaben gegen reguläre Ausdrücke beachtet, bei Verwendung von `BegriffeIC` dagegen ignoriert. Es ist aber auch möglich, im regulären Ausdruck selbst mittels `(?i)` das Ignorieren der Groß-/Kleinschreibung für den nachfolgenden Teil des Ausdrucks zu aktivieren und per `(?-i)` wieder zu deaktivieren. Das ermöglicht es insbesondere, auch nur für *Teile* innerhalb eines Ausdrucks die Groß-/Kleinschreibung zu ignorieren bzw. zu beachten, wie im vorhergehenden Listenpunkt am Beispiel einer Pascal-Anweisung demonstriert.
  Solche „ausdrucksinternen" Regelungen überstimmen also die mit der Bewerterauswahl getroffene allgemeine Grundeinstellung.
- Bei Kombination von RegEx-Modus mit Smart-Modus (s.u.) gilt: Jede Vorgabe aus der Wortliste, die als Zahl interpretiert werden kann, wird auch numerisch mit den Eingaben verglichen, nur Vorgabe-Strings, die keine Zahlen sind, werden als reguläre Ausdrücke interpretiert.
- Bei Kombination mit Split-Modus (s.u.) wird empfohlen, *nicht* den Querstrich als Trennzeichen für die Aufzählung der Teilbegriffe zu verwenden, wenn Sie innerhalb der Teilbegriffe die obige Syntax für „beschriftete reguläre Ausdrücke" verwenden möchten. Das ist zwar möglich, aber dann müssten die Querstriche zum Einfassen der regulären Ausdrücke jeweils durch Voranstellen eines Backslashes (\) „escaped" werden, um nicht als Teilbegriffstrenner interpretiert zu werden.

Abschließend ein **Beispiel**:

```
Init:
 Aufgabentyp: BEGRIFFESMARTIC
 Name:  BegriffeSmartIC mit RegEx
 Punkte: 2
 Gewichtung: 1
 Felder: 2
 Uebereinstimmung: RegEx
 InputType: text
 Wortliste: /Zwei(-|\s*)und(-|\s*)vierzig/<Zweiundvierzig> | 42
 Wortliste: ]0;10]
```

Dieses Beispiel zeigt eine Konfiguration für eine Begriffsfrage, in der nach zwei Begriffen gefragt ist:

1. Eingabe der Zahl 42, und zwar…
   - wahlweise wirklich als Zahl, also z.B. als »42«, »42,0«, »0042.0«, »4.2e1« oder ähnlich,
   - oder als Wort, wobei Eingaben wie »Zweiundvierzig«, »Zwei-und-Vierzig« oder auch »Zwei und Vierzig« akzeptiert werden, jeweils in beliebiger Groß-/Kleinschreibung. Vor und nach dem »und« sind jeweils entweder ein Bindestrich oder beliebig viele White-Spaces wie Leerzeichen (und bei Wahl von 0 Leerzeichen entsprechend auch eine Zusammenschreibung) erlaubt. Das erlaubt allerdings auch Eingaben wie »Zwei-undvierzig« oder »Zwei und-vierzig«.
2. Eingabe einer Zahl > 0 und ≤ 10.

Der Bewertertyp mit Suffix `IC` stellt sicher, dass die Groß-/Kleinschreibung beim Textvergleich (zur

Zweiundvierzig) ignoriert wird, und `SMART` aktiviert die Zahlenerkennung, wie im Folgenden noch ausgeführt, so dass z.B. die zweite Wortliste `]0;10]` insbesondere weder ein Textliteral darstellt, das Studierende genauso eingeben müssten, noch einen regulären Ausdruck, mit dem eine Texteingabe verglichen würde, sondern vielmehr als Lösungsintervall interpretiert wird, aus dem eine numerische Eingabe erwartet wird.

Da hier nach zwei Begriffen in beliebiger Reihenfolge gefragt ist, und mindestens einer der gefragten Begriffe Text ist, sind für alle beiden Eingaben entsprechend Text-Inputs zu verwenden und hier `InputType: text` anzugeben. Diese Angabe ist optional, wenn sie weggelassen wird, gilt implizit die Annahme, dass die zu bewertenden Eingaben aus einem Textinput stammen.

Punkte: Gleichmäßige Punkteverteilung oder Einzelpunkte pro Begriff

Normalerweise geben Sie – wie oben unter Init-Blöcke allgemein beschrieben – genau eine erreichbare Punktzahl mit dem Schlüssel `Punkte:` an. Beachten Sie dabei, dass die hier angegebene Punktzahl *P* dann durch die Anzahl *n* der gefragten Begriffe (angegeben unter Schlüssel `Felder:`) teilbar sein muss. Für jeden korrekt genannten Begriff werden dann $\frac{1}{n}\,P$ Punkte vergeben, also „ein n-tel der erreichbaren Punkte".

(Falls Sie in den Kursparametern ein Punkteformat mit 1 bzw. 2 Nachkommastellen aktiviert haben, genügt es, wenn das 10- bzw. 100-fache der Punktzahl durch *n* teilbar ist. Falls Sie z.B. insgesamt nur 2 Punkte vergeben, aber nach 4 Begriffen gefragt ist, und das Punkteformat erlaubt mindestens eine Nachkommastelle, so ist das Zehnfache von 2 (20) durch 4 teilbar, jede korrekte Begriffsnennung wird dann entsprechend mit $\frac{20}{4}$ = 5 Zehntelpunkten = 0,5 Punkten bewertet.)

Alternativ können Sie aber auch zu jedem Begriff (jeder `Wortliste`) eine Einzelpunktzahl festlegen. Dazu erzeugen Sie im Init-Block der Begriffsfrage einfach genauso viele `Punkte:`-Einträge wie es `Wortliste:`-Einträge gibt. Dann bezieht sich die erste Punktzahl auf die erste Wortliste, die zweite Punktzahl auf die zweite Wortliste etc.
Aus Gründen der Übersichtlichkeit wird empfohlen, die Punktzahl immer hinter der Wortliste anzugeben, also z.B.:

```
Wortliste: Erster Begriff | Synonym zum ersten Begriff
Punkte: 1
Wortliste: Zweiter Begriff | Synonym zum zweiten Begriff
Punkte: 2
…
```

„Nicht-Ganz-Synonyme" mit abweichender Punktzahl

In einer Wortliste können Sie ja, wie bereits beschrieben, neben einem einzelnen gefragten „Hauptbegriff" auch noch mehrere alternative Begriffe als Synonyme festlegen. Wenn man das Wort Synonym wörtlich nimmt, haben diese Begriffe dieselbe Bedeutung und sollten daher in der Regel auch mit derselben Punktzahl bewertet werden.

Es kann aber auch sinnvoll sein, an Stelle eines mit voller (Teil-)Punktzahl bewerteten Hauptbegriffs einen alternativen Begriff zwar zu akzeptieren, der nicht ganz gleichwertig ist, und diesen dafür mit einer geringeren (Teil-)Punktzahl zu bewerten (also gegenüber dem Musterbegriff/Hauptbegriff einen gewissen Punktabzug vorzunehmen).

Wenn Sie die oben vorgestellte Möglichkeit nutzen, die Punktzahlen pro Wortliste separat festzulegen (statt einer Gesamtpunktzahl, die auf alle gefragten Begriffe zu verteilen ist), dann können Sie dort alternativ auch jedem Wort einer Wortliste (sprich: dem Hauptbegriff und jedem

seiner „Synonyme“) eine separate Einzelpunktzahl zuordnen.
Hierzu wird wieder dieselbe Schreibweise verwendet, wie sie auch schon bei anderen Fragetypen mit Support für unterschiedliche Punktzahlen zu unterschiedlichen Lösungsvorgaben unterstützt wird: Geben Sie hinter `Punkte:` eine Liste von Punktzahlen an, getrennt per Semikolon.

- Das sollte möglichst eine Punktzahl pro Wort in der zugehörigen Wortliste sein.
- Ist die Punkteliste länger als die Wortliste, werden die „überschüssigen“ Punktzahlen ignoriert.
- Ist die *Punkteliste kürzer* als die Wortliste, wird für jedes nachfolgende Synonym ohne „eigene“ Punktzahl dieselbe Punktzahl wie für den Hauptbegriff (erste Punktzahl in der Liste) vergeben. Enthält die Wortliste z.B. drei Wörter und die zugehörige Punkteliste nur zwei Punktzahlen, so gilt die zweite Punktzahl für den zweiten Begriff (erstes Synonym), die erste Punktzahl für den ersten und dritten Begriff („Hauptbegriff“ und zweites Synonym).

Beispiel:

```
…
Felder: 2
…
Wortliste: Hallo | Allo | Hello
Punkte: 2; 1
Wortliste: Welt | Monde | World
Punkte: 5; 3; 4
```

In diesem Hello-World-Beispiel sei nach zwei Begriffen gefragt, einmal nach dem Wort „Hallo“ (oder „Hello“ oder „Allo“) und einmal nach „Welt“ (oder „World“ oder „Monde“). Die volle Punktzahl gibt es dabei jeweils für die deutschen Bezeichnungen (2 Punkte für „Hallo“ und 5 Punkte für „Welt“). Für „Allo“ statt „Hallo“ gibt es nur einen Punkt, für „Hello“ wieder 2 Punkte (da keine individuelle Angabe erfolgt ist). Für „Monde“ statt „Welt“ gibt es 3 statt 5 Punkten und für „World“ immerhin 4 statt 5 Punkten.

Die maximal erreichbare Punktzahl der Begriffsfrage ist die Summe der Maxima aller Punktlisten, hier also 2 + 5 = 7 Punkte.

In der Regel sollte die erste Punktzahl einer Liste die größte sein, der „Hauptbegriff“ (erste Begriff der Wortliste) also die Bestpunktzahl für diesen Begriff erzielen, während Synonyme, wenn sie überhaupt abweichend bewertet werden sollen, dann höchstens mit einer *kleineren* Punktzahl!
Ein Grund dafür: In der Muster-Anzeige des Bewerters wird in der Regel immer nur der Hauptbegriff mit der erreichbaren Punktzahl genannt, ein Synonym dazu wird lediglich dann mit aufgeführt, falls der Studierende genau dieses genannt hat. Hat nun ein Student den Hauptbegriff genannt und ist dieser nicht die Maximalpunktzahl wert, wird ihm in der Muster-Ausgabe also auch nur dieser Hauptbegriff mit der damit erreichbaren Punktzahl angezeigt, aber es wird nicht verraten, mit welchem Begriff die Bestpunktzahl erreichbar gewesen wäre. Das kann vielleicht in Ausnahmefällen erwünscht sein (z.B. Bonuspunkte-Vergabe für gewisse Synonyme), aber im Regelfall sollte das vermieden werden und eben, wie gesagt, der erstgenannte und in der Mustervorgabe immer angezeigte Begriff auch die Bestpunktzahl wert sein.

Smart-Modus

Im Normalfall (`Begriffe` oder `BegriffeIC`) werden die studentischen Eingaben mit Ihren vorgegebenen Begriffen „wörtlich“ als Text verglichen, ggf. unter Erlaubnis von gewissen Abweichungen wie Groß-/Kleinschreibung (`BegriffeIC`) oder durch Festlegen einer `Uebereinstimmung` von unter 100%.

Der Smart-Modus erlaubt zwei Abweichungen von diesem wörtlichen Textvergleich:

1. Zahlenvergleiche analog zu Bewertertyp `Zahlen`, aber innerhalb einer Begriffsfrage
2. Wildcard (`*`)-Vorgabe, um eine Frage aus der Wertung zu nehmen und alle Einsendungen mit voller Punktzahl zu bewerten.

Zahlen als Begriffe

Im Smart-Modus erkennt der Bewerter Zahlen als solche. D.h. wenn einer oder mehrere Ihrer Begriffe sich als Zahl interpretieren lassen, und auch ein Benutzer eine Eingabe vornimmt, die sich ebenfalls als Zahl interpretieren lässt, werden diese nicht per Text-Vergleich, sondern numerisch verglichen. Haben Sie z.B. den Begriff „42“ vorgegeben, so werden im Smart-Modus auch studentische Eingaben wie „42,00“, „42.0“, „+42“ oder „4.2e+1“ als korrekte Eingaben dieses Begriffs interpretiert (im Gegensatz zur Eingabe „42.000“, die als 42000 statt als 42 interpretiert wird). Im Text-Modus dagegen müsste eine Eingabe dann exakt „42“ lauten, um als korrekt erkannt zu werden.

Es wird (sowohl in Ihrer Angabe der Zahl in der `Wortliste` als auch für Eingaben von Studenten) sowohl die europäische Schreibweise mit Komma als auch die amerikanische mit Punkt als Dezimaltrenner erkannt. Ein Punkt als Tausendertrenner ist ebenfalls erlaubt, wenn er nicht mit einem Dezimaltrenner verwechselt werden kann. Grenzfall: In einer Zahleneingabe, die auf „.000“ endet (und ansonsten weder Punkt noch Komma enthält) wird der Punkt als Tausendertrenner interpretiert, weil dies die wahrscheinliche Interpretation ist (eine Eingabe auf „Komma-Null-Null-Null“ mit genau drei Nullen enden zu lassen, ist unüblich). Andernfalls wird ein einzelner Punkt als Dezimaltrenner interpretiert.

Der Wert `Uebereinstimmung` wird nur bei Text-Vergleichen verwendet, bei Zahlenvergleichen spielt er keine Rolle.

Zahlenintervalle

Wie beim Bewertertyp `Zahlen` können Sie (im Smart-Modus) nicht nur nach konkreten Zahlen fragen, sondern auch nach Zahlen aus einem Lösungsintervall. Geben Sie in diesem Fall ein Intervall als erwarteten Begriff in der `Wortliste`-Einstellung nach folgender Syntax ein: `[von;bis]`. Dabei sind „von“ und „bis“ durch die Intervallgrenzen (Zahlen) zu ersetzen, die bei dieser „Klammerstellung“ selbst noch als Teil des Intervalls gelten. Wenn die Grenze nicht mehr zum Intervall gehören soll, kehren Sie die jeweilige Klammer um, so dass sie „nach außen zeigt“. Beispiel: Zum „Begriff“ `[0; 11,2[` werden im Smart-Modus alle Zahlen-Eingaben ≥ 0 und < 11,2 akzeptiert.

Als Trennzeichen zwischen den Zahlen ist ein `;` oder ein `/` erlaubt. Leerzeichen zwischen Klammern, Zahlen und Trennzeichen sind zulässig.

„Gemischte“ Wortlisten

Im »Smart«-Modus können Sie Vorgaben aus Text und Zahlen oder Intervallen beliebig mischen. So können Sie z.B. auch eine Zahl als Synomym für einen Text (oder umgekehrt) festlegen.

Beispiel: Sie könnten als Begriff die Zahl „42“ und als Synonym den Text „Zweiundvierzig“ akzeptieren:

```
Wortliste: 42 | Zweiundvierzig
```

Zu diesem Begriff würden nun die Eingaben „Zweiundvierzig“ und „42“, aber (im Smart-Modus) auch z.B. „42,0“ oder „+42“ als korrekte Antworten gewertet.

Frage aus der Wertung nehmen

Falls Sie eine Begriffsfrage nachträglich aus der Wertung nehmen wollen, ersetzen Sie *jeden* Ihrer Begriffe durch ein `*` ohne Synonyme (also: `Wortliste: *`, *m*-mal wiederholt bei *m* gefragten Begriffen/Eingabefeldern) und aktivieren Sie den »Smart«-Modus. Dann wird jede Eingabe mit voller Punktzahl bewertet.

Split-Modi

Normalerweise wird zu jedem gefragten Begriff ein Eingabefeld in der Aufgabenseite vorgesehen und in einer `Wortliste` stehen ein Begriff und ggf. seine Synonyme, die mit den Eingaben aus dem jeweiligen Eingabefeld entweder immer lexikalisch oder im Smart-Mode auch ggf. erweitert (z.B. als Zahl interpretiert) verglichen werden. In diesem Fall besteht also jeder Begriff aus zwar nicht zwangsläufig genau einem Wort, aber zumindest aus einer immer zusammenhängenden Zeichenkette. So kann es z.B. sein, dass Sie als Begriff den Namen einer Person erwarten, bestehend aus Vor- und Nachname. Theoretisch können Sie nun verschiedene Schreibweisen (darunter auch die Schreibweisen „Vorname Nachname“ sowie „Nachname, Vorname“, ggf. Varianten mit Initial etc.) als Synonyme festlegen.

Es ist aber auch möglich, gefragte Begriffe in mehrere Teilbegriffe zu zerlegen und ein Eingabefeld pro Teilbegriff vorzusehen. (Anders als bei den nachfolgenden *Begriffsfolge*-Fragetypen müssen sich dann aber alle gefragten Begriffe in exakt gleich viele Teilbegriffe zerlegen lassen, da ja die Reihenfolge der Begriffseingabe beliebig ist, also keine Eingabefelder einem festen Begriff zugeordnet sind, sondern jede Folge von Eingabefeldern für Teilbegriffe zur Eingabe jedes der gefragten Begriffe geeignet sein muss.)

Um das obige Beispiel mit Vor- und Nachnamen von Personen aufzugreifen: Sie könnten z.B., wenn Sie nach den Namen mehrerer Personen fragen, jeweils *zwei* Namenseingabefelder pro Person vorsehen, wobei es egal ist, in welches der beiden Felder jeweils der Vor- oder der Nachname eingetragen werden soll. Beide Teilbegriffe zusammen bilden den Gesamtbegriff (hier den Namen einer Person).

Genauer haben Sie bei der Zerlegung von Begriffen in Teilbegriffe zwei Optionen:

1. **Split-Teilmengen-Modus**: Analog dazu, wie dieser Begriffe-Fragentyp dazu ausgelegt ist, nach einer Teilmenge vorgegebener Begriffe zu fragen, die in beliebiger Reihenfolge eingegeben werden dürfen, kann auch nach einer *Teilmenge* der Teilbegriffe gefragt werden, wobei die Reihenfolge der Teilbegriffseingabe (z.B. Vor-/Nachname) egal ist. Dabei kann (wie beim Beispiel von Vor- und Nachname) nach *allen* Teilbegriffen oder (bei weniger Eingabefeldern als Teilbegriffen in der Musterlösung) auch nach einer *echten* Teilmenge gefragt werden.
   In diesem Fall (jedes Teilwort kann in jedes Input-Feld zu jeder Begriffseingabe gleichermaßen eingetragen werden) sollten alle Inputs gleichartig eingestellt sein. Daher ist auch unter `InputType`, wenn angegeben, nur eine einzige Einstellung (`text` oder `number`) vorgesehen.
2. **Split-Sequenz-Modus**: Analog zu Begriffsfolge-Fragen (s.u.) ist es auch möglich, dass Nutzer zu jedem gefragten Begriff stets alle seine Teilbegriffe in einer bestimmten Reihenfolge eingeben müssen, z.B. immer erst den Vor- und dann den Nachnamen einer Person, und dass Sie hierfür eben lediglich getrennte Eingabefelder vorsehen möchten (z.B. in einer Tabellenanordnung).
   Hier sollte zwar auch für jedes Begriffseingabe die Abfolge von Inputs für die Teilwortsequenzen jeweils gleichartig sein, *innerhalb* der Teilworteingabe können sich die einzelnen Eingabefelder aber durchaus unterscheiden. Z.B. könnte jeder gefragte Begriff aus

einer Sequenz zweiter Texteingaben und einer Zahleneingabe bestehen und für die Zahleneingabe jeweils ein `<input type="number">` verwendet werden. Daher ist in diesem Fall vorgesehen, unter `InputType` eine Aufzählung von Inputtypen anzugeben, im gerade genannten Beispiel also `InputType: text, text, number`.

Bei reinen Texteingaben ist zumindest der Sequenzmodus von eingeschränktem Nutzen, er könnte i.W. höchstens Vorzüge im Formulardesign (z.B. Tabellenanordnung von Eingabefeldern) haben. Ihre wahre Stärke spielt die Zerlegung von Begriffen in Teilbegriffe für numerische Eingaben im Smart-Modus (s.o.) aus.
So könnte z.B. in einer Buchhaltungsaufgabe nach mehreren Buchungssätzen gefragt sein. Die Reihenfolge der Buchungen spielt keine Rolle, nur ihre Gesamtmenge, weshalb der `Begriffe`-Fragetyp hierfür schonmal gut geeignet ist. Ein Buchungssatz besteht aber aus mehreren Zahlen (z.B. Habenkonto, Sollkonto und Buchungsbetrag), und für Zahlen wiederum bietet sich der Smart-Modus an, damit verschiedene Schreibweisen derselben Zahl (insbesondere beim Betrag) gewertet werden können, ohne unzählige Synonyme festlegen zu müssen. Das Problem ist aber, dass der Smart-Modus im Standardfall nur für die gesamte Eingabe funktioniert, also erwarten würde, dass jeder Begriff *eine* Zahl ist und nicht eine Abfolge mehrerer Zahlen und ggf. auch noch Texte dazu. In diesem Szenario hilft es, die Begriffe (Buchungssätze) in Teilbegriffe zu zerlegen, die jeweils Zahlen sein können. Ein Buchungssatz wäre also z.B. mindestens eine Folge von drei Zahlen (Sollkonto, Habenkonto, Betrag), die in diesem Fall im zweiten oben genannten Modus (Sequenzmodus/Teilbegriffsfolge) ausgewertet werden müssen.

Um diesen Modus zu nutzen, sind zunächst im Aufgabenformular entsprechend mehrere Input-Elemente als Eingabefelder für die einzelnen Teilbegriffe anzulegen. Eine Tabellenanordnung ergäbe hier oft Sinn (beim Buchungssatz-Beispiel mit den Spalten Haben, Soll und Betrag und entsprechenden Eingabefeldern in den Tabellenzellen). Sie erzeugen dazu pro Begriff (Buchungssatz) mehrere Input-Elemente, die alle denselben Feldnamen aufweisen, und ordnen dem ersten davon im Attribut `data-separator` ein Trennzeichen zu, mit dem die einzelnen Teileingaben dann in der Gesamteinsendung (String) aufgezählt gespeichert werden. (Siehe Abschnitt Mehrere gleichnamige Eingabefelder, `data-separator`-Attribut für Details hierzu.)

Dann geben Sie hier in der Bewerterkonfiguration entsprechend in den `Wortliste`-Einstellungen jeden Begriff als Folge von Teilbegriffen an, die untereinander ebenfalls durch ein von Ihnen gewähltes Trennzeichen separiert sind, z.B. durch Querstrich `/` oder Semikolon `;`. Es ist sicherlich übersichtlicher, aber nicht notwendig, für beides (Input-Felder und Bewerter-Wortlisten) dasselbe Trennzeichen zu verwenden.

Als nächstes müssen Sie dem Bewerter noch mitteilen, dass diese Wortliste-Einträge aus Teilbegriffen bestehen und welches Trennzeichen Sie dort verwendet haben. Das geschieht über die Property `Wortliste-Separator:`.

Beispiel für Buchungssätze:

```
Init:
 Aufgabentyp: BEGRIFFESMART
 Name:  Buchungssatz-Demo
 Punkte: 10
 Gewichtung: 1
 Felder: 2
 Uebereinstimmung: 100
 InputType: number, number, number
 Wortliste-Separator: /
 Wortliste: 110 / 300 / 42,42  | 300 / 110 / -42,42
 Wortliste: 112 / 300 / ]8;10] | 300 / 112 / [-10;-8[
```

Dieses Beispiel geht von drei Eingabefeldern pro Buchungssatz aus (und zwar alle drei vom Typ `<input type="number"...>`, wobei diese sich in Details wie Support von Nachkommastellen und negativen Zahlen für Beträge vs. nur positive ganze Zahlen für Kontonummern unterscheiden können, was aber dem Bewerter hier egal ist), und dass nach genau zwei von zwei Buchungssätzen gefragt ist (die aber in beliebiger Reihenfolge genannt werden dürfen). Ein Buchungssatz wird in den Wortlisten durch zwei ganze Zahlen und einen Betrag (oder ein Lösungsintervall), jeweils getrennt per Querstrich, definiert. Das Beispiel zeigt, dass auch hier Synonyme definiert werden können, hier, indem eine Vertauschung von Soll- und Habenkonto erlaubt wird, wenn dafür der negative Betrag verwendet wird. (Ob das in der Aufgabe sinnvoll ist, sei mal dahingestellt, es geht ja nur um eine „Technologiedemo".) Die zweite Wortliste demonstriert dabei, dass statt eines festen Betrages auch ein Lösungsintervall als Betrag definiert werden kann (wie oben unter Smart-Modus beschrieben).

In diesem Fall wird dabei der *Split-Sequenz-Modus* verwendet, d.h. die Studierenden müssen die jeweiligen drei Zahlen in der vorgegebenen Reihenfolge eingeben, damit die Eingabe als korrekter Buchungssatz erkannt wird. (Punkte gibt es wie im normalen Modus immer nur für jeden korrekt genannten Begriff, d.h. es gibt keine Teilpunkte für Eingaben, bei denen lediglich einzelne, aber nicht alle Teilbegriffe korrekt eingegeben wurden.)

Um den *Split-Teilmengen-Modus* zu verwenden, muss hinter jeder `Wortliste`-Property immer *direkt* eine `Teilwortanzahl`-Property folgen, die angibt, *wieviele* der Teilworte eines Wortes aus der Wortliste jeweils gefragt sind. Dabei muss der Wert beim Begriffe-Bewerter für alle Wortlisten immer gleich sein! (Dass die Angabe hinter *jeder* Wortliste redundant stehen muss, hängt damit zusammen, dass die Konfigurationssytax bei Begriffe- und den im folgenden Abschnitt beschriebenen Begriffsfolge-Bewertern identisch ist, und bei Begriffsfolgen tatsächlich zu jeder Wortliste hier eine abweichende Angabe möglich ist.)
Der Wert von `Teilwortanzahl` muss dabei übereinstimmen mit der Anzahl der gleichnamigen Eingabefelder pro Begriff.

Wenn also z.B. nach insgesamt 3 Begriffen gefragt sein sollte, zu denen jeweils 2 Teilbegriffe eingegeben werden sollen, so sind insgesamt 3 × 2 Eingabefelder (je zwei gleichnamige Felder, also z.B. zwei Felder namens `FeldA1`, zwei Felder namens `FeldA2` und zwei namens `FeldA3` o.ä.) im Formular anzuordnen, z.B. in Tabellen-/Matrixform. Die Property `Felder` müsste dann den Wert 3 bekommen, und es sind dann *mindestens* 3 Wortlisten anzulegen, von denen *jede* um die Angabe `Teilwortanzahl: 2` zu ergänzen wäre.

Als Beispiel für den Teilmengen-Modus soll noch einmal das obige Beispiel einer Begriffe-Frage wieder aufgegriffen werden, in der nach Namen gefragt ist, wobei zu jedem Namen zwei Eingabefelder für getrennte Eingabe von Vor- und Nachnamen (in beliebiger Reihenfolge) bereitstehen sollen. Die Konfiguration dazu könnte z.B. wie folgt aussehen:

```
Init:
 Aufgabentyp: BEGRIFFEIC
 Name:  Test Teilbegriffe
 Gewichtung: 1
 Felder: 2
 Uebereinstimmung: 75
 Wortliste-Separator: ,
 Wortliste: Kästner,Erich
 Teilwortanzahl: 2
 Punkte: 3
 Wortliste: Tischbein,Emil
 Teilwortanzahl: 2
 Punkte: 2
 …
```

In diesem Beispiel ist nicht nur `Teilwortanzahl: 2` eingetragen, sondern unter `Wortliste` stehen dazu jeweils auch genau zwei Teilworte (Vor- und Nachname), d.h. es wird nach *allen* Teilworten gefragt, die `Teilwortanzahl`-Property legt lediglich fest, dass diese beiden Namen in beliebiger Reihenfolge eingegeben werden dürfen (während bei Weglassen dieser Property der Split-Sequenz-Modus aktiv wäre).
Falls die Anzahl der unter `Wortliste` aufgezählten Teilwörter dagegen größer als der Wert `Teilwortanzahl` ist, dann wird entsprechend nach einer echten Teilmenge dieser Teilwörter gefragt. Zählen Sie in jeder Wortliste z.B. (hier kommasepariert) 4 Teilbegriffe auf und legen dazu `Teilwortanzahl: 2` fest, so sind eben zwei der vier Teilbegriffe von einem Studenten zu nennen.

Obiges Beispiel demonstriert auch die korrekte Kombination von Split-Teilmengen-Modus mit unterschiedlicher Punktzahlvergabe für verschiedene Begriffe (hier: Namen) über separate `Punkte`-Properties: Wichtig ist hier, dass – wie gesagt – `Teilwortanzahl` jeweils *direkt* auf `Wortliste` folgen muss, die `Punkte`-Property zu einer Wortliste also nicht zwischen `Wortliste` und `Teilwortanzahl` stehen darf.

Abschließend noch ein Wort zum „Escaping“:
Normalerweise können Sie dies durch sinnvolle Wahl des `Wortliste-Separator`-Trennzeichens ohnehin vermeiden, aber falls Sie ein Trennzeichen wählen, das in einem oder mehreren Begriffen auch als Bestandteil vorkommt, dann können Sie letztere Vorkommen durch Voranstellen eines Backslash (`\`) maskieren. (Ausnahme: Sollte der Separator selbst ein Backslash sein, ist ein Slash (`/`) als Escape-Zeichen zu verwenden.) Jedes Vorkommen eines Backslashes (bzw. Slashes, s.o.) in einer Wortliste ist (nur im Split-Mode, also nur, wenn `Wortliste-Separator` definiert wurde) ebenfalls zu maskieren, effektiv also zu verdoppeln. Sollte ein Begriff komplett auf einem oder mehreren Backslashes enden und darauf ein Pipe-Symbol zur Einleitung eines Synonyms folgen, schreiben Sie ein Leerzeichen vor das Pipe-Symbol, damit dieses nicht selbst durch den Backslash maskiert wird.

Typen `Begriffsfolge`, `BegriffsfolgeIC`, `BegriffsfolgeSmart` und `BegriffsfolgeSmartIC`

Während beim Typ `Begriffe` die Nennung von *m* aus *N* Begriffen, also einer (Teil-)*Menge* von vorgegebenen Begriffen ohne vorgegebene Reihenfolge, gefragt ist, beschreibt der Typ `Begriffsfolge` die Frage nach einer *Aufzählung/Folge* von *N* Begriffen, d.h. die Nennungen müssen in einer bestimmten Reihenfolge erfolgen, und Mehrfachnennungen sind möglich.

Ein typischer Anwendungsfall für diesen Bewerter ist eine **Lückentext**-Aufgabe: In einem solchen Text seien N Textlücken vom Studenten jeweils durch eine Eingabe (Begriff) auszufüllen. Hierbei muss natürlich der richtige Begriff in die korrekte Lücke eingefügt werden, d.h. die Reihenfolge der Eingaben / die Verteilung der Eingaben auf die Textlücken muss stimmen, und es kann durchaus sein dass in mehrere Lücken derselbe Begriff eingetragen werden muss.

Die Begriffsfolge-Bewerter arbeiten weitgehend analog zu den oben vorgestellten Begriffe-Bewertern, nur dass sie eben keine Teilmenge einer vorgegebenen Begriffsfolge abfragen, sondern exakt diese Begriffsfolge.

Die **Konfigurationsoptionen** sind weitgehend dieselben wie bei Typ `Begriffe`, Insbesondere gibt der Aufgabenautor wieder für jeden Begriff (z.B. jede Textlücke in einem Lückentext) eine `Wortliste` vor, die aus mindestens einem Wort (dem in der Lücke erwarteten Begriff) besteht, aber auch (durch `|` getrennt) Synonyme aufzählen kann, also weitere im selben Eingabefeld (z.B. derselben Textlücke) ebenfalls als korrekt zu akzeptierende Begriffe.

Im Vergleich zu den `Begriffe`-Optionen gibt es jedoch folgende Abweichungen:

1. Die Angabe `Felder` entfällt: Anders als beim Begriffe-Fragetyp, wo *m* aus *N* Begriffe gefragt waren und daher eine Anzahl von *m* Eingabefeldern ins Formular einzufügen war und diese Zahl *m* dem Bewerter in der Einstellung `Felder` mitzuteilen war, wird bei *diesem* Fragetyp dagegen immer nach der kompletten Folge von *N* Begriffen gefragt. D.h. der Aufgabenautor muss immer genauso viele Eingabefelder vorsehen, wie er `Wortliste`n in der Bewertereinstellung angibt, und ein Bedarf für eine gesonderte Einstellung `Felder` besteht hier nicht.
2. Die (optionale) Angabe `InputType` ist hier nicht nur einmalig / global für den gesamten Lückentext anzugeben, sondern separat für jede `Wortliste`. Denn anders als beim Begriffe-Fragetyp, wo jeder Begriff in jedes der Eingabefelder eingetragen werden kann und daher alle Begriffs-Eingabefelder gleichartig sein sollten, kann beim Lückentext jede Textlücke einen anderen Input-Typ verwenden.
   Wenn Sie Number-Inputs verwenden und dazu die `InputType`-Angabe benötigen, sollten also genau so viele `InputType`-Keys wie `Wortliste`-Keys in der Konfiguration vorkommen, wobei der i-te `InputType`-Eintrag eben angibt, ob das Eingabefeld zur i-ten Wortliste ein `<input type="number">` ist oder nicht. (Für Selectboxen ist hier auch `text` anzugeben.)
   Der Übersichtlichkeit halber wird empfohlen, die `InputType`-Angabe zu einer Textlücke/einem Eingabefeld jeweils unter der zugehörigen `Wortliste`-Angabe anzugeben, also `Wortliste` und `InputType` immer abwechselnd einzutragen. Notwendig ist das aber nicht, es könnten z.B. auch zuerst alle `Wortliste`-Zeilen notiert werden und im Anschluss eine Folge genauso vieler `InputType`-Zeilen (oder umgekehrt).

Beispiel:

Zu einem Lückentext der Art *»Die „FernUniversität in ____ " liegt in der Stadt ____ im Bundesland ____ der ____ .«* könnte die Bewerterkonfiguration wie folgt aussehen:

```
Init:
      Aufgabentyp: Begriffsfolge
      Name: Lückentext
      Punkte: 30
      Gewichtung: 1
      Uebereinstimmung: 90
      Wortliste:  Hagen
      Wortliste:  Hagen
      Wortliste:  Nordrhein-Westfalen|NRW
      Wortliste:  Bundesrepublik Deutschland|BRD
```

Analog zum `Begriffe`-Bewerter gibt es vom `Begriffsfolge`-Typ wieder eine `BegriffsfolgeIC`-Variante, welche die Groß-/Kleinschreibung der Benutzereingaben ignoriert, und per `BegriffsfolgeSmart` (bzw. `BegriffsfolgeSmartIC`) kann zusätzlich wieder der Smart-Modus aktiviert werden.

*Die meisten Einstellungen gelten ganz **analog zum Begriffe-Bewerter** und werden hier daher nicht erneut erläutert.* Die folgenden Abschnitte gehen im Wesentlichen auf Punkte ein, in denen es Unterschiede zum Begriffe-Bewerter gibt.

Punkte: Gleichmäßige Punkteverteilung oder Einzelpunkte pro Begriff

Normalerweise geben Sie – wie oben unter Init-Blöcke allgemein beschrieben – genau eine erreichbare Punktzahl mit dem Schlüssel `Punkte` an. Beachten Sie dabei, dass – zumindest im Standardfall, dass Ihre Kursumgebung nur ganze Punkte unterstützt – die hier angegebene Punktzahl *P* dann durch die Anzahl *n* der gefragten Begriffe/Textlücken (= Anzahl der `Wortliste`n) teilbar sein muss. Für jeden korrekt genannten Begriff / jede korrekt ausgefüllte Textlücke werden dann $\frac{1}{n}\,P$ Punkte vergeben, also „ein n-tel der erreichbaren Punkte“.

(Auch hier gilt, analog zu Typ `Begriffe` oben, dass bei Aktivierung eines Punkteformats mit einer oder zwei Nachkommastellen entsprechend „nur“ das Zehn- oder Hundertfache der Punkte durch *n* teilbar sein muss, so dass sich $\frac{1}{n}\,P$ noch ohne Rundung mit der gewählten Nachkommastellenzahl darstellen lässt.)

Alternativ können Sie aber auch zu jedem Begriff (jeder Textlücke/`Wortliste`) eine Einzelpunktzahl festlegen, indem Sie genauso viele `Punkte`- wie `Wortliste`-Einträge erzeugen. Siehe Bewertertyp `Begriffe` oben für mehr Details und ein Beispiel.

Smart-Modus

Der Smart-Modus des Begriffsfolge-Bewerbers entspricht weitgehend dem des Begriffe-Bewerters, siehe oben.

In erster Linie ermöglicht er also das numerische Vergleichen von Zahleneingaben mit in der `Wortliste` stehenden Zahlen oder Intervallen, während der Standardmodus immer nur einen Textvergleich vornimmt. Schreiben Sie z.B. einen Lückentext, in dem in einer oder mehreren Textlücken eine bestimmte Zahl oder eine Zahl aus einem bestimmten Lösungsintervall (analog zum Bewertertyp `Zahlen`) erwartet wird, aktivieren Sie den Smart-Modus und geben Sie in der Wortliste die erwartete Zahl bzw. das Lösungsintervall ein. (Siehe obigen Abschnitt zum `BegriffeSmart`-Typ für mehr Details hierzu.)

Einzelne oder alle Begriffe/Textlücken aus der Wertung nehmen

Anders als beim `BegriffeSmart`-Bewerter können Sie bei einer *Begriffsfolge* nicht nur die gesamte Frage, sondern auch einzelne Begriffe/Textlücken aus der Wertung nehmen, indem Sie die Wortliste-Vorgabe für den enstprechenden Begriff durch `Wortliste: *` (ohne Synonyme) ersetzen. Im Smart-Modus bedeutet dieses Sternchen, dass an dieser Stelle (also z.B. in der zugehörigen Textlücke) beliebige Eingaben erlaubt sind, also die entsprechenden Punkte immer vergeben werden sollen.
Um die gesamte Frage (z.B. den gesamten Lückentext) aus der Wertung zu nehmen, sind entsprechend *alle* Wortlisten durch `*` zu ersetzen.

Beispiel:

Die folgende Beispielkonfiguration zeigt die Frage nach einer Begriffsfolge von 4 Begriffen, wobei als Begriff 1 „Hallo“ oder „Hello“ genannt werden muss, als Begriff 2 „Welt“ oder „World“. Die Groß-/Kleinschreibung wird jeweils nicht beachtet (da ein Bewertertyp mit `IC`-Suffix gewählt wurde). Die Eingabe zum dritten Begriff ist beliebig, jede Eingabe (auch gar keine Eingabe in diesem Feld) wird als korrekt bewertet und mit 10 Punkten bewertet (d.h. jeder Aufgabenteilnehmer hat diese 10 Punkte sicher und kann lediglich mit den Begriffen 1, 2 und 4 noch jeweils 10 weitere Punkte „hinzuverdienen“). Als Begriff Nr. 4 ist eine beliebige Zahl ≥ 0 und < 10 einzugeben.

```
Init:
      Aufgabentyp: BegriffsfolgeSmartIC
      Name: Lückentext
      Punkte: 40
      Gewichtung: 1
      Uebereinstimmung: 100
      Wortliste: Hallo|Hello
      InputType: text
      Wortliste: Welt|World
      InputType: text
      Wortliste: *
      InputType: text
      Wortliste: [0;10[
      InputType: number
```

Split-Modi

Auch für Begriffsfolge-Fragen / Lückentexte kann wieder eine Zerlegung der einzelnen Begriffe in Teilbegriffe genutzt werden. Wie schon oben im gleichnamigen Unterabschnitt von Typen `Begriffe`, `BegriffeIC`, `BegriffeSmart` und `BegriffeSmartIC` beschrieben, haben Sie hier auch wieder zwei verschiedene Split-Modi zur Auswahl:

1. **Split-Sequenz-Modus**: Hier müssen auch die Teile eines Begriffs immer vollständig und in derselben Reihenfolge eingegeben werden. Das hat auf den ersten Blick keinen besonderen Vorteil gegenüber dem Standardvorgehen, die Teilbegriffe einfach direkt zu vollwertigen Begriffen zu erklären. Einen Unterschied macht es aber bei der Bewertung, weil im Normalbetrieb jeder korrekt genannte Begriff eine bestimmte Punktzahl erhält, in diesem Fall aber immer nur für einen korrekt ausgefüllten Gesamtbegriff Punkte vergeben werden, während z.B. einer von zwei korrekten Teilbegriffen keine Punkte erhält. (Haben Sie z.B. je ein Feld für den Vor- und den Nachnamen einer Person vorgesehen und diese als Teilfelder *eines* Begriffs vorgesehen, dann erhält ein Teilnehmer nur Punkte für einen vollständig korrekt genannten Namen, also wenn er sowohl Vor- als auch Nachname korrekt genannt hat, aber keine Teilpunkte, wenn nur einer der beiden Namensteile stimmt.)
   Wenn Sie (ggf. teilweise) Number-Inputs verwenden und daher die Option `InputType` zu

jedem zusammengesetzten Begriff angeben, ist darin jeweils (weil ja jeder Begriff seinerseits noch aus Teilbegriffen besteht, also mehr als ein Input-Element pro Begriff/Wortliste existiert, und im Split-Sequenz-Modus jedes dieser Input-Elemente auch unterschiedlich konfiguriert sein kann) wieder eine *kommaseparierte Aufzählung* von Worten `text` oder `number` anzugeben, ein Wert pro Teilbegriff/Eingabeelement. Das ist analog zum Begriffe-Bewerter oben, nur dass dies hier eben für jeden Gesamtbegriff / jede Wortliste separat anzugeben ist.

2. **Split-Teilmengen-Modus**: Hiermit ermöglichen Sie, dass die Teilbegriffe eines Begriffs in ihre jeweiligen Eingabefelder in beliebiger Reihenfolge eingegeben werden dürfen, wobei auch nur nach einer *echten* Teilmenge der Teilbegriffe gefragt sein kann. So können Sie z.B. in einem Lückentext zwei Eingabefelder als Textlücken vorsehen, diese als Teileingabefelder zu *einem* Begriff im Split-Teilmengen-Modus konfigurieren und in der Wortliste dazu z.B. vier verschiedene korrekte Teilbegriffe dazu definieren. Dann muss ein/-e Teilnehmer/-in eben zwei der vier Teilbegriffe in beliebiger Reihenfolge in die beigen Eingabefelder eintragen, um den Punkt für diesen Gesamtbegriff zu erhalten.
   Falls Sie hier die `InputType`-Angaben verwenden, gilt wieder (analog zum Begriffe-Bewerter), dass unter diesem Key nur *genau ein* Wort (`text` oder `number`) anzugeben ist, da im Teilmengen-Modus ja alle Inputs zu den Teilbegriffen des gesplitteten Begriffs identisch konfiguriert sein müssen.

Anders als beim Begriffsbewerter muss hier nicht für jeden gefragten Begriff einheitlich der Modus und ggf. die Teilmengengröße festgelegt werden, sondern diese Einstellung kann *für jeden Begriff / jede Textlücke separat* erfolgen.

Die Konfiguration erfolgt wieder über die `Teilwortanzahl`-Property: Wenn hinter einer `Wortliste`-Property *direkt* eine `Teilwortanzahl`-Property folgt (und überhaupt der Split-Modus durch Angabe der `Wortliste-Separator`-Property aktiviert wurde), dann ist für diese Wortliste der Teilmengen-Modus aktiviert (und die `Teilwortanzahl`-Property legt fest, wieviele der Teilworte genannt werden müssen, sie muss mit der Anzahl der Eingabefelder zum Begriff übereinstimmen). Für Wortlisten, auf die keine `Teilwortanzahl`-Angabe *direkt* folgt, ist jeweils der Sequenz-Modus aktiv (und es sind genauso viele Eingabefelder zum Begriff vorzusehen wie die Anzahl der Teilworte, aus denen die Worte der Wortliste bestehen).
Beachten Sie, wenn Sie also zu einer Wortliste eine Teilwortanzahl angeben möchten und zusätzlich auch pro Wortliste einen InputType und ggf. eine separate Punktzahl angeben möchten, dass Sie zwar die Properties `InputType` und `Punkte` beliebig hinter jede `Wortliste` schreiben können, aber die `Teilwortanzahl` jeweils wirklich *direkt* auf die `Wortliste` folgen muss, also `InputType` und `Punkte` nicht dazwischen stehen dürfen, sondern dann hinter der `Teilwortanzahl` folgen sollten![grundTAdirektNachWL]

Fragen und Aufgaben aus der Wertung nehmen

Sollten sich einzelne Fragestellungen als missverständlich / fehlerhaft erwiesen haben, können Sie diese nachträglich aus der Wertung nehmen. Die Teilnehmer bekommen dann unabhängig von ihrer Antwort immer volle Punktzahl.

Wie Sie einzelne Fragen aus der Wertung nehmen, steht jeweils in den vorangegangenen Abschnitten spezifisch für den jeweiligen Bewertungstyp angegeben. In der Regel notieren Sie dazu ein `*` unter `Vorgabe:`. Bei Multiple-Choice-Fragen (Mehrfachauswahl, X aus N) können Sie alternativ auch nur einzelne Antwortalternativen aus der Wertung nehmen (durch `+`-Suffix zu kennzeichnen, s.o.).

Um eine *gesamte Aufgabe* aus der Wertung zu nehmen, gehen Sie wie folgt vor:

- Nehmen Sie jede Frage innerhalb der Aufgabe aus der Wertung, d.h. bearbeiten Sie die `Vorgabe` für jeden `Init`-Block entsprechend.

Damit bekommen allerdings im Standardfall nur die Aufgabenteilnehmer die volle Punktzahl. Wenn Sie möchten, dass *alle Heftteilnehmer* die volle Punktzahl bekommen, auch wenn sie die fragliche Aufgabe – vielleicht auch wegen der fehlerhaften Aufgabenstellung – gar nicht bearbeitet haben, dann sind zusätzlich folgende Maßnahmen nötig:

- Stellen Sie in der fortgeschrittenen Aufgabenerstellung die Korrekturart von »automatisch« auf »automatisch, auch unbewertete (Teil)aufgabe(n) bewerten« um.

Falls alle Aufgabenteilnehmer immer alle Fragen angezeigt bekommen, genügt das bereits. Nur falls Sie eine *randomisierte Fragenauswahl* eingestellt haben, so dass jedem Aufgabenteilnehmer nur eine zufällige Auswahl von Fragen aus der Aufgabenseite angezeigt wird, und Sie dazu die oben beschriebene Einstellung `NurBearbeiteteTeilaufgaben: true` vorgenommen haben, dann ist noch folgende dritte Maßnahme nötig:

- Geben Sie zusätzlich zu `NurBearbeiteteTeilaufgaben: true` auch noch die Einstellung `PunktzahlUnbearbeitet:` an und notieren Sie dort die Punktzahl hinter dem Doppelpunkt, die den Teilnehmern vergeben werden soll, die die Aufgabe nicht bearbeitet haben. Das sollte immer die volle Aufgabenpunktzahl sein, also die pro Frage erreichbare Punktzahl multipliziert mit der Anzahl der für jeden Teilnehmer zufällig auszuwählenden Fragen.

Der Beginn einer solchen Korrekturmoduleinstellung könnte also z.B. wie folgt aussehen:

```
Schablonenoption:
 Musterloesung
 Gesamtbewertung
NurBearbeiteteTeilaufgaben: true
PunktzahlUnbearbeitet: 8

Init:
 Aufgabentyp: 1AUSN
 Name:  Frage 1
 Kommentar:
 Punkte:  4
 Gewichtung: 1
 N:  4
 Vorgabe: *
… (min. 2 weitere analoge Init-Blöcke folgend)
```

Dieses Beispiel soll zu einer Aufgabenseite gehören, die mehrere Einfachauswahlfragen à jeweils 4 Punkten enthält, die alle aus der Wertung genommen wurden (`Vorgabe: *`), und von denen per Randomisierung zufällig zwei Fragen ausgewählt werden, so dass in der Aufgabe insgesamt 8 Punkte erreichbar sind.

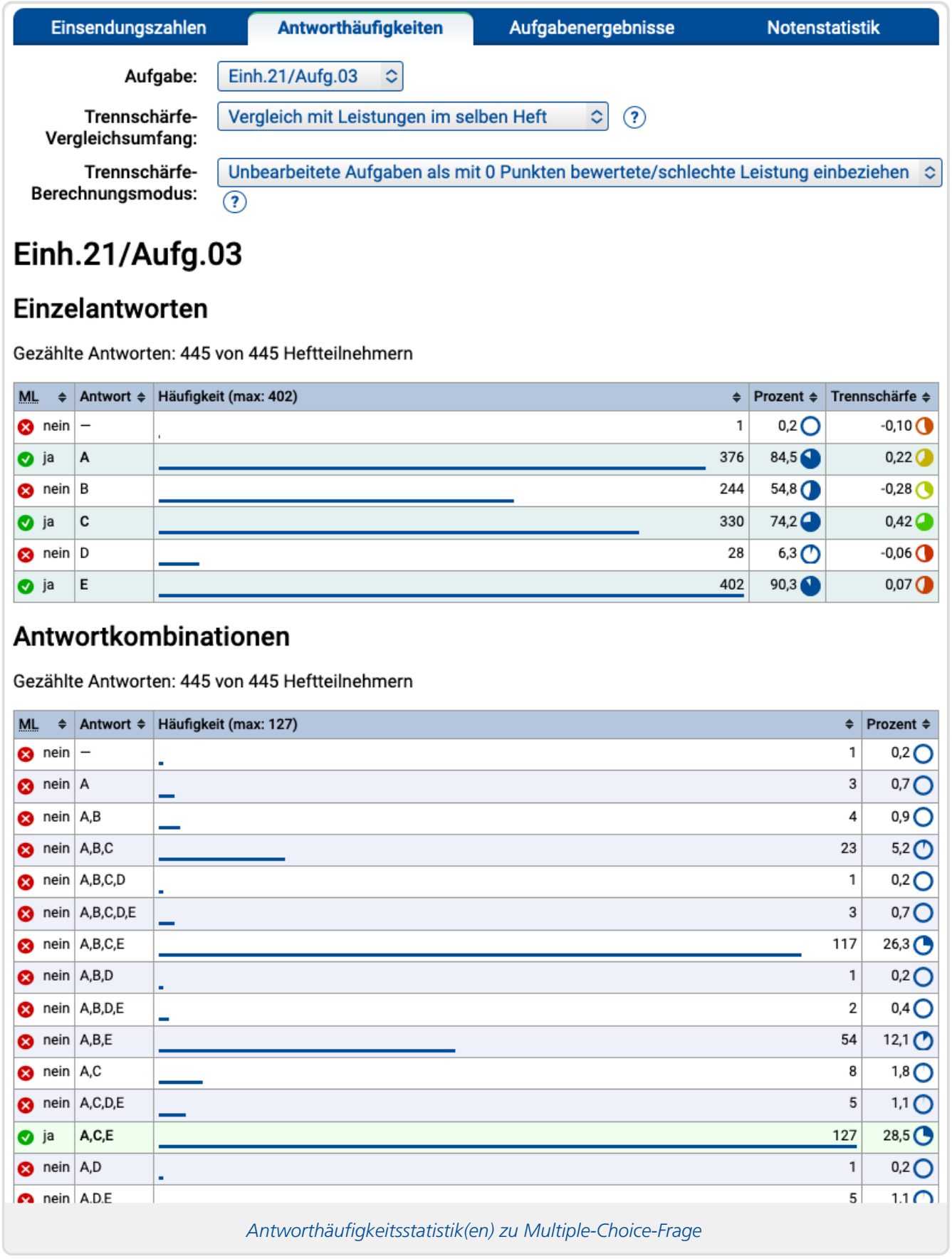
Bisher haben wir den Fall der Punkte für unbearbeitete Aufgaben nur betrachtet für komplett aus der Wertung genommene Aufgaben. Prinzipiell ist es aber natürlich auch möglich, *nur einzelne Fragen einer Aufgabe aus der Wertung zu nehmen* und andere Fragen derselben Aufgabe weiterhin regulär zu werten (sofern keine zufällige Fragenauswahl benutzt wird). Auch in diesem Fall können Sie natürlich überlegen, welche Korrekturart Sie einstellen. Bei normaler »automatisch«-Einstellung bekämen dann nur die *Aufgabenteilnehmer* Punkte für die aus der Wertung genommenen Fragen gutgeschrieben (zusätzlich zu den Punkten, die sie regulär in den verbliebenen Fragen derselben

Aufgabe erreichen). Bei Korrekturart »automatisch, auch unbewertete (Teil)aufgabe(n) bewerten« bekommen dann auch diejenigen *Heftteilnehmer*, die die Aufgabe nicht bearbeitet haben, die vollen Punkte für die aus der Wertung genommenen Fragen und jeweils 0 Punkte für die regulär bewerteten (unbeantworteten) Fragen derselben Aufgabenseite.
Dieses Szenario „nur einzelne, aber nicht alle Fragen einer Aufgabe aus der Wertung nehmen" ist nicht sinnvoll mit zufälliger Fragenauswahl kombinierbar, schon allein, da es dann ja vom „Würfelglück" der einzelnen Teilnehmer abhinge, wieviele Punkte sie gutgeschrieben bekämen. Daher erübrigen sich in diesem Fall auch weitere Konfigurationsmaßnahmen wie die `PunktzahlUnbearbeitet:`-Einstellung.

Antworthäufigkeits-Statistik zu einer Aufgabe aktivieren/konfigurieren

Wie eingangs im Abschnitt zu den Standard-Kursressourcen bereits angekündigt, können zu einer Aufgabe spezielle Statistiken zur Häufigkeit gegebener Antworten erstellt werden, indem eine Standard-Kursressource `antwortstatistikx.y.xml` erstellt wird.

Die folgende Abbildung zeigt beispielhaft eine solche Antworthäufigkeitsstatistik für eine X-aus-5-Multiple-Choice-Frage, genauer zwei verschiedene Statistiken zur selben Frage:



Einsendungszahlen | Antworthäufigkeiten | Aufgabenergebnisse | Notenstatistik

Aufgabe: Einh.21/Aufg.03

Trennschärfe-Vergleichsumfang: Vergleich mit Leistungen im selben Heft

Trennschärfe-Berechnungsmodus: Unbearbeitete Aufgaben als mit 0 Punkten bewertete/schlechte Leistung einbeziehen

Einh.21/Aufg.03

Einzelantworten

Gezählte Antworten: 445 von 445 Heftteilnehmern

ML	Antwort	Häufigkeit (max: 402)	Prozent	Trennschärfe
nein	–	1	0,2	-0,10
ja	A	376	84,5	0,22
nein	B	244	54,8	-0,28
ja	C	330	74,2	0,42
nein	D	28	6,3	-0,06
ja	E	402	90,3	0,07

Antwortkombinationen

Gezählte Antworten: 445 von 445 Heftteilnehmern

ML	Antwort	Häufigkeit (max: 127)	Prozent
nein	–	1	0,2
nein	A	3	0,7
nein	A,B	4	0,9
nein	A,B,C	23	5,2
nein	A,B,C,D	1	0,2
nein	A,B,C,D,E	3	0,7
nein	A,B,C,E	117	26,3
nein	A,B,D	1	0,2
nein	A,B,D,E	2	0,4
nein	A,B,E	54	12,1
nein	A,C	8	1,8
nein	A,C,D,E	5	1,1
ja	A,C,E	127	28,5
nein	A,D	1	0,2
nein	A,D,E	5	1,1

Antworthäufigkeitsstatistik(en) zu Multiple-Choice-Frage

Die erste der beiden Statistiken zeigt die Häufigkeit der einzelnen Antworten, sowie die Markierung, welche dieser Antworten richtig bzw. falsch sind. Die zweite Statistik zeigt die Häufigkeit der tatsächlich gegebenen kombinierten Antworten (als Aufzählung der markierten Antwortalternativen).

Die Statistik aus obiger Beispielabbildung wurde definiert über folgende `aufgabenstatistik1.3.xml`:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<antwortstatistiken xsi:noNamespaceSchemaLocation="https://online-
uebungssystem.fernuni-hagen.de/download/antwortstatistik.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <statistik maxlaenge="10" trenner="," case-sensitive="true">
        <titel>Einh.21/Aufg.03</titel>
        <untertitel>Einzelantworten</untertitel>
        <feld>A1</feld>
        <trennschaerfe/>
        <muster>A</muster>
        <muster>C</muster>
        <muster>E</muster>
    </statistik>
    <statistik maxlaenge="10" case-sensitive="true">
        <untertitel>Antwortkombinationen</untertitel>
        <feld>A1</feld>
        <muster>A,C,E</muster>
    </statistik>
</antwortstatistiken>
```

Die folgenden Abschnitte behandeln den Aufbau der XML-Datei im Allgemeinen.

XML Schema Definition (XSD)

Eine formale Schema-Spezifikation für diesen XML-Aufbau (im XSD-Format) können Sie im Online-Übungssystem herunterladen (siehe `xsi:noNameSpaceSchemaLocation` in obigem Listing, außerdem wird das XSD auch in der fortgeschrittenen Aufgabenerstellung verlinkt). Dieses XSD enthält auch deutschsprachige Erläuterungstexte zu den einzelnen XML-Elementen und -Attributen und fungiert somit als Referenz.

Für die Offline-Erstellung-/Bearbeitung der XML-Datei wird ein XML-Editor empfohlen, der das im XML verlinkte XSD lesen und auswerten kann. Im Gegensatz zu einem einfachen Texteditor unterstützt ein solcher XML-Editor i.d.R. Features wie Validieren (also Fehlermeldungen, wenn Sie vom Schema abweichen), Auto-Completion (Vorschläge erlaubter/erwarteter Kindelemente oder Attribute) oder auch Anzeige der Hilfetexte aus dem XSD zum gerade bearbeiteten Element.

Falls Sie eine solche XML-Datei zu einer Aufgabe neu erstellen möchten, wird empfohlen, in der fortgeschrittenen Aufgabenerstellung zunächst die Funktion zum Generieren einer XML-Datei auszuführen und diese dann zur Nachbearbeitung herunterzuladen. Auf diese Weise erhalten Sie insb. bereits den XML-Rahmen samt Referenz aufs XSD „geschenkt“. Alternativ können Sie aber auch eine XML-Datei komplett von Hand erstellen und das Root-Tag z.B. aus obigem Listing kopieren.

Beschreibung des Dateiaufbaus

Ihre Statistikseite kann mehrere Statistiken enthalten: Zunächst kann Ihre Aufgabe ja aus mehreren Fragen bestehen <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#fragen> , und Sie können zu jeder Frage eine Statistik erzeugen – oder auch mehr als eine Statistik pro Frage, wie in obigem Fallbeispiel, das zu einer einzigen Multiple-Choice-Frage zwei verschiedene Statistiken definiert.

Für jede zu erstellende Statistik ist im XML (unterhalb des Root-Elements `antwortstatistiken`) ein `statistik`-Element zu erstellen.

Attribute von `statistik`

maxlaenge

Das Element `statistik` hat ein Pflichtattribut, das Sie also in jedem Fall angeben müssen: `maxlaenge`. Darin ist eine natürliche Zahl (> 0) anzugeben. Diese legt fest, bis zu welcher Länge Eingaben überhaupt getrennt gezählt[23] werden. Legen Sie z.B. `maxlaenge="5"` fest, so werden alle Eingaben aus dem (später noch festzulegenden) Eingabefeld einzeln aufgeschlüsselt, die maximal 5 Zeichen lang sind, während längere Eingaben unter einem Sammeleintrag »Länger als 5 Zeichen« zusammengefasst werden. Die Längenbeschränkung soll sicherstellen, dass die Statistik nicht „explodieren" kann. Gerade falls lange Eingaben möglich sein sollten, gehören sie in der Regel nicht in eine solche Statistik: Einerseits würden sehr lange Eingaben mitunter schon zu riesigen Tabellenzellen führen, außerdem könnte es von sehr langen Eingaben auch eine so große Anzahl unterschiedlicher Eingaben geben, dass die Anzahl der Tabellenzeilen sehr lang und die Statistik damit dennoch wenig aussagekräftig wird (weil solche langen Eingaben i.d.R. ohnehin nicht mehrfach, sondern nur einmal vorkommen).

In obigem Beispiel einer X-aus-5-Frage können 5 verschiedene Antworten gegeben werden, mit Kommas als Trennzeichen ist die Maximallänge aller gültigen Antworten also 9 Zeichen (Länge von `A,B,C,D,E`). Das ist im Beispiel auf 10 Zeichen aufgerundet worden.

Im Falle von aus mehreren Teileingaben zusammengesetzten Eingaben (vgl. Mehrere gleichnamige Eingabefelder, `data-separator`-Attribut) gilt die `maxlaenge`-Schranke für die gesamte, zusammengesetzte Einsendung (und gibt *nicht* die Maximallänge für jede Teileingabe an).

case-sensitive

Optional können Sie noch das Attribut `case-sensitive` vom Typ Boolean hinzufügen, das festlegt, ob die Groß-/Kleinschreibung der Antworten ignoriert oder beachtet werden soll. Wird das Attribut nicht angegeben, so wird als Defaultwert `true` angenommen. Es wird empfohlen, diese Option wirklich nur dann auf `false` einzustellen, wenn Eingaben in unterschiedlicher Groß-/Kleinschreibung überhaupt möglich sind und dennoch unabhängig von ihrer Groß-/Kleinschreibung gezählt werden sollen. In der Statistik-Anzeige (Tabelle) werden dann alle gegebenen Antworten in Kleinschreibung ausgegeben. (Bei der Multiple-Choice-Frage aus obigem Beispiel wäre es z.B. eigentlich egal, ob die Groß-/Kleinschreibung beachtet wird oder nicht, da ohnehin alle Alternativen nur aus Großbuchstaben bestehen. Aber die Ausgabe der Antworten in der Statistik soll in unveränderten Großbuchstaben erfolgen.)

regex

Optional kann das Attribut `regex` vom Typ Boolean hinzugefügt werden. Wenn es nicht angegeben wird, wird der Defaultwert „false" angenommen.

Die Einstellung `regex="true"` bewirkt, dass versucht wird, Musterlösungen aus den Kindelementen `muster` oder `synonym` nicht als String-Literal, sondern als *regulären Ausdruck* (ausführliche Einführung und Referenz, englisch <https://www.regular-expressions.info> , Wikipedia <https://de.wikipedia.org/wiki/Regulärer_Ausdruck>) zu interpretieren. Alle Eingaben, die auf diesen regulären Ausdruck passen, werden dann als korrekte Antwort klassifiziert und zu einem Statistik-Eintrag

zusammengefasst, d.h. es werden nicht alle verschiedenen Eingaben, die alle auf das Muster passen, getrennt gezählt, sondern vielmehr die Gesamtsumme aller aufs Muster passenden Eingaben ausgegeben.

Dieser Modus ist insbesondere vorgesehen für Statistiken zu automatisch bewerteten Begriffsfragen mit Fragetyp `Begriffe` oder `Begriffsfolge` bei jeweils aktiviertem RegEx-Matching für Lösungsvorgaben in der Bewerterkonfiguration.

trenner / escape

Ebenfalls optional ist das Attribut `trenner`: Wenn die Antworten aus dem Eingabefeld als Ganzes gezählt werden sollen (wie bei der zweiten Statistik in obigem Beispiel), lassen Sie das Attribut weg. Wird es angegeben, so werden die Eingaben nach dem darin befindlichen regulären Ausdruck <https://www.regular-expressions.info> in Teileingaben zerlegt, welche dann gezählt werden. Falls mehrere Eingaben aus verschiedenen Input-Elementen im selben Eingabefeld gespeichert werden – wie es z.B. bei Multiple-Choice-Fragen der Fall ist, wo mehrere Checkboxen demselben Eingabefeld (`FeldA1` in obigem Beispiel) zugeordnet werden –, zählt das Online-Übungssystem diese Antworten automatisch durch Kommas getrennt auf (siehe Beispielstatistik oben), der übliche Wert für das `trenner`-Attribut ist daher ein Komma: `trenner=","`.

Es gibt noch einen zweiten Modus zur Trennung von Eingaben, nicht mittels eines regulären Ausdrucks, sondern mittels eines einzelnen Trennzeichens sowie eines Escape-Zeichens. Dies setzt voraus, dass die Eingaben auch entsprechend zusammengefügt wurden, also Teileingaben durch ein Trennzeichen getrennt aufgezählt wurden und alle möglichen Vorkommen des Trennzeichens in den Teileingaben selbst, die also keine Trennstelle markieren sollen, durch Voranstellen eines Escape-Zeichens markiert wurden. Vorkommen des Escape-Zeichens in den Eingaben müssen dann ebenfalls ein weiteres Escape-Zeichen vorangestellt bekommen haben (also verdoppelt worden sein).

Genau so werden Eingaben aus mehreren gleichnamigen Eingabefeldern vom Online-Übungssystem zusammengesetzt, wenn zum Eingabefeld das `data-separator`-Attribut verwendet wurde, siehe Abschnitt Mehrere gleichnamige Eingabefelder, `data-separator`-Attribut.

Solche Einsendungen können Sie für die Statistik wieder entsprechend zerlegen, indem Sie im Attribut `trenner` genau das Trennzeichen aus dem `data-separator`-Attribut angeben und der Statistik *zusätzlich* auch noch das Attribut `escape` hinzufügen, welches das Escape-Zeichen festlegt. Per Default ist das für solche Einsendungen ein Backslash (`escape="\"`), es sei denn, das Trennzeichen ist bereits ein Backslash, dann ist als Escape-Zeichen ein Slash anzugeben (`trenner="\" escape="/"`).

sortiert-aufzaehlen

Das `trenner`-Attribut kann optional ergänzt werden um ein zweites Attribut namens `sortiert-aufzaehlen`. (Falls kein `trenner` -Attribut angegeben wird, wird dieses Attribut ggf. ignoriert.) Dies bewirkt, dass die Eingabe zwar zunächst nach dem im `trenner` angegebenen regulären Ausdruck (bzw. Trennzeichen, wenn zusätzlich `escape` angegeben wurde) in mehrere Teilantworten aufgeteilt wird, dass diese Liste von Teilantworten aber anschließend alphabetisch sortiert und wieder zu einer neuen Liste zusammengefügt werden soll, wobei der String, der hier im `sortiert-aufzaehlen`-Attribut als neuer Trenner verwendet wird. Am Ende wird also aus einer eingegangenen Aufzählung wieder eine Aufzählung, aber dieses Verfahren kann dazu dienen, diese Aufzählung zu sortieren und/oder die Trennzeichen zu ersetzen oder zu vereinheitlichen. Zum Zwecke der Vereinheitlichung werden außerdem alle Leerzeichen vor und nach den Trenner-Vorkommen ignoriert/entfernt, und im Falle einer Verwendung von `trenner`- mit `escape`-Attribut werden auch die Escape-Sequenzen aufgelöst (analog zum `aufzaehlen`-Attribut, s.u.).

Die Sortierfunktion ergibt z.B. Sinn bei Multiple-Choice-Fragen mit Zufallsreihenfolge der Antwortalternativen: Falls z.B. zwei Teilnehmer dieselben Antworten A, C und D markieren, aber in ihren jeweiligen Aufgabenformularen die Checkboxen dazu in verschiedenen Reihenfolgen stehen, könnte z.B. Student 1 die Einsendung `A,D,C` vornehmen, während Student 2 mit denselben Markierungen die Einsendung `C,A,D` einsendet. In der Statistik werden Sie sich in der Regel nur dafür interessieren, wie oft die Kombination aus den drei Antworten A, C und D genannt wurde, egal in welcher Reihenfolge, also die beiden Beispiel-Einsendungen oben nicht als getrennte Statistikeinträge sehen wollen. Hier gibt es dann also Sinn, `trenner="," sortiert-aufzaehlen="," ` ins `<statistik>`-Element zu notieren, damit eben alle beide Antworten vor der Zählung zu `A,C,D` konvertiert werden.

aufzaehlen

Alternativ zu `sortiert-aufzaehlen` kann auch das Attribut `aufaehlen` angegeben werden, welches ebenfalls die zuvor zerlegten Teile wieder aufzählt, allerdings ohne sie im Vorfeld zu sortieren. Das kann z.B. für eine „normalisierte“ Darstellung einer Aufzählung sorgen und z.B. das Trennzeichen aus `trenner` durch einen „hübscheren“ String ersetzen, außerdem werden Leerzeichen rund um die Trennzeichen aus der Originaleinsendung entfernt und bei Kombination mit dem `escape`-Attribut auch die Escape-Maskierungen aus der Eingabe entfernt.

Nehmen wir z.B. die Attributkombination `trenner="|" escape="\" aufzaehlen=", "` an: Damit würden die Einsendungen `Hallo| Welt`, `Hallo | Welt` und `Hallo |Welt` sämtlich in der Statistik als `Hallo, Welt` zusammengefasst und Eingabe wie `Ha\|\|o | Welt` zu `Ha||o, Welt` umgewandelt.

`nachkommastellen`

Falls nach einer Zahl in einem bestimmten Lösungsintervall gefragt sein sollte, sollten Sie das optionale Attribut `nachkommastellen` angeben. Es darf den Wert 0 oder größer haben. Wird es angegeben, so werden alle Eingaben, die korrekt als Zahl interpretiert sind, auf die angegebene Anzahl von Nachkommastellen gerundet und dann gezählt. Beispiel: Falls Sie `nachkommastellen="2"` angeben, so gelten alle Eingaben wie `"1,5"`, `"1.50"`, aber auch `"1.498"` oder `"1,502"` als gleich, und ihre gemeinsame Anzahl (hier 4) wird dann in der Statistik angegeben, beschriftet mit `"1,50"`. Einzige Ausnahme: Falls Sie über das unten noch vorzustellende `musterinvervall`-Element festlegen, dass korrekte Eingaben in einem Intervall wie [1,50..2,00[liegen müssen, so werden in der Statistik unter Label `"1,50"` nur die Werte ∈ [1,500..1,505[zusammengefasst, während die kleineren Werte ∈ [1,495..1,500[dann separat gezählt und als `" < 1,50"` aufgeführt werden. Grund ist, dass bei Eingabe eines Lösungsinvervalls korrekte und falsche Antworten getrennt gezählt werden sollen, und eben in diesem Fall die Werte außerhalb des Lösungsintervalls, die erst durch Rundung ins Intervall „rutschen“ würden, extra herausgerechnet werden. Dasselbe passiert natürlich auch im umgekehrten Fall, dass korrekte Eingaben durch Rundung aus dem Lösungsintervall herausrutschen.

`zahl-aus-text`

Falls nach einer Zahl gefragt sein sollte, aber Studierende auch beliebigen Text eingeben können, kann es z.B. zu folgender Situation kommen: Gefragt sein könnte nach einem Betrag in Euro, der einfach nur als Zahl eingegeben werden soll, aber ein Student schreibt noch die Einheit „EUR“ oder „€“ vor oder hinter der Zahl mit ins Eingabefeld (also z.B. „50 €“ oder „EURO 50,-“. Sie haben nun die Wahl, ob Sie eine solche Eingabe als (falschen) Text auswerten lassen möchten oder ob aus diesem Text (sofern er nicht länger ist als die `maxlaenge`-Einstellung, s.o.) die Zahl „50“ extrahiert und als solche (und somit richtige Antwort) in der Statistik gezählt werden soll:

- Mit Attribut `zahl-aus-text="true"` mit im `statistik`-Element aktivieren Sie die Extraktion von Zahlen aus Texteingaben. Genauer: Enthält die Eingabe genau eine Zahl, ggf. aber mit Text davor oder dahinter, so wird die Eingabe als diese Zahl gewertet. Text ohne Zahl wird genauso wie Text mit mehr als einer Zahl (wie z.B. „4,3 von 5“) werden nach wie vor als Text (und somit i.d.R. falsche Antwort) gezählt.
- Mit Attribut `zahl-aus-text="false"` dagegen wird die Zahlen-Extraktion explizit abgeschaltet.
- Falls Sie das Attribut *nicht* angeben, gelten folgende Standardeinstellungen:
  - Wenn das Attribut `nachkommastellen` (s.o.) angegeben wurde, ist die Zahlenextraktion per Default aktiviert (lässt sich aber per `zahl-aus-text="false"` abschalten).
  - Wenn `nachkommastellen` nicht angegeben wurde, ist die Zahlenextraktion per Default abgeschaltet (lässt sich aber per `zahl-aus-text="true"` aktivieren.)

Diese Defaultwerte wurden auch mit Blick auf den internen automatischen Aufgabenbewerter gewählt. Denn der Modus, den Sie für die Statistikanzeige wählen, sollte natürlich mit der tatsächlichen Bewertung der jeweiligen Aufgabe konsistent sein:

- Der Zahlenbewerter verwendet z.B. selbst eine Zahl-aus-Text-Extraktion, d.h. er erwartet eine Zahleneingabe, und wenn vor oder nach einer Zahl noch Text eingegeben wurde, wird er den ignorieren und nur die Zahl bewerten. Wenn Sie also Fragen erstellen, die mit dem internen Zahlenbewerter bewertet werden sollen, dann sollten Sie auch in der Statistik diesen Modus aktivieren. Normalerweise sollte aber für Statistiken zu Zahlenfragen ohnehin auch das

`nachkommastellen`-Attribut angegeben werden, welches die Zahlenextraktion auch ohne Angabe dieses Attributs implizit mit aktiviert.

  - Anmerkung: Die Eingabefelder zu Zahlenfragen werden im Aufgabenformular typischerweise so eingestellt, dass ohnehin kein Text eingegeben werden kann (durch `type="number"` am Input-Tag, siehe Dokumentation zur Formular-Eingabevalidierung <https://online-uebungssystem.fernuni-hagen.de/download/JSFormValidation/FormularValidierung.html>). In dem Fall ist die Einstellung ohnehin nicht so wichtig. Sie haben aber auch die Möglichkeit, reine Texteingabefelder zu verwenden, und spätestens in diesem Fall hätte die Option dann doch wieder eine Auswirkung auf die Statistik.

- Die Begriffs- und Begriffsfolge-/Lückentext-Bewerter können jeweils in einen Smartmodus versetzt werden, in denen sie reine Zahleneingaben von Text unterscheiden können und in dem Fall dann Zahlen auch als korrekt werten, wenn diese zwar wertgleich, aber nicht syntaktisch gleich zur Musterlösung sind. Wenn also z.B. die Eingabe „50“ erwartet wird, werden im Smart-Modus auch Eingaben wie „50,0“ oder „50.00“ akzeptiert. Sobald Text vorkommt, wird das Ganze jedoch nicht als Zahlen-, sondern als Texteingabe interpretiert. Für eine Statistik zu diesem Bewerter sollte daher auch in der Antwortstatistik der `zahl-aus-text`-Modus *nicht* aktiviert werden. Da hier aber auch `nachkommastellen` normalerweise nicht angegeben wird, ist das auch hier ohnehin der Default.

Die genaue Auswirkung der Einstellung auf die Zählung in der Statistik hängt ebenfalls vom `nachkommastellen`-Attribut ab:

- *Wenn* `nachkommastellen` angegeben wurde, so werden ja alle Zahleneingaben, die nach Rundung auf diese Nachkommastellenzahl übereinstimmen, sich aber in der konkreten Schreibweise (wie Nachkommastellenzahl, Eingabe des Dezimaltrenners, führende Nullen oder Vorzeichen etc.) unterscheiden, in der Statistik zu einem einzigen Eintrag mit normierter Schreibweise zusammengefasst. Dasselbe gilt dann auch für solche Texteingaben mit enthaltener Zahl: Auch diese werden mit den Eingaben derselben Zahl ohne oder mit anderem Text zusammengefasst, d.h. die Text-Suffixe oder -Präfixe in den Eingaben tauchen dann in der Statistik gar nicht auf.
  Beispiel: Eine Eingabe wie „EUR 50,-“, „50 €“ oder „50,00 €“ würde also jeweils als Zahl 50 interpretiert und mit reinen Zahleneingaben wie „50“, „0050“ oder „50,0“ etc. zusammengefasst.
  Das ist, wie gesagt, das Defaultverhalten, lässt sich aber durch Angabe von `zahl-aus-text="false"` unterbinden.
- Wenn dagegen das `nachkommastellen`-Attribut *nicht* angegeben wurde, aber dafür explizit `zahl-aus-text="true"` gesetzt wurde, so gilt:
  1. Falls Sie dennoch nur numerische Musterlösungen mittels `musterintervall`-Elementen (s.u.) festgelegt haben, so werden zwar alle Texteingaben, die eine in diesem Intervall liegende Zahl enthalten, als korrekt markiert, in der Statistik wird dann aber dennoch jede verschiedene Schreibweise/Texteingabe einzeln aufgeführt und gezählt. D.h. eine Zusammenfassung von verschiedenen Texteingaben mit gleichem Zahlenwert erfolgt dann nicht, sondern nur in Kombination mit der `nachkommastellen`-Einstellung, wie oben beschrieben.
     Haben Sie also z.B. ein Musterintervall [50, 50] festgelegt, so würden Eingaben wie „EUR 50,-“, „50 €“ oder „50,00 €“zwar sämtlich in der Statistik als korrekte Antworten markiert, aber dennoch alle einzeln gezählt.
  2. Falls Sie eine Aufgabe gestellt haben, in der Texteingaben und auch Zahleneingaben akzeptiert werden, im `Statistik`-Element *kein* `nachkommastellen`-Attribut angeben und korrekte Eingaben mit normalen `muster`-Elementen für Text-Musterlösungen (statt `Musterintervall`) erzeugen, aber dann Zahlenintervalle als

Synonyme zu Textantworten festlegen, so wird bei expliziter Einstellung `zahl-aus-text="true"` auch jede Texteingabe, die eine Zahl aus einem solchen Synonym-Intervall enthält, als Synonym der entsprechenden Antwort betrachtet. Wie alle Synonyme (und anders als oben unter Punkt 1.) werden solche (als Synonyme erkannte Eingaben) dann in der Statistik nicht einzeln gezählt, sondern mit allen Nennungen des zugehörigen Lösungsbegriffs und all seiner Synonyme zusammengefasst.

input-type

Hier können Sie den Wert des `type`-Attributs des entsprechenden HTML-Eingabefelds (`input`) eintragen, über das die in dieser Statistik zu zählenden Eingaben von den Studierenden eingesendet wurden.

Wenn Sie das Attribut nicht angeben oder einen ungültigen bzw. unbekannten Wert eintragen, wird (genau wie von Webbrowsern beim `type`-Attribut von `input`-Elementen) der Defaultwert `text` angenommen.

Derzeit (Stand Mai 2023) wird nur ein vom Standardwert `text` abweichender Inputtype erkannt, und zwar `number`:

Falls die Studierenden hier ausschließlich Zahlen eingeben können, und zwar nicht über ein `<input type="text" ...>`, sondern über ein `<input type="number" ...>`, dann sollten Sie auch hier in der Statistik das Attribut `input-type="number"` angeben. (Ebenso sollten dazu in Quittungen Variablen der Art `$FeldA1DECIMAL` statt `$FeldA1NUM` verwendet werden, und im Falle einer Autokorrektur durch den internen Zahlenbewerter oder Begriffsbewerter dort jeweils die Option `InputType: number` in die Bewerterproperties eintragen.)
Diese Option bewirkt dann eine einheitliche Verarbeitung von Zahleneingaben durch z.B. Statistikerzeugung und Autokorrektur. Insbesondere gilt für Text-Inputs die bereits beschriebene Heuristik, dass in Eingaben wie »1.000« mit genau einem Punkt, auf den drei Nullen folgen, dieser Punkt als Tausendertrenner gelesen und somit ignoriert wird. Bei Zahleninputs (bei denen auch Eingaben wie »1,000« als »1.000« vom Browser gesendet werden), passiert so eine Plausibilisieren explizit nicht, sondern ein Punkt wird *immer* als Dezimaltrennzeichen gelesen.

Das Attribut `input-type="number"` kann *nicht* mit `zahl-aus-text="true"` kombiniert werden. Da in Zahlen-Inputs ohnehin kein Text eingegeben werden kann, ergibt diese Kombination auch keinen Sinn. (Das genaue Verhalten der Statistikerzeugung im Fall, dass beides angegeben wird, ist nicht definiert und kann sich jederzeit ändern.)

uebereinstimmung

Der integrierte Aufgabenbewerter für Begriffs- und Lückentext-Fragen bietet, wie oben im Kapitel zu den Bewertereinstellungen beschrieben, eine Bewerteroption `Uebereinstimmung` als Prozentangabe, mit der festgelegt werden kann, wie stark eine Eingabe von der Musterlösung abweichen darf, um noch als „Tippfehler" und somit trotz der Abweichung als korrekte Antwort gezählt zu werden. Falls in der Aufgabe beispielsweise die Eingabe „Schifffahrt" erwartet wird (11 Zeichen), aber ein Student „Schiffhart" (10 Zeichen) eingibt, würde der Bewerter eine Übereinstimmung in 9 von 11 Zeichen ermitteln (1 von 11 Zeichen vergessen und zwei weitere vertauscht, ein solcher „Buchstabendreher" wird nur als ein einziger Fehler gezählt), also 81,81%, was abgerundet wird auf 81%. Wäre in der Aufgabe der Bewerter nun z.B. auf `Uebereinstimmung: 80` eingestellt, so würde die Eingabe „Schiffhart" noch als korrekt bewertet.
In Ihrer Statistik jedoch würde – in Grundeinstellung – diese abweichende Eingabe „Schiffhart" gesondert gezählt und ggf. (sofern Sie darin die Musterlösung „Schifffahrt", aber nicht die Musterlösung „Schiffhart" als `muster`-Kindelement festlegen, siehe unten) als falsche Eingabe markiert.

Damit in Fällen wie gerade beschrieben die Statistik analog zum Aufgabenbewerter arbeitet und von letzterem ignorierte Tippfehler in der Statistik ebenfalls ignoriert werden, fügen Sie in solchen Fällen dem `statistik`-Tag noch das Attribut `uebereinstimmung` mit demselben Prozentwert hinzu. In obigem Beispiel sähe das also wie folgt aus:

```
<statistik … uebereinstimmung="80">
 …
 <muster>Schifffahrt</muster>
</statistik>
```

Mit dieser Einstellung würden nun alle Eingaben wie „Schiffhart", „Schifffart", „Schiffahrt" etc., die zwar nicht exakt, aber zu mindestens 80% mit der Musterlösung „Schifffahrt" übereinstimmen, in der Statistik als Eingabe „Schifffahrt" mitgezählt und nicht als (falsche) Eingabe gesondert in der Statistik aufgeführt

Dieses Attribut wird nur für Textvergleiche mit String-Literalen verwendet. Falls Sie das Attribut `regex="true"` angegeben haben, wird das Attribut `uebereinstimmung` ganz ignoriert. Und auf Musterintervalle (Zahlenvergleiche) hat es ansonsten auch keine Auswirkung.

Kindelemente von `statistik`

`titel` und `untertitel`

Die ersten beiden Kindelemente sind `titel` und `untertitel`. In `titel` geben Sie eine Überschrift für die Statistik an, typischerweise einfach den Titel der Frage, zu der die Statistik erstellt wird. Ein Untertitel ist optional und vor allem für den Fall vorgesehen, dass zur selben Frage zwei (oder mehr) Statistiken erzeugt werden sollen – genau wie in obigem Beispiel mit zwei Statistiken zur selben Multiple-Choice-Frage. In diesem Fall geben Sie per `untertitel` jeder Statistik eine untergeordnete Überschrift. Außerdem ist in dem Fall nur der ersten dieser Statistiken ein `titel` zu geben, nachfolgende Statistiken, die sich demselben Titel unterordnen sollen, bekommen kein eigenes `titel`-Kindelement.

feld

Darauf folgt das Pflichtelement `feld`: Dieses beschreibt den Inhalt der Statistik. Genauer: Die Statistik soll ja zählen, wie oft bestimmte Eingaben (von den einzelnen Teilnehmern) auf eine bestimmte Frage gegeben wurden. Die Eingaben wiederum erfolgen in Eingabefelder, wie Sie sie im Aufgabenformular erstellt haben. Zur Erinnerung: Das Formular kann ein oder mehrere `input`-, `textarea`- oder `select`-Elemente enthalten, die einen Namen der Form `FeldA1` tragen müssen, wobei `A` hier die erste (technische) Teilaufgabe <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#ta> bezeichnet und `1` die Feldnummer zu dieser Teilaufgabe darstellt.

Um nun also die Eingaben, die in einem bestimmten Eingabefeld wie `FeldA1` erfolgt sind, in einer Statistik zu zählen, ist dieser das Kindelement `<feld>A1</feld>` hinzuzufügen. (Die Groß-/Kleinschreibung der Teilaufgabenkennung ist dabei egal, `<feld>a1</feld>` wäre also auch korrekt.)

Sie haben auch die Möglichkeit, *mehrere* `feld`-Elemente hintereinander aufzuzählen. In dem Fall wird die Vereinigungsmenge aller Eingaben zu diesen Feldern gebildet und alle Vorkommen daraus gezählt.

Ein Anwendungsfall dafür liegt z.B. vor bei den vom Aufgabenerstellungassistenten erzeugten Begriffe-Fragen mit mehreren Eingabefeldern. Lautet z.B. eine Frage: »Nennen Sie 10 deutsche Bundesländer.« und enthält das Aufgabenformular dazu entsprechend 10 Eingabefelder (z.B. `FeldB3` bis `FeldB12`) zur Eingabe je eines Bundesland-Namens, so wären hier entsprechend in der Statistik zehn Feld-Elemente einzufügen, z.B.:

```
…
<statistik maxlaenge="25" case-sensitive="false">
    <titel>Deutsche Bundesländer</titel>
    <feld>B3</feld>
    <feld>B4</feld>
    …
    <feld>B12</feld>
    <muster>Nordrhein-Westfalen</muster>
    <muster>Hamburg</muster>
    <muster>Bremen</muster>
    <muster>Hessen</muster>
    …
</statistik>
…
```

filter

Für die Ermittlung der Antworthäufigkeit in Prozent wird normalerweise die Teilnehmerzahl der jeweiligen Frage als Bezugsgröße (Maximum) genommen. D.h. wenn Sie einstellen, dass alle Antworten aus einem bestimmten Feld gezählt werden sollen, wird zusätzlich auch gezählt, wieviele Einsendungen zu diesem Eingabefeld (genauer: der Teilaufgabe zu der das Feld gehört) insgesamt vorliegen. Aus beiden Angaben kann ermittelt werden, wieviel Prozent der Teilnehmer die jeweilige Antwort gegeben haben.

Es kann aber Situationen geben, in denen dieses Standardverhalten zur Ermittlung der Antwortzahl in Prozent nicht so gewünscht ist:

- Nehmen wir z.B. an, eine Multiple-Choice-Frage werde so randomisiert, dass nicht jede/-r Aufgabenteilnehmer/-in auch jede Antwortoption sieht und somit diese Antwort überhaupt geben kann.
- Wenn nun nur 50% der Teilnehmer/-innen überhaupt die Antwort `_A` ankreuzen können und von diesen wiederum 80% die Antwort tatsächlich geben, so würde „regulär" eine Antwortquote von 50% × 80% = 40% ermittelt.
- Diese Statistik sagte also dann aus, dass weniger als die Hälfte der Teilnehmer/-innen die Alternative markiert haben (für wahr hielten). In gewisser Weise stimmt das ja auch, aber es suggeriert umgekehrt, dass 60% aller Teilnehmer/-innen die Alternative für falsch hielten – und das wäre eindeutig falsch.
- Interessant wäre vielmehr in erster Linie die folgende Aussage: »80% der Teilnehmer/-innen, denen diese Option nach Randomisierung zur Beantwortung angezeigt wurde, hielten diese für wahr (nur 20% für falsch).« Ebenso wäre die absolute Anzahl dieser Teilnehmer interessant.
- Auch die Berechnung der Trennschärfe sollte in diesen Fällen abweichen: Die genaue Trennschärfeberechnung wird in der Online-Hilfe zur Statistik erläutert. Hier sei nur kurz zusammengefasst, dass für die Folge *X* eine Folge von Einsen und Nullen verwendet wird: Es gibt normalerweise ein Folgenglied pro Student/-in, der/die zur Teilaufgabe etwas eingesendet hat, das den Wert 1 hat, wenn er/sie die jeweilige Antwort, deren Trennschärfe berechnet werden soll, gegeben hat, oder den Wert 0, wenn er/sie die Antwort nicht gegeben hat. In diesen Fällen, in denen gar nicht jede/-r Einsender/-in die Antwort überhaupt geben *kann*, sollten diejenigen, denen die Antwortoption nicht zur Verfügung stand, gar nicht (mit 0-Werten) in diese Folgen der Trennschärfenberechnung mit einfließen!

Diese Ziele können Sie mit Hilfe des `filter`-Elements erreichen. Wenn Sie dieses angeben, werden zwei neue Spalten zur Statistik hinzugefügt (Anzahl der Teilnehmer/-innen, die eine bestimmte Antwort überhaupt geben konnten, sowie davon der prozentuale Anteil der tatsächlich gegebenen Antworten) und auch die Trennschärfeberechnung entsprechend angepasst.
Das setzt natürlich eine entsprechende Aufgabeneinrichtung voraus, denn die Information, wem welche Antwortoptionen zur Verfügung standen, muss mit in den Einsendungen hinterlegt sein, um sie für die Statistik zählen zu können.

Bei randomisierten MC-Fragen z.B. sieht das Online-Übungssystem vor, dass pro Frage zwei Eingabefelder verwendet werden, eines für die Checkboxen, in denen die Teilnehmer ihre Antworten erfassen, und ein „Hidden Field", in dem die ihnen überhaupt angezeigten Auswahloptionen und die Zuordnung der ihnen dabei individuell angezeigten Antwortbuchstaben zu globalen Antwortkennungen aus der Aufgabendefinition gespeichert werden. Das wird oben im Abschnitt Blocklokale Randomisierung demonstriert. Aus den Daten dieses zweiten (versteckten) Eingabefeldes lässt sich die hier benötigte Information auslesen.

Wenn Sie derartige Daten vorliegen haben, fügen Sie dann dem `statistik`-Element hinter dem (bzw. den) `feld`-Kindelement(en) (für die sichtbaren Eingabefelder/Antworten) ein `filter`-Element ein, welches auf das zuvor noch unberücksichtigte Eingabefeld mit den Randomisierungsinformationen Bezug nimmt.

Dieses `filter`-Element hat folgende **Attribute**:

- `feld`: Kennung dieses Feldes, aus dem die Information bezogen werden soll, welche Studierenden zur Ermittlung der Maximalanzahl gezählt werden sollen. Der Aufbau ist wie im `feld`-Element oben, also ein Buchstabe für die Teilaufgabe und eine Feldnummer.
- `matches`: Hinterlegen Sie hier ein „Muster" für einen regulären Ausdruck <https://www.regular-expressions.info> , der beschreibt, welche Einsendungen (in diesem Feld) zur Maximalzahl (wieviele Teilnehmer-/innen eine bestimmte Antwort A überhaupt geben konnten) gezählt werden

sollen, wobei in diesem regulären Ausdruck an Stelle der konkreten Antwort A der Platzhalter "%s" angegeben wird. (Beispiel folgt unten.)

Als **Elementinhalt** kann im `filter`-Element ein Text angegeben werden, der dann als Legende für die damit neu definierte Anzahl-Spalte und die zugehörige Prozentangabe unter der Tabelle mit angegeben wird.

Beispiel 1:
Für eine Multiple-Choice-Aufgabe mit randomisierter Auswahl einer Teilmenge von Antwortoptionen könnte das Statistik-Element wie folgt aussehen:

```
<statistik maxlaenge="30" trenner="," case-sensitive="true">
 <titel>MC Selection+Shuffle</titel>
 <untertitel>Einzelantworten</untertitel>
 <feld>A7</feld>
 <filter feld="A8" matches=".*%s/.*">
  Anzahl Teilnehmer/-innen, denen diese Antwortalternative nach Randomisierung
angeboten wurde
 </filter>
 <trennschaerfe/>
 <muster>_A</muster>
 …
</statistik>
```

Hier werden in `FeldA7` die eigentlichen Checkbox-Eingaben erwartet, in `FeldA8` das „Mapping“ aus Paaren der Form wie `_A/C`, das besagt: Die vom Aufgabenautor vorgesehene erste Antwortalternative `_A` wurde dem/der Teilnehmer/-in angezeigt und für ihn/sie individuell mit `C` bezeichnet. Ein regulärer Ausdruck wie `.*_A/.*` wiederum „matcht“ auf dieses Mapping, wenn darin eben der Substring `_A/` vorkommt (`.*` steht für „beliebig viele beliebige Zeichen“), also dem/der Einsender/-in die Antwortoption `_A` wirklich (mit einem beliebigen individuellen Buchstaben bezeichnet) angezeigt wurde und somit diese/-r Einsender/-in zur Maximalzahl derjenigen Teilnehmer/-innen hinzugezählt werden soll, die diese Antwort überhaupt geben konnten. Da bei der Statistikerstellung für jede gegebene Antwort (hier also z.B. `_A`, `_B`, `_C`, etc.) jeweils ein solcher regulärer Ausdruck benötigt wird, diese sich aber bis auf die entsprechende Antwort selbst nicht unterscheiden, muss hier nicht für jede Antwortmöglichkeit separat ein eigener regulärer Ausdruck angeben werden, sondern eben ein einziges Muster mit dem Platzhalter `%s` an Stelle der jeweiligen Antwort.

Die damit erzeugte Statistik kann dann z.B. wie in folgendem Screenshot aussehen:

MC Selection+Shuffle

Einzelantworten

Gezählte Antworten: 5 von 5 Heftteilnehmern

ML	Antwort	Häufigkeit (max: 3)	von *)	Prozent *)	Prozent gesamt **)	Trennschärfe
ja	_A	2	4	50,0	40,0	-0,88
ja	_B	3	5	60,0	60,0	0,51
nein	_C	1	1	100,0	20,0	–
ja	_D	2	5	40,0	40,0	0,90
nein	_E	2	3	66,7	40,0	-0,96
nein	_F	1	3	33,3	20,0	0,69
nein	_G	1	2	50,0	20,0	1,00
nein	_H	1	3	33,3	20,0	-0,83
ja	_I	1	3	33,3	20,0	0,99

*) Anzahl Teilnehmer/-innen, denen diese Antwortalternative nach Randomisierung angeboten wurde
**) Bezogen auf die Gesamtanzahl gezählter Antworten, siehe oben

Hier sehen Sie die zwei mit „*)“ markierten neuen Spalten, die auf Basis der `filter`-Einstellung des obigen Beispiels erzeugt wurden, einschließlich der unter der Tabelle stehenden Legende, die ebenfalls im obigen XML-Auszug festgelegt wurde.

Noch ein Hinweis: Falls die Groß-/Kleinschreibung beim Pattern Matching ignoriert werden soll, ist dem regulären Ausdruck das Flag `(?i)` (case-insensitive) voranzustellen. Das wird z.B. benötigt, wenn eine Multiple-Choice-Frage *mit 3 Antwortalternativen* (wahr / falsch / keine Antwort) verwendet wird, wobei zu einer Alternative `_A` bei Ankreuzen von »wahr« ein `_A` eingesendet wird, bei Ankreuzen von »falsch« dagegen ein `_a`, im Mapping aber immer nur die Schreibweise `_A/...` steht. Dann soll auch die Anzahl der gegebenen `_a`-Antworten (wieviele Studierenden haben geantwortet, dass sie Alternative `_A` für falsch halten) mit der Anzahl der Studierenden verglichen werden, denen die Option `_A` überhaupt angeboten wurde.

Beispiel 2:
Betrachten wir als Ausgangspunkt das erste Unterbeispiel aus Beispiel 3 des Abschnitts Blocklokale Randomisierung: Eine Zuordnungsfrage mit zufälliger Zeilenreihenfolge. Nur anders als in dem Beispiel sollen die Zeilen der Matrix (die einzelnen Teilfragen) nicht (nur) gemischt werden – das wäre der Statistik völlig egal –, sondern es soll daraus eine zufällige Auswahl getroffen werden. Dazu soll hier lediglich die Zahl 3 (Anzahl der auszuwählenden Zeilen) in der `$Randomize`-Deklaration reduziert werden auf 2. (Ob die Zufallsreihenfolge (Shuffle) beibehalten wird, spielt hier keine Rolle.)

In dem besagten Beispiel wird in `FeldA1` eine Aufzählung von Zeilen-/Fragennummern gespeichert, aus der nicht nur bei „Shuffle-Modus“ die Zufallsreihenfolge ablesbar ist, sondern auch die zufällige Zeilenauswahl. D.h. in diesem Feld steht die Nummer jeder Frage, die dem jeweiligen Studenten angezeigt wurde. In den weiteren Feldern `FeldA2` bis `FeldA4` stehen dann die eigentlichen Einsendungen.

Eine Statistikdefinition dazu könnte z.B. wie folgt aussehen:

```
<statistik maxlaenge="1">
 <titel>Shuffle Rows</titel>
 <untertitel>Pünktchen und…</untertitel>
 <feld>A2</feld>
 <filter feld="A1" matches="(.*,)?1(,.*)?|.*%s">
  Anzahl Teilnehmer/-innen, denen diese Fragezeile nach Randomisierung
angeboten wurde
 </filter>
 <trennschaerfe/>
 <muster>A</muster>
</statistik>
<statistik maxlaenge="1">
 <untertitel>Die dicke…</untertitel>
 <feld>A3</feld>
 <filter feld="A1" matches="(.*,)?2(,.*)?|.*%s">
  Anzahl Teilnehmer/-innen, denen diese Fragezeile nach Randomisierung
angeboten wurde
 </filter>
 <trennschaerfe/>
 <muster>B</muster>
</statistik>
<statistik maxlaenge="1">
 <untertitel>Gaius Julius…</untertitel>
 <feld>A4</feld>
 <filter feld="A1" matches="(.*,)?3(,.*)?|.*%s">
  Anzahl Teilnehmer/-innen, denen diese Fragezeile nach Randomisierung
angeboten wurde
 </filter>
 <trennschaerfe/>
 <muster>C</muster>
</statistik>
```

Betrachten Sie hier jeweils die `filter`-Einstellung, die hier gänzlich anders aussieht als beim Multiple-Choice-Beispiel oben. Denn hier werden ja auch nicht Antwortalternativen zufällig ausgeblendet, sondern ganze Fragen. Für jede dieser 3 Fragen wird eine separate Antworthäufigkeitstatistik erzeugt. Es ist aber interessant, dort eben die jeweiligen Antworthäufigkeiten nicht (nur) in Relation zur Anzahl der Aufgabeneinsender zu setzen, sondern pro Frage eine Angabe zu erhalten, wievielen Teilnehmern überhaupt diese Frage gestellt wurde (und wieviel Prozent davon bestimmte Antworten gegeben haben).

Diese Anzahl der Personen, denen die Frage gestellt wurde, ist also für jede Antwortalternative gleich, hängt ja nur von der Fragenauswahl ab. Daher wird (vorerst zumindest) der Platzhalter `%s` im Muster gar nicht benötigt. Wir benötigen lediglich pro Frage ein Muster, das erkennt, ob die Nummer der Frage in der Einsendung von `FeldA1` (Aufzählung der einzelnen Fragennummern) vorkommt. Ein einfacher regulärer Ausdruck dafür lautet schlicht z.B. `.*1.*` für die erste Frage, `.*2.*` für die zweite Frage etc. Solche regulären Ausdrücke funktionieren aber nur, so lange es nicht mehr als 9 Fragen gibt. Sobald die Fragenzahl zweistellig werden kann, würde `.*1.*` z.B. nur feststellen, ob eine Frage gewählt wurde, in deren Nummer die Ziffer 1 vorkommt, also Frage Nr. 1, 10, 11, 12, …, 19, 21, etc. Im obigen Beispiel wurde daher ein etwas komplexerer Ausdruck wie `(.*,)?1(,.*)?` für die erste Frage verwendet. Dieser sucht nach einer 1, die entweder auf ein Komma und beliebigen Text vor dem Komma folgt oder direkt am Anfang der Aufzählung steht und auf die entweder ein Komma und weiterer Text folgt oder die ansonsten das Ende der Aufzählung bildet. Das vermeidet die Erkennung von Teilübereinstimmungen mit zweistelligen Nummern.

Mit diesem Ausdruck allein hätte die Statistik aber noch einen kleinen Schönheitsfehler, denn sie zählt nicht nur die Anzahl der Antworten zu jeder Antwortalternative, sondern auch die Anzahl der Studenten, die eine Frage gar nicht beantwortet haben – und in letzterem Fall wird nicht unterschieden, ob jemand die Frage nicht beantwortet hat, *obwohl* er/sie sie gestellt bekommen hat oder *weil* er/sie sie *nicht* gestellt bekommen hat und somit gar nicht beantworten konnte. D.h. diese Anzahl der Teilnehmer ohne Antwort sollte – abweichend von den Statistiken zu den gegebenen Antworten – *nicht* mit der Anzahl der Teilnehmer verglichen werden, denen die Frage gestellt wurde, sondern immer mit der Anzahl *aller* Frageneinsender. Das wird in obigem Beispiel durch den Zusatz `|.*%s` erreicht: In `%s` wird ja die gegebene Antwort (hier ein Buchstabe) eingesetzt, und für jede „echte" (nicht leere) Antwort sucht der reguläre Ausdruck dann also, ob in der Aufzählung der gewählten Fragen-Nummern die Nummer dieser Frage vorkommt *oder* ob die Aufzählung mit dem Antwortbuchstaben endet. Da letzteres niemals der Fall ist, ist dieser „Oder-Zusatz" für „echte" Antworten also wirkungslos. Anders sieht es aus für die Statistik, wieviele Teilnehmer gar keine Antwort gegeben haben. Dort wird nämlich der `%s`-Platzhalter durch den leeren String ersetzt (also praktisch ersatzlos entfernt), womit der Suchausdruck hier also auf „oder beliebiger Text" endet und somit *immer* zutrifft, egal was im `FeldA1` steht. D.h. als Vergleichswert zur Anzahl der leeren Antworten (und nur dieser) werden hier tatsächlich *alle* Einsender gezählt, unabhängig von der Fragenauswahl.

Die Statistik dazu kann dann z.B. wie folgt aussehen:

Shuffle Rows

Pünktchen und …

Gezählte Antworten: 4 von 5 Heftteilnehmern

ML	Antwort	Häufigkeit (max: 3)	von *)	Prozent *)	Prozent gesamt **)	Trennschärfe
ja	A	3	4	75,0	75,0	0,49
nein	B	1	4	25,0	25,0	-0,49

*) Anzahl Teilnehmer/-innen, denen diese Fragezeile nach Randomisierung angeboten wurde
**) Bezogen auf die Gesamtanzahl gezählter Antworten, siehe oben

Die dicke …

Gezählte Antworten: 4 von 5 Heftteilnehmern

ML	Antwort	Häufigkeit (max: 3)	von *)	Prozent *)	Prozent gesamt **)	Trennschärfe
nein	–	3	4	75,0	75,0	-0,29
ja	B	1	1	100,0	25,0	–

*) Anzahl Teilnehmer/-innen, denen diese Fragezeile nach Randomisierung angeboten wurde
**) Bezogen auf die Gesamtanzahl gezählter Antworten, siehe oben

Gaius Julius …

Gezählte Antworten: 4 von 5 Heftteilnehmern

ML	Antwort	Häufigkeit (max: 2)	von *)	Prozent *)	Prozent gesamt **)	Trennschärfe
nein	–	2	4	50,0	50,0	-0,27
nein	A	1	3	33,3	25,0	-0,43
ja	C	1	3	33,3	25,0	1,00

*) Anzahl Teilnehmer/-innen, denen diese Fragezeile nach Randomisierung angeboten wurde
**) Bezogen auf die Gesamtanzahl gezählter Antworten, siehe oben

Statistik mit filter-Angabe zu Zuordnungs-Frage

In dieser Statistik sehen wir, dass es insgesamt 5 Heftteilnehmer/-innen gab, von denen nur 4 diese Zuordnungsaufgabe bearbeitet haben.

1. Dabei wurde die erste Frage („Pünktchen und …“) zufällig allen vier Teilnehmern/-innen gestellt und 3 davon haben sie richtig beantwortet.
2. Die zweite Frage („Die dicke …“) wurde nur einem/-r Teilnehmer/-in gestellt, der/die sie richtig beantwortet hat. Die anderen Drei haben keine Antwort gegeben (und konnten es auch gar nicht). Beachten Sie, dass – wie oben beschrieben – in der Zeile für keine Antwort (beschriftet mit „—“) daher in Spalte „von“ die Gesamtteilnehmerzahl steht und nicht die Anzahl 1 der „Fragenteilnehmer“. Die Angabe, dass 3 von 1 die Frage nicht beantwortet hätten, ergäbe auch keinen Sinn.
3. Die dritte Frage („Gaius Julius …“) wurde entsprechend den übrigen drei Teilnehmer/-innen gestellt. Eine/-r davon hat diese richtig, eine/-r weitere/-r hat sie falsch beantwortet. Demnach hat der/die Dritte sie gar nicht beantwortet. Insgesamt haben 2 von 4 Aufgabenteilnehmern/-

innen die Frage also nicht beantwortet: Eine/-r, obwohl ihm/ihr die Frage gestellt wurde, eine/-r, weil sie ihm/ihr gar nicht gestellt wurde (sondern statt dessen die zweite Frage). Hier können Sie gut erkennen, dass die Spalte „von“ in allen Zeilen zu „echten“ Antworten (hier „A“ und „C“) also die Anzahl 3 der „Fragenteilnehmer“ enthält, die Zeile „—“ derjenigen ohne Antwort als einzige Tabellenzeile wieder die Anzahl 4 der Aufgabeneinsender.

`trennschaerfe`

Falls der Aufgabenstatistik auch eine *Trennschärfe*-Spalte hinzugefügt werden soll, fügen Sie hinter dem letzten `feld`-Element ein Kindelement namens `trennschaerfe` ein. In der Regel genügt ein einfaches „Marker“-Element ohne Attribute und ohne Inhalt. In dem Fall kann ein Benutzer die Parameter zum Vergleichsumfang der Trennschärfemessung in der Benutzeroberfläche einstellen und variieren. Falls Sie die Trennschärfeparameter dagegen selbst konstant festlegen möchten, fügen Sie dem Element die zwei Attribute `vergleich` und `unbearbeiteteAufg` hinzu:

- Für `vergleich` sind die beiden Werte `kurs` und `heft` zugelassen. Mit dem Attribut `vergleich="kurs"` legen Sie fest, dass die Antwort (1 für von einem Studenten gegeben, 0 für nicht gegeben) in Relation zur Leistung (erreichten Punkten) desselben Studenten in allen anderen Aufgaben desselben Kurses gesetzt werden soll, mit `vergleich="heft"` schränken Sie den Vergleichsumfang auf die Leistungen in den restlichen Aufgaben desselben Aufgabenhefts ein.
- `unbearbeiteteAufg` ist vom Typ `xsd:boolean`. `unbearbeiteteAufg="true"` gibt an, dass bei der Ermittlung der Vergleichsleistung jedes Studenten die von ihm nicht bearbeiteten Aufgaben (aus dem Vergleichsumfang, s.o.) als 0-Punkte-Leistung gewertet werden sollen. Mit Wert `false` dagegen werden unbearbeitete Aufgabe ignoriert, also gar nicht in die Vergleichsleistung einbezogen.

Von entsprechend parametrisierten `trennschaerfe`-Elementen können Sie auch mehr als eines angeben (dann mit abweichenden Attributen), um entsprechend mehrere unterschiedlich konfigurierte Trennschärfemaße als weitere Spalten zur Statistiktabelle hinzuzufügen.

(Mehr zur Trennschärfe finden Sie in den Online-Hilfeseiten, wenn Sie eine Antworthäufigkeitsstatistik öffnen.)

`muster`

Nach der Angabe der `feld`- und ggf. `trennschaerfe`-Elemente ist die eigentliche Statistikdefinition fertig. Sie können jedoch optional noch eine Musterlösung[24] zur Frage hinterlegen. Wenn keine Musterlösung angegeben wird, werden zwar alle Antworten gezählt, aber nur bei Angabe einer Musterlösung, können die einzelnen gegebenen Antworten in der Statistikausgabe zusätzlich nach „richtig“ oder „falsch“ klassifiziert werden (siehe Screenshot am Beginn dieses Abschnitts). Auch eine „Wertung“ der Trennschärfe mit einer farbigen Tortengrafik wie in obiger Abbildung setzt eine Musterlösung voraus, weil für korrekte Antworten positive, für falsche Antworten dagegen negative Korrelationskoeffizienten als „gut“ interpretiert werden.

Weiterhin werden ohne Angabe von Lösungen grundsätzlich alle unterschiedlichen Eingaben einzeln gezählt. Geben Sie dagegen ein *Lösungsmuster* für verschiedene Schreibweisen eines bestimmten Lösungsbegriffs (z.B. als regulären Ausdruck oder Musterintervall) an, so werden alle Eingaben, die auf dieses Muster passen, zu einem einzigen Statistikeintrag zusammengefasst und nicht mehr einzeln gezählt.

Die Angabe der Musterlösungen erfolgt in der Regel durch ein oder mehrere `muster`-Elemente

hinter dem/den `feld`-Element(en). Jede mögliche Antwort, die in der Statistik als „richtig“ zu markieren ist, ist in einem `muster`-Element anzugeben, jede andere Antwort wird dann als falsch markiert. (Wenn gar kein `muster`-Element existiert, entfällt die komplette Ausgabe der »ML«-Spalte mit den richtig-/falsch-Markierungen.)

Beispiele für solche `muster`-Elemente finden Sie in mehreren vorangegangenen XML-Listings. Beachten Sie beim Multiple-Choice-Beispiel oben, dass für die Statistik mit den Einzelantworten (vgl. `trenner=","`-Attribut) auch die korrekten Einzelantworten jeweils als `muster` angegeben werden, während bei der Statistik für die kombinierten Antworten in obigem Fall nur ein einziges `muster`-Element existiert, das die einzig vollständig korrekte Antwortkombination aufzählt. (Falls mehrere Kombinationen als korrekt gelten, könnten aber auch dann entsprechend mehrere `muster`-Elemente aufgeführt werden.)

Im Normalfall muss eine Eingabe exakt mit dem Inhalt eines `muster`-Elements übereinstimmen, um als korrekte Antwort gewertet zu werden, es gibt jedoch auch Einstellungen, mit denen ein „Lösungsmuster“ an Stelle eines exakt einzugebenden Lösungsbegriffs festgelegt werden kann:

- Wenn im `statistik`-Element das Attribut `ignore-case="true"` angegeben wird, muss die Groß-/Kleinschreibung der Eingaben nicht mehr mit der des Muster-Elements übereinstimmen. Das Ignore-Case-Attribut lässt sich auch mit den beiden folgenden Attributen kombinieren:
- Wenn im `statistik`-Element das Attribut `uebereinstimmung` angegeben wurde und einen Wert < 100 hat, muss keine hundertprozentige, sondern nur noch anteilige Übereinstimmung vorliegen.
- Wenn im `statistik`-Element das Attribut `regex="true"` angegeben wurde, wird der Inhalt des `muster`-Element nicht mehr als String-Literal interpretiert, mit dem eine Eingabe exakt oder oder anteilig übereinstimmen muss, sondern als regulärer Ausdruck <https://www.regular-expressions.info> , also eine formale Beschreibung eines Textmusters, auf das eine Eingabe passen muss. (Zu regulären Ausdrücken siehe auch den Abschnitt zur Bewerterkonfiguration `Begriffe`.)
- Weitergehende Möglichkeiten, um zusammengehörige, aber nicht gleiche „richtige“ Einsendungen in der Statistik zusammenzufassen, sind der Einsatz von Synonymen (siehe folgenden Abschnitt) und speziell für Eingaben von Zahlen aus einem Lösungsintervall die `musterintervall`-Elemente, die später in diesem Abschnitt vorgestellt werden, optional kombiniert mit dem `zahl-aus-text`-Attribut im `statistik`-Element.

(Siehe vorangegangenen Abschnitt Attribute von `statistik` für mehr Details zu den genannten Attributen.)

In all diesen Fällen werden alle zu dem entsprechenden „Lösungsmuster“ hinreichend passenden Antworten zu einem einzigen Statistikeintrag zusammengefasst, auch wenn sie nicht hundertprozentig übereinstimmen. Dieser Statistikeintrag wird dann standardmäßig mit dem Inhalt des `muster`-Elements beschriftet.

Beispiel: Nehmen Sie folgenden Ausschnitt eines Statistik-Elements mit (mindestens) dem einen folgenden Muster-Element an:

```
<statistik ignore-case="true" uebereinstimmung="90">
  …
  <muster>FernUniversität in Hagen</muster>
</statistik>
```

In diesem Fall werden alle auch Antworten, die in der Groß-/Kleinschreibung beliebig abweichen und darüber hinaus ggf. auch bis zu 10% vom Vorgabetext abweichen, also z.B. auch die Eingabe »fernuniversitaet in hagen«, als dazu passende Antworten gezählt und in der Statistik mit unter dem Eintrag »FernUniversität in Hagen« aufsummiert.

Ein ähnliches Beispiel wäre eine Statistik wie folgt:

```
<statistik regex="true" ignore-case="false" …>
  …
  <muster>Fern[Uu]ni(versit(ä|ae)t)? (in )?Hagen</muster>
</statistik>
```

In diesem Fall würden alle Antworten wie »FernUniversität in Hagen«, »FernUniversitaet Hagen« oder »Fernuni Hagen« als auf obiges Muster passende Antworten erkannt[25] und in der Statistik unter dem Eintrag »Fern[Uu]ni(versit(ä|ae)t)? (in)?Hagen« zusammengefasst. (Bei `ignore-case="true"` wären auch noch Einsendungen in anderer Groß-/Kleinschreibung darunter.)

Ein solcher regulärer Ausdruck als Anzeigetext für einen Statistikeintrag ist allerdings nicht sonderlich hübsch oder gut lesbar. Falls Sie es vorziehen, dass der Statistikeintrag, der alle oben akzeptierten Schreibweisen zusammenfasst (also die Anzahl aller Einsendungen bezeichnet, die auf den unter `muster` angegebenen regulären Ausdruck <https://www.regular-expressions.info> passen), lieber einfach nur mit einem gut lesbaren Stellvertretertext wie »FernUniversität in Hagen« beschriftet werden soll, dann geben Sie im Muster-Element die Alternativbeschriftung im optionalen Attribut `anzeige` mit an:

```
  <muster anzeige="FernUniversität in Hagen">Fern[Uu]ni(versit(ä|ae))t)? (in )?
Hagen</muster>
```

(Hinweis am Rande: Das `anzeige`-Attribut ist analog auch noch in den Elementen `synonyme` und `synonym` definiert und erfüllt dort denselben Zweck, also einen alternativen Beschriftungstext für einen Statistikeintrag festzulegen.)

musterintervall

Bei `musterintervall` handelt es sich um eine Alternative zu `muster`-Elementen speziell zur Erkennung und Zusammenfassung *numerischer* Eingaben:

Falls die möglichen Antworten keine Strings sind, sondern Zahlen, die in einem bestimmten Lösungsintervall liegen müssen, um als korrekt gewertet zu werden (vgl. obige Ausführungen zu den Attributen `nachkommastellen` und `zahl-aus-text`), dann können Sie das bzw. die zugehörige(n) Lösungsintervall(e) über ein bzw. mehrere `musterintervall`-Elemente angeben. (`muster`- und `musterintervall`-Elemente sind nicht kombinierbar, d.h. Sie können nur *entweder* `muster`- *oder* `musterintervall`-Elemente innerhalb desselben `statistik`-Elements angeben.)

Ein Musterintervall hat zunächst zwei Pflichtattribute `von` und `bis`, in denen die Intervallgrenzen anzugeben sind. Per Default sind diese implizit Teil des Intervalls, jedoch kann über die optionale Angabe der Boolean-Attribute `linksoffen` bzw. `rechtsoffen` explizit gesteuert werden, ob diese Intervallgrenzen noch zum Intervall gehören sollen oder nicht.

Das Lösungsintervall `"[0,0..9,5["`, also die Menge $\{x \in \mathbb{R} \mid x \geq 0{,}0 \wedge x < 9{,}5\}$, könnten Sie auf

folgende Weise darstellen:

```
<musterintervall von="0.0" bis="9.5" linksoffen="false" rechtsoffen="true"/>
```

Dabei ist die Angabe von `linksoffen="false"` nicht nötig (da false ohnehin der Defaultwert ist), macht in diesem Fall aber das XML besser lesbar, da so die Kenntnis über den Defaultwert nicht vorausgesetzt wird.

Als Beispiel sei das `statistik`-Element einer Frage angegeben, die nach einer Zahl fragt, für die gilt, dass das Ergebnis einer Rundung auf eine ganze Zahl eine einstellige Zahl ≠ 0 ist. Die Eingabe muss also eine Zahl ≥ 0,5 und < 9,5 sein, wahlweise auch mit negativem Vorzeichen. Eingaben mit mehr als einer Nachkommastelle sollen auf dabei in der Statistik auf eine Nachkommastelle gerundet werden, Eingaben von insgesamt mehr als 25 Zeichen werden nicht berücksichtigt:

```
<statistik maxlaenge="25" nachkommastellen="1">
    <titel>Zahlen</titel>
    <feld>A10</feld>
    <musterintervall von="-9.5" bis="-0.5" linksoffen="true"
rechtsoffen="false"/>
    <musterintervall von="0.5" bis="9.5" linksoffen="false"
rechtsoffen="true"/>
</statistik>
```

synonyme

Im Anschluss an eine Folge von `muster`-Elementen kann sich bei Bedarf eine Folge von `synonyme`-Elementen anschließen (nicht anwendbar nach `musterintervall`en).

Synonyme werden insbesondere beim integrierten Begriffs- bzw. Begriffsfolge-Bewerter verwendet. Allgemein sind Synonyme mögliche Antworten, die als synonym, also gleichbedeutend mit Musterbegriffen angesehen werden.

Beispiel: Angenommen, Sie haben eine Aufgabe gestellt, in der die Eingabe des Namens des Bundeslandes Nordrhein-Westfalen erwartet wird. In der Statistik-Definition haben Sie bereits `<muster>Nordrhein-Westfalen</muster>` eingetragen, damit dieser Begriff in der Statistik als korrekte Antwort markiert wird. Nun soll die Aufgabe aber auch die Eingabe »NRW« als korrekt akzeptieren. Wie stellen Sie das in der Statistik dar?

1. Sie könnten die Statistikdefinition dazu um einen weiteren Eintrag `<muster>NRW</muster>` ergänzen. Dann werden alle Eingaben von `Nordrhein-Westfalen` und alle Eingaben von `NRW` in der Statistik *getrennt* gezählt, d.h. Ihnen wird aufgeschlüsselt, wieviele Studenten den ausgeschriebenen Namen und wieviele die Abkürzung eingegeben haben. Allerdings enthält die Statistik dann *keine Gesamtsumme*, also wieviele Studenten die Frage korrekt (mit irgendeiner der beiden Schreibweisen) beantwortet haben.
2. Wenn Sie eher an dieser *Gesamtsumme* als an einer Einzelaufschlüsselung interessiert sind, dann legen Sie »NRW« nicht als separate Musterlösung, sondern als Synonym für »Nordrhein-Westfalen« fest, und zwar wie folgt:

```
<muster>Nordrhein-Westfalen</muster>
…
<synonyme fuer="Nordrhein-Westfalen">
    <synonym>NRW</synonym>
</synonyme>
```

Mit dieser Definition werden alle Eingaben von »NRW« unter dem Statistikeintrag »Nordrhein-Westfalen« mitgezählt. D.h. falls z.B. 18 Teilnehmer die Antwort »Nordrhein-Westfalen« eingeben und 7 Teilnehmer die Antwort »NRW« eingeben, wird die Statistik in einer Tabellenzeile »Nordrhein-Westfalen« die Summe 25 aufführen (und keinen separaten Eintrag für »NRW« aufweisen).

Beachten Sie, dass *erst alle* `muster`-Elemente aufzuzählen sind, und *danach* alle `synonyme`-Elemente folgen dürfen, d.h. `muster`- und `synonyme`-Elemente dürfen nicht vermischt im XML stehen.

3. Wenn Sie an *beidem* (Gesamtsummen pro Begriff als auch Einzelaufschlüsselung) interessiert sind, erstellen Sie dazu *zwei* Statistiken, eine mit Synonymdefinitionen und eine mit separaten `muster`-Einträgen pro Synonym. Tyischerweise geben Sie beiden denselben Titel (geben also nur bei der ersten das `titel`-Element an) und unterschiedliche Untertitel, analog zu den beiden separaten Statistiken zu Multiple-Choice-Fragen, wie sie zu Beginn dieses Abschnitts beispielhaft (auch mit Screenshot) gezeigt wurden.

Allgemein sind **Synonym-Definitionen** wie folgt aufgebaut:

Ein `synonyme`-Element besitzt genau ein *Pflichtattribut* `fuer`, in welchem der Inhalt des `muster`-Elements (bzw. seine `id`) wiederholt wird, für das die Synonyme festgelegt werden sollen. Weiterhin hat ein `synonyme`-Element ein oder mehrere *Kindelemente* `synonym` oder `intervall` (beliebig gemischt).

Falls Sie statt „atomarer" `muster`-Elemente auf die im Folgenden noch beschriebenen `musterfolge`- oder `mustermenge`-Konstruktionen in Verbindung mit Statistik-Attribut `aufzaehlen` oder `sortiert-aufzaehlen` zurückgreifen, ist im `fuer`-Attribut des `synonyme`-Tags (und in den Kindelementen) jeweils eine entsprechende zusammengesetzte Aufzählung von Musterteilbegriffen anzugeben. Darauf wird auch im Folgenden noch genauer eingegangen.

Genau in letzterem Kontext, wenn also der Wert des `fuer`-Attributs nicht 1:1 mit dem Wert eines `muster`-Elements übereinstimmt, und Sie aber auch den Wert des `fuer`-Attributs nicht 1:1 in der Form als Anzeigetext des entsprechenden Statistikeintrags sehen möchten (weil darin vielleicht schwer lesbare reguläre Ausdrücke vorkommen), dann können Sie auch im `synonyme`-Element zusätzlich das optionale Attribut `anzeige` angeben, das eben eine alternative Beschriftung für die Statistiktabelle festlegt. Im Normalfall wird dieses Attribut jedoch nicht benötigt.

`synonym`-Kindelemente sind wiederum einfache String-Elemente und legen Text-Synonyme für die Musterlösung fest. Insgesamt wird also mit einer solchen Konstruktion aus einem `synonyme`-Element mit einem oder mehreren `synonym`-Kindern ausgedrückt, dass jeder von einem Teilnehmer im jeweiligen Eingabefeld genannte Begriff, der mit einem der Synonyme übereinstimmt, nicht gesondert in der Statistik gezählt werden soll, sondern wie eine Nennung desjenigen Begriffs, der im `fuer`-Attribut steht.

Analog zu `muster`-Elementen können auch `synonym`-Elemente um ein `anzeige`-Attribut ergänzt werden, das in der Statistik an Stelle des eigentlichen Elementinhalts angezeigt werden soll, z.B. falls letzterer ein regulärer Ausdruck ist.

Textsynonyme werden *normalerweise* wörtlich (zeichenweise) mit einer Eingabe verglichen (wobei ggf. über die Attribute `case-sensitive`, `regex` und `uebereinstimmung` des `statistik`-Elements gewisse „Lockerungen“ für den Textvergleich festgelegt werden können, wie schon unter `muster` beschrieben). So wie beim internen Begriffs- und Begriffsfolgen-Bewerter (im so genannten Smart-Modus) *alternativ* zum Textvergleich auch ein Zahlenvergleich möglich ist, so kann auch hier in der Statistikdefinition ein Synonym alternativ als Zahl oder sogar Zahlenintervall eingetragen werden, indem Kindelement `intervall` ins `synonyme`-Element aufgenommen wird. Der Aufbau ist analog zum `musterintervall`-Element, das oben beschrieben wurde, nur legt dieses Element eben keine eigenständige Musterlösung für eine Zahlenfrage fest, sondern ein Synonym für eine andere Musterlösung.

Beispiel: In einer Aufgabe sei nach der Zahl 42 gefragt, und diese soll wahlweise als Text „Zweiundvierzig“ oder auch englisch „forty-two“ eingegeben werden können oder auch als Zahl „42“, von der wiederum auch alternative Schreibweisen wie „+42“, „42,0“, „42.0“ etc. akzeptiert werden sollen. Damit Sie nicht für jede erdenkliche gleichwertige Schreibweise ein Textsynonym festlegen müssen, kann hier ein einelementiges Intervall (das genau die Zahl 42 enthält) als Synonym festgelegt werden.

Die Statistikdefinition dazu könnte dann z.B. wie folgt aussehen:

```
<statistik maxlaenge="20" case-sensitive="false" uebereinstimmung="80">
    <untertitel>Dritter Begriff</untertitel>
    <feld>A10</feld>
    <trennschaerfe/>
    <muster>Zweiundvierzig</muster>
    <synonyme fuer="Zweiundvierzig">
        <intervall von="42.0" bis="42.0" linksoffen="false"
rechtsoffen="false"/>
        <synonym>forty-two</synonym>
    </synonyme>
</statistik>
```

In der Statistik würden dann alle Eingaben wie „zweiUndVierzig“ (Groß-/Kleinschreibung wird ignoriert), „fourty-two“ (stimmt in 9 von 10 Zeichen und damit zu über 80% mit dem Muster-Synonym überein), „42,0“, „4.2E1“ (Kurzschreibweise für $4{,}2 \cdot 10^1$) etc. sämtlich als Synonym zu „Zweiundvierzig“ erkannt und gemeinsam in einer Statistikzeile „Zweiundvierzig“ zusammengefasst, also nicht einzeln als gegebene Antworten aufgeführt.

Hinweis: Synonyme können nur mit `muster`-Elementen kombiniert werden (oder den im Folgenden noch beschriebenen `musterfolge`-Elementen) und stehen *nicht* für Statistiken zur Verfügung, die statt dessen auf `musterintervall`-Elemente setzen.

Insbesondere können Intervalle lediglich Synonyme für eine Textantwort sein. Wenn Sie zu einem Eingabefeld, zu dem nach einer Zahl gefragt wird, mehrere Intervalle nicht als separate Muster-Intervalle vorgeben möchten, sondern nur ein Musterintervall und weitere Intervalle als Synonyme fürs erste Intervall (so dass die Eingaben pro Intervall in der Statistik nicht einzeln separat gezählt, sondern zusammengefasst werden), dann gehen Sie wie folgt vor:

Erstellen Sie für das Eingabefeld ein `muster`-Element mit einem „Dummy-Text“, der das Lösungsintervall ggf. samt seiner „Synonym-Intervalle“ beschreibt, sowie einem Attribut `id`, in dem Sie einen ggf. kürzeren Text eingeben (der nicht als mögliche Eingabe erwartet wird). Erstellen Sie dazu dann ein `synonyme`-Element, in dessen `fuer`-Attribut Sie denselben Text eintragen wie ins `id`-Attribut des `muster`-Elements. In diesem `synonyme`-Element tragen Sie sämtliche

Lösungsintervalle per `intervall`-Elemente ein.

Das `id`-Attribut des `muster`-Elements dient zwei verschiedenen Zwecken:

1. Es ermöglicht Ihnen, den „Dummy-Text“, der hier die Zusammenfassung der Synonym-Intervalle beschreiben soll, nicht doppelt im XML (also nicht ins `fuer`-Attribut des `synonyme`-Elements) eintragen zu müssen. So ist der Text im Zweifel leicht nachträglich änderbar, ohne diese Referenz im `fuer`-Attribut immer mit anpassen zu müssen.
2. Es bewirkt, dass in der Statistik *ausschließlich* der von Ihnen festgelegte „Dummy-Text“ angezeigt wird. Für `muster`-Elemente *ohne* `id`-Attribut, aber *mit* Synonymen, gilt dagegen, dass das Online-Übungssystem in der Statistiktabelle hinter dem Muster-Begriff selbst auch noch die bekannten Synonyme mit aufzählt. Das unterbleibt bei Angabe eines `id`-Attributs.

Beispiel-Ausschnitt für eine Musterlösung, die „nur“ aus zwei Intervallen besteht:

```
…
<muster id="intervalle_0_10">]0;10] oder [-10;0[</muster> <!-- Textinhalt frei
wählbar  -->
…
<synonyme fuer="intervalle_0_10">
    <intervall von="0.0" bis="10.0" linksoffen="true" rechtsoffen="false"/>
    <intervall von="-10.0" bis="0.0" linksoffen="false" rechtsoffen="true"/>
</synonyme>
```

Der Beschriftungstext („Dummy-Text“) in obigem Muster-Element ist frei wählbar, könnte z.B. alternativ auch lauten „-10 bis 10 ohne Null“ o.ä.. Auch die ID ist frei wählbar, muss aber im `id`-Attribut und `fuer`-Attribut jeweils übereinstimmen, und sie sollte so gewählt sein, dass mit sehr hoher Wahrscheinlichkeit kein Student genau diesen Text (statt einer Zahl) eintippt, denn solche Eingaben würden dann u.U. unter diesem Statistik-Posten mitgezählt.

musterfolge / mustermenge / formatfolge

Wenn Sie mehrere gleichnamige Eingabefelder mit `data-separator`-Attribut im Aufgabenformular verwenden, geben Sie in der Statistik zunächst einmal dazu passend jeweils ein `trenner`-Attribut (mit demselben Trennzeichen wie in `data-separator`) an, dazu ein `escape`-Attribut (sofern der Separator genau ein Zeichen ist) und zwar `escape="/"`, falls `trenner="\"` und ansonsten `escape="\"` für alle anderen (einstelligen) Trennzeichen.

Dann haben Sie mit den oben stehenden Mitteln zunächst folgende Möglichkeiten:

1. Ohne weitere Angaben trennen Sie die Begriffe in ihre Teilbegriffe auf und zählen also in der Statistik die Vorkommen der einzelnen Teilbegriffe. Über `muster`- oder `musterintervall`-Elemente können Sie festlegen, welche dieser Teilbegriffe als korrekt oder nicht korrekt in der Statistik markiert werden sollen.
2. Sie können zusätzlich zu `trenner` und `escape` ein `aufzaehlen`- oder `sortiert-aufzaehlen`-Attribut angeben, um diese Begriffe in einer „normierten“ Form wieder zusammenzufügen und diese dann zählen lassen. Auch dazu können Sie mittels `muster`-Elementen (in derselben normierten Schreibweise) entsprechend markieren, welche solche Teilbegriffsfolgen in der Statistik als korrekt/Musterlösung markiert werden sollen.

Wenn das aber beides nicht genügt, weil Sie zwar eine Zusammensetzung der Begriffe zählen möchten (wie in 2.), jedoch z.B. die Toleranzeinstellung `uebereinstimmung` nicht auf den zusammengesetzten Begriff anwenden möchten, sondern auf seine Teile, oder wenn Sie den

`regex`-Modus benutzen und daher nicht der zusammengesetzte Begriff als ein einziger regulärer Ausdruck interpretiert werden soll, sondern als eine Aufzählung einzelner regulärer Ausdrücke für Teilbegriffe, oder wenn Teilworte mit einem `musterintervall` verglichen werden sollten, was in einem aus mehreren Teilworten zusammengesetzten `muster`-String nicht möglich ist, dann haben Sie (zusätzlich zur Angabe der `trenner`- und `escape`-Attribute sowie eines `sortiert-aufzaehlen`- oder `aufzaehlen`-Elements) folgende Möglichkeiten:

Über `musterfolge`- oder `mustermenge`-Elemente können Sie flexibler festlegen, welche Zusammensetzungen von Teilbegriffen in der Statistik als korrekt / Musterlösung markiert werden sollen:

Ein `musterfolge`-Element umfasst stets eine Folge von Kindelementen, und zwar `muster`- und/oder `musterintervall`-Elemente. Eine wieder zusammengesetzte Einsendung wird dann als korrekt markiert, wenn (vor der Wiederzusammenfügung) die einzelnen Teileinsendungen jeweils auf die entsprechenden Musterstrings oder Musterintervalle passen, und zwar bei Verwendung des `aufzaehlen`-Attributs in der Reihenfolge, in der sie auch vom Studenten genannt wurden, bzw. bei Verwendung des `sortiert-aufzaehlen`-Attributs werden die Teileingaben erst sortiert und dann mit der Musterfolge verglichen.

`musterfolge`-Elemente können nur *an Stelle von* `muster`-Elementen verwendet und nicht mit diesen vermischt werden.

Es ist möglich, hinter `musterfolge`-Elementen noch `synonyme`-Elemente anzugeben. Die funktionieren dann aber nicht auf Teilbegriffsebene, sondern nur für die abschließenden Statistikeinträge, also bei Verwenden von des `aufzaehlen`- oder `sortiert-aufzaehlen`-Attributs nur für die wieder zusammengefügten / neu aufgezählten Zeichenketten. Dabei ist dann auch zu beachten, dass hier – anders als bei der einfachen Kombination von `muster` und `synonym`-Elementen – die `musterfolge`-Elemente für alle Schreibweisen, also auch die Synonyme, erstellt werden müssen.

Beispiel: Angenommen, es wurde nach einem von zwei möglichen Begriffen gefragt, und zwar als Abfolge zweier Teilbegriffe (in zwei separaten Eingabefeldern): Im ersten Feld ist das Wort „positiv“ oder „negativ“ einzutragen, im zweiten dann entweder eine positive Zahl ≤10 bzw. eine negative Zahl ≥ -10. Die Statistikdefinition dazu könnte dann wie folgt aussehen:

```
<antwortstatistiken xsi:noNamespaceSchemaLocation="https://online-
uebungssystem.fernuni-hagen.de/download/antwortstatistik.xsd">
  <statistik maxlaenge="27" trenner="/" escape="\" aufzaehlen=" / " case-
sensitive="false">
    <titel>Test</titel>
    <feld>A1</feld>
    <trennschaerfe/>
    <musterfolge>
      <muster>positiv</muster>
      <musterintervall von="0.0" bis="10.0" linksoffen="true"
rechtsoffen="false"/>
    </musterfolge>
    <musterfolge>
      <muster>negativ</muster>
      <musterintervall von="-10.0" bis="0.0" linksoffen="false"
rechtsoffen="true"/>
    </musterfolge>
  </statistik>
</antwortstatistiken>
```

Weiteres Beispiel: Angenommen, es wurde nach der Eingabe zweier von drei Begriffen gefragt, in beliebiger Reihenfolge: Richtig seien alle Antwortkombinationen zweier der drei Worte „Anton“, „Berta“ und „Caesar“ oder alternativ alle Kombinationen zweier der drei Worte „A“, „B“ und „C“, welche in der Statistik nicht separat, sondern als Synonyme zu den ersteren gezählt werden sollen. Mit `musterfolge`-Elementen könnte eine Statistikdefinition dazu wie folgt aussehen. **Hinweis: Dieses Beispiel ist noch nicht der Optimalfall, sondern eher ein Negativbeispiel.** Im Folgenden wird eine bessere Lösung mit Hilfe von `mustermenge` gezeigt.

```
<statistik maxlaenge="28" case-sensitive="false" uebereinstimmung="75"
trenner="/" sortiert-aufzaehlen=", " escape="\">
    <titel>Nicht ideal!</titel>
    <feld>A1</feld>
    <feld>A2</feld>
    <trennschaerfe/>
    <musterfolge>
        <muster>Anton</muster>
        <muster>Berta</muster>
    </musterfolge>
    <musterfolge>
        <muster>Anton</muster>
        <muster>Caesar</muster>
    </musterfolge>
    <musterfolge>
        <muster>Berta</muster>
        <muster>Caesar</muster>
    </musterfolge>
    <musterfolge>
        <muster>A</muster>
        <muster>B</muster>
    </musterfolge>
    <musterfolge>
        <muster>A</muster>
        <muster>C</muster>
    </musterfolge>
    <musterfolge>
        <muster>B</muster>
        <muster>C</muster>
    </musterfolge>
    <synonyme fuer="Anton, Berta">
        <synonym>A, B</synonym>
    </synonyme>
    <synonyme fuer="Anton, Caesar">
        <synonym>A, C</synonym>
    </synonyme>
    <synonyme fuer="Berta, Caesar">
        <synonym>B, C</synonym>
    </synonyme>
</statistik>
```

Durch die Kombination mit `sortiert-aufzaehlen` reduziert sich die Anzahl der möglichen Muster-Kombinationen zwar schon deutlich, in der Statistik werden Eingabe wie „A/B“ und „B/A“ immer als „A, B“ zusammengefasst. Dennoch müssen immer noch viele verschiedene Musterfolgen einzeln notiert werden. Außerdem funktioniert dieses Vorgehen mit der alphabetischen Sortierung auch nur, wenn die Sortierung aller akzeptierten Eingaben und der zugehörigen Musterlösung immer gleich ist. Da hier auch Teilübereinstimmungen von 75% akzeptiert werden, könnte z.B. prinzipiell auch die Antwort „Berta/Aaesar“ akzeptiert werden (da „Aaesar“ zu mehr als 75% mit „Caesar“ übereinstimmt). Da allerdings nach der Sortierung und Neu-Aufzählung die Eingabe zu „Aaesar, Berta“ konvertiert würde, was nicht mehr zur Musterfolge „Berta, Caesar“ passt, funktioniert hier die Erkennung der Eingabe und Zusammenfassung mit Statistikeintrag „Berta, Caesar“ nicht mehr.

Besser geeignet für diese Frage nach einer Teilmenge von Teilbegriffen ist daher die Verwendung des `mustermenge`-Elements an Stelle von `musterfolge`. Die beiden Elemente unterscheiden sich nicht im Aufbau, nur in der Bedeutung: Eine Eingabe wird dann als Musterlösung erkannt, wenn

jedes eingegebene Teilwort zu einem Element der Mustermenge passt. Die Eingabe-Reihenfolge ist dabei egal, und die Mustermenge darf auch mehr Elemente enthalten als die Eingabefolgen (hier also alle drei Namen, obwohl nur zwei einzugeben sind). Damit lässt sich obiges Beispiel wie folgt verbessern:

```
<statistik maxlaenge="28" case-sensitive="false" uebereinstimmung="75"
trenner="/" sortiert-aufzaehlen=", " escape="\">
    <titel>Besser</titel>
    <feld>A1</feld>
    <feld>A2</feld>
    <trennschaerfe/>
    <mustermenge>
        <muster>Anton</muster>
        <muster>Berta</muster>
        <muster>Caesar</muster>
    </mustermenge>
    <mustermenge>
        <muster>A</muster>
        <muster>B</muster>
        <muster>C</muster>
    </mustermenge>
    <synonyme fuer="Anton, Berta">
        <synonym>A, B</synonym>
    </synonyme>
    <synonyme fuer="Anton, Caesar">
        <synonym>A, C</synonym>
    </synonyme>
    <synonyme fuer="Berta, Caesar">
        <synonym>B, C</synonym>
    </synonyme>
</statistik>
```

Die zuvor betrachtete Beispieleingabe „Berta / Aaesar“ würde hier als zur Mustermenge {„Anton“, „Berta“, „Caesar“} passend bewertet (da jeder der beiden Teileingaben „Berta“ und „Aaesar“ zu jeweils mehr als 75% mit zwei verschiedenen Wörtern dieser Menge übereinstimmen) und daher dann mit zum Statistikeintrag „Berta, Caesar (oder A,C)“ hinzugezählt. Dass die „sortiert-aufzaehlen“-Option daraus eigentlich den Statistikeintrag „Aaerta, Berta“ machen würde, spielt hier aufgrund der Angleichung an einen Mustereintrag keine Rolle mehr. (Eine falsche Eingabe wie „Hallo / Aaesar“ dagegen, die auf keine der Mustermengen passt, stünde nach alphabetischer Neusortierung in der Statistik als »Aaesar, Hallo“ aufgeführt.)

Beachten Sie, dass die Erkennung, *ob* eine Antwortkombination eine korrekte oder falsche Antwort ist, per sowohl bei Verwendung von `musterfolge` als auch `mustermenge` jeweils auf Teilwortbasis erfolgt, während die Synonymerkennung (welche Statistikeinträge anschließend zu einem einzigen zusammenzufassen statt getrennt zu zählen sind) in einem späteren Schritt der Statistikerzeugung auf den zusammengesetzten Aufzählungen basiert. Im letzten Beispiel sind daher für die Synonyme nach wie vor alle (nach alphabetischer Sortierung) möglichen Kombinationen der zwei aus drei Antworten einzeln aufzuführen.

Die `musterfolge`- bzw. `mustermenge`-Elemente bieten in den letzten Beispielen unter anderem den Vorteil, dass die Toleranz-Einstellung `uebereinstimmung="75"` nicht für die zusammengesetzten Begriffe zählt, sondern für jeden Teilbegriff aus den Musterfolge-Eingaben einzeln. Das wurde in den letzten Beispielen bereits demonstriert, jedoch galt dort bislang stets für jedes der Teilworte *dieselbe* Übereinstimmungs-Einstellung.

Darüber hinaus haben Sie die Möglichkeit, bei Verwendung von `musterfolge`- oder `mustermenge`-Elementen die folgenden drei Einstellungen nicht nur global (als Attribute im `statistik`-Element) anzugeben, sondern *separat für die einzelnen Folgenglieder*:

- `nachkommastellen`
- `input-type`
- `zahl-aus-text`
- `uebereinstimmung`

Dazu können Sie *vor* den Musterangaben maximal ein `formatfolge`-Element einfügen, das wiederum ein Kindelement `format` pro erwartetem Teilwort der Eingaben enthält. Die `format`-Elemente wiederum haben selbst keine Kindelemente, aber können optional die oben genannten drei Attribute enthalten.

Wenn eine solche Formatfolge angegeben wurde, werden für die jeweiligen Teilworte nach dem Auftrennen der Eingaben per `trenner` (und ggf. `escape`) diese Einstellungen einzeln angewandt.

Beispiel: Angenommen, in einer Aufgabe wurde nach Buchungssätzen gefragt, die aus jeweils drei Zahleneingaben (Sollkontonummer, Habenkontonummer, Buchungsbetrag) bestehen. Die Teileingaben sollen einzeln mit Musterintervallen verglichen werden, damit die Zählung nicht von der konkreten Schreibweise der Zahlen, sondern von deren Wert abhängt (s.o.). Die Geldbetrags-Eingaben sollen dabei in der Statistik (egal wieviele Nachkommastellen tatsächlich von den Studenten eingegeben wurden) auf 2 Nachkommastellen gerundet werden, die Kontonummern dagegen auf ganze Zahlen. D.h. die Einstellung `nachkommastellen` soll nicht für die gesamte Antwortstatistik einheitlich, sondern für die Teilworte der Buchungssätze unterschiedlich definiert werden. Dies geschieht mit Hilfe einer Formatfolge wie folgt:

```
<statistik maxlaenge="31" trenner="/" escape="\" aufzaehlen=" / ">
    <titel>Test</titel>
    <feld>A10</feld>
    <feld>A11</feld>
    <trennschaerfe/>
    <formatfolge>
        <format nachkommastellen="0"/>
        <format nachkommastellen="0"/>
        <format nachkommastellen="2"/>
    </formatfolge>
    <musterfolge>
        <musterintervall von="110" bis="110"/>
        <musterintervall von="300" bis="300"/>
        <musterintervall von="42.42" bis="42.42"/>
    </musterfolge>
    <musterfolge>
        <musterintervall von="112" bis="112"/>
        <musterintervall von="300" bis="300"/>
        <musterintervall von="0" bis="10" linksoffen="true"
rechtsoffen="true"/>
    </musterfolge>
</statistik>
```

Nehmen wir als Variante noch an, dass Sie im HTML keine beliebigen Textfelder (`<input type="text">`) für die Eingabe der Buchungssätze verwendet haben, sondern jeweils Inputs der Art `<input type="number" inputmode="numeric" min="0" max="999"…>` (Eingabe

einer natürlichen dreistelligen Zahl) für die Kontonummer-Felder sowie `<input type="number" inputmode="decimal" step="0.01">` (Eingabe Fließkommazahl mit maximal zwei Nachkommastellen) für die Betragsfelder.

Dann sollten Sie die Formatfolge noch jeweils um die Inputtype-Number-Angabe ergänzen:

```
…
<formatfolge>
    <format nachkommastellen="0" input-type="number"/>
    <format nachkommastellen="0" input-type="number"/>
    <format nachkommastellen="2" input-type="number"/>
</formatfolge>
…
```

Die Nachkommastellen-Attribute sollten Sie dennoch beibehalten. Die Funktion der Rundung im Falle der Eingabe von Zahlen mit mehr Nachkommastellen kommt dann zwar im Regelfall nicht mehr zum Einsatz, da Webbrowser (zumindest die neueren Versionen) normalerweise keine längeren Eingaben zulassen. Das Attribut hat aber auch Auswirkungen auf die „normalisierte“ Darstellung der gezählten Antworten in der Statistik.

Weiterführende Themen

Im vorangegangenen Kapitel wurde die Erstellung von Aufgaben sowohl mit Assistenten als auch mit der sog. „fortgeschrittenen Aufgabenerstellung“ vorgestellt. Jedoch beschränkte sich das Kapitel auf den *Normalfall* von statischen Aufgabenseiten: Der Student öffnet das Aufgabenformular zu einer Aufgabe, sendet darüber seine Lösungen ein und erhält eine Einsendebestätigung („Quittung“). Das ist ein relativ starres, einstufiges Bearbeitungsmuster. Auch wenn mehrere Teilaufgaben <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#ta> zu bearbeiten sind, wiederholt sich dasselbe Muster für jede Teilaufgabe: Der Student ruft erneut das Aufgabenformular auf, füllt die Felder der Teilaufgabe aus, sendet die Eingaben ein und erhält die Quittung. Etwas mehr Dynamik erhält der Prozess bei Einsatz eines Vorkorrekturmoduls, da die Quittungsseite dann nicht nur eine einfache Eingangsbestätigung enthält, sondern bereits Feedback liefert, woraufhin der Student eine weitere Wiederholung dieser Sequenz (Aufgabenformular, Einsendung, Quittung) vornehmen kann.

In der fortgeschrittenen Aufgabenerstellung kann der Aufgabenautor aber bei Bedarf auch von diesem starren Schema abweichen. In diesem Kapitel sollen verschiedene Anregungen dazu gegeben werden (ohne Anspruch auf vollständige Darstellung aller Möglichkeiten).

Multi-Step-Submit: Aufgabeneinsendung in mehreren Schritten

Im Grundlagen-Kapitel wurde bereits darauf eingegangen, dass eine Aufgabe in mehrere Teilaufgaben <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#ta> zerlegt werden kann, deren Formulareingaben getrennt voneinander einzusenden sind.

Im Normalfall finden sich die Eingabemöglichkeiten für alle Teilaufgaben gleichzeitig sichtbar in ein- und derselben Aufgabenseite – mit den bereits beschriebenen Nachteilen. Es wäre für gewisse Aufgabenstellungen möglicherweise wünschenswert, den Studenten immer nur das Formular einer Teilaufgabe zur selben Zeit anzuzeigen.

Das wäre unter anderem in der Aufgaben-HTML-Seite mittels JavaScript realisierbar, z.B. durch Einbindung von Tabs, mit denen sich scriptgesteuert zwischen den einzelnen Teilaufgaben-Formularen wechseln lässt – wobei das Script auch eine Warnung aussprechen sollte, wenn versucht wird, nach Vornahme von Eingaben aber ohne Submit zu einer anderen Teilaufgabe zu wechseln. (Alternativ wäre zu überlegen, dieses Tabbed Form nicht in Teilaufgaben zu zerlegen, sondern alle Eingaben in allen Tabs in einem einzigen Submit einzusenden, also zu einer einzigen Teilaufgabe zusammenzufassen.)

Ein anderer, explizit vom Online-Übungssystem unterstützter Weg zur getrennten Teilaufgabenbearbeitung (der jedoch kein beliebiges Wechseln zwischen den Teilaufgaben erlaubt, sondern nur die Vorwärtsnavigation in einer vorgegebenen Sequenz, optional auch mit Verzweigungen) basiert auf Teilaufgaben-Formularen innerhalb von Quittungsseiten. Dieser Ansatz ist im Folgenden gemeint, wenn von „Multi-Step-Submit“ die Rede ist.

Beispiel: einfache Multi-Step-Submit-Sequenz

Nehmen wir beispielhaft an, die Aufgabe 2 im ersten Aufgabenheft soll in drei Schritten bearbeitet werden: Der Student soll zuerst nur das Formular der Teilaufgabe `A` sehen und ausfüllen. Sobald er die Eingaben eingesendet hat, sollen weitere Fragen gestellt werden, die die Teilaufgabe `B` bilden. Nach deren Einsendung soll wiederum ein drittes Formular (Teilaufgabe `C`) angezeigt werden. Abschließend, nach Einsendung auch dieses dritten Formulars, sollen alle Eingaben zu den drei Teilaufgaben `A`, `B` und `C` quittiert werden.

Dies ist im Übungssystem (nur in der fortgeschrittenen Aufgabenerstellung) wie folgt realisierbar:

- In der eigentlichen Aufgabendatei `aufgabe2.1.html` wird (nur) das Formular für Teilaufgabe `A` realisiert. Der Submit-Button `einsendenA` sollte hier geeignet beschriftet werden, um anzeigen, dass nach Einsendung weitere Fragen folgen werden, z.B. mit »Weiter…«, »Speichern und fortfahren…« oder ähnlich.
- In der Quittungsvorlage `quittung2.1.a.html` werden einerseits (wie üblich) die Eingaben der Teilaufgabe `A` zur Kontrolle aufgelistet („quittiert"). Andererseits enthält diese HTML-Datei aber auch das Aufgabenformular für Teilaufgabe `B`. Das Form-Element wird genauso aufgebaut, wie das in der Aufgabenseite. Hierzu gelten also dieselben Regeln wie in Abschnitt zur HTML-Formular-Erstellung beschrieben.
- Die zugehörige `quittung2.1.b.html` enthält analog das dritte Eingabeformular für die Teilaufgabe `C` sowie ggf. eine Eingangsbestätigung der Eingaben zu Teilaufgabe `B`.
    - Die normale Regelung für einstufige Aufgabenbearbeitung, nach welcher eine Quittungsseite in ihrer Funktion als Eingangsbestätigung nur die Eingaben anzeigen kann, die auch wirklich im letzten Submit eingesendet wurden, gilt für Multi-Step-Bearbeitung *nicht*: Diese Quittung kann nicht nur die Einsendungen zu Teilaufgabe `B` anzeigen, sondern darf auch alle bereits eingegangenen Eingaben von `A` *und* `B` zusammenstellen.
    - Dazu sind neben den Variablen der Art `$Feld1`, `$Feld2` etc., welche sich auf die Eingaben zur gerade eingesendeten Teilaufgabe (hier: `B`) beziehen, bei Multi-Step-Submit ausnahmsweise auch Feld-Variablen mit expliziter Teilaufgabenkennung einsetzbar, wie z.B. `$FeldA1` für die Eingaben im ersten Feld der Teilaufgabe `A`.
    - Das Online-Übungssystem erkennt Quittungen, die zu Multi-Step-Submit-Aufgaben gehören, automatisch und sorgt dann dafür, dass derartige Feld-Variablen für andere Teilaufgaben in diesem Sonderfall definiert sind.
- Die Eingaben zur dritten und letzten Teilaufgabe `C` werden von `quittung2.1.c.html` bestätigt.
    - Dies ist nun die *eigentliche* Einsendebestätigung, die die mehrstufige Aufgabenbearbeitung abschließt.
    - Dementsprechend sollte sie alle Eingaben zu allen drei Teilaufgaben `A`, `B`, und `C` über entsprechende `$Feld<TA><Nr>`-Variablen zusammenstellen.
    - Davon abgesehen ist dies eine „gewöhnliche" Quittungsseite ohne weiteres `form`-Tag.
- Die Funktion »Teilaufgaben und Felder erzeugen« erkennt Multi-Step-Formulare und berücksichtigt bei ihrer Analyse nicht nur das Formular in der Aufgabenseite selbst, sondern auch die Formulare in allen Quittungsseiten zu den Teilaufgaben, zu denen die Aufgabenseite Submit-Buttons enthält (sowie rekursiv auch alle weiteren Quittungsseiten zu denen ein so gefundenes Formular in einer anderen Quittungsseite Submit-Buttons enthält).

Jeder Student wird in diesem Beispiel also normalerweise dieselben drei Schritte der Aufgabenbearbeitung absolvieren. Unterbricht ein Student die Bearbeitung der Aufgabe z.B. nach

der Einsendung zu Teilaufgabe `B`, so sind die Eingaben zu den ersten beiden Teilaufgaben dennoch gespeichert und können gewertet werden. Ruft der Student später die Aufgabe erneut auf, sieht er zunächst wieder nur das Formular zu Teilaufgabe `A`, welches seine gespeicherten Eingaben vom letzten Mal enthält. Er kann diese Eingaben ändern oder unverändert neu einsenden, um wieder zur Teilaufgabe `B` weiterzuwechseln. Auch hier sieht er seine letzen Eingaben und ein erneuter Submit führt weiter zum – in diesem Beispielfall noch leeren – Formular der Teilaufgabe `C`, in welchem nun die fehlenden Eingaben noch nachgeholt werden können.

Dieses Beispiel ist immer noch relativ „starr" insofern, als dass immer noch jeder Student immer dieselben Teilaufgaben bearbeitet. Der wesentliche Unterschied zur einzigen Aufgabenseite ist, dass sich die Fragen auf mehrere HTML-Seiten verteilen, die immer in Sequenz abgearbeitet werden.

Der Multi-Step-Ansatz bietet aber auch verschiedene Möglichkeiten zur „Dynamisierung".

Verzweigungen in Multi-Step-Sequenzen

Auf immer noch statischem Wege lassen sich verschiedene Verzweigungen im Fluss durch die Teilaufgaben erreichen, indem in einem Formular mehrere Teilaufgaben mit separaten Submit-Buttons angelegt werden. Z.B. könnte es in der `aufgabe2.1.html` zwei Teilaufgaben `A` und `B` geben, und je nachdem, welchen der beiden Buttons `einsendenA` oder `einsendenB` der Student nun anklickt, können ihm im Anschluss unterschiedliche weitere Fragen gestellt werden, indem z.B. `quittung2.1.a.html` ein Formular für Teilaufgabe `C`, `quittung2.1.b.html` dagegen ein Formular für Teilaufgabe `D` enthält.

Allerdings stellt diese Konstruktion nicht sicher, dass ein Student wirklich *entweder* die Sequenz (`A`, `C`) *oder* die Sequenz (`B`, `D`) bearbeitet, da er durch erneuten Aufruf der Aufgabe auch die jeweils zweite Sequenz im Anschluss zusätzlich ausführen kann.

Dynamisierung über Vorkorrekturmodule

Dadurch, dass Vorkorrekturmodule <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#moduleoverview> teilaufgabenspezifisch direkt nach jeder Einsendung ausgeführt werden und das Aussehen einer Quittungsseite beeinflussen können, ist mit ihnen insbesondere auch eine dynamische Multi-Step-Aufgabenbearbeitung möglich:

Nehmen wir z.B. an, die `aufgabe2.1.html` enthalte wieder genau ein Formular zu einer Teilaufgabe `A` mit entsprechendem Submit-Button `einsendenA`. Zu dieser Teilaufgabe `A` sei nun ein spezielles Vorkorrekturmodul registriert, dessen Ausgabe in `quittung2.1.a.html` über die Variable `$Vorkorrektur` eingebunden wird.

Dann ist es insbesondere möglich, dass das Vorkorrekturmodul ein HTML-Fragment[26] zurückgibt, welches ein komplettes, dynamisch berechnetes Formular enthält. Auf diese Weise kann das Vorkorrekturmodul spezifisch die weiteren zu stellenden Fragen berechnen. Es kann bestimmen, welche Eingabefelder das Formular enthalten soll und zu welcher Teilaufgabe deren Inhalte einzusenden sind.

Das Vorkorrekturmodul sollte nur den „dynamischen Kern" des Formulars erzeugen, während die `form`-Tags selbst sowie ggf. immer gleiche (statische) Formularelemente in der Vorlagendatei stehen. Bestimmt das Vorkorrekturmodul z.B. nur eine Auswahl von Eingabefeldern für die immer selbe Teilaufgabe, sollte auch der Submit-Button mit in der Vorlagendatei stehen. Entscheidet das Modul dagegen darüber, für welche Teilaufgabe eingesendet wird, wird es auch den entsprchenden Submit-Button selbst generieren und als Teil der Vorkorrektur ausgeben. Das folgende Beispiellisting

geht von letzterem Fall aus:

```
<form method="POST"
action="$WebAssignServer/$Veranstaltername/Einsendung/$KursNr/$VersionsNr/$Aufg
abenheftNr/$AufgabenNr/">
  …
  $Vorkorrektur
  <!--| Vorkorrektur fügt dynamisch Formularelemente und Submit-Button ein |-->
  …
</form>
```

Für vom Vorkorrekturmodul erzeugte Formularfelder gilt dasselbe wie für statische Felder in HTML-Dateien, insbesondere kann ein Vorkorrekturmodul die Felder, die es dynamisch einfügt, auch dynamisch mit gewissen (vom Studenten editierbaren) Inhalten / Vorschlägen füllen. Wie bei statisch vorgefüllten Feldern gilt auch hier, dass einem Studenten diese Vorschläge des Vorkorrekturmoduls nur dann angezeigt werden, wenn er für die zugehörige Teilaufgabe noch nie etwas eingesendet hat – andernfalls sieht er dort seine letzte Eingabe.

Bei diesem Vorgehen ist jedoch ein *wichtiger Aspekt der Aufgabeneinrichtung* zu beachten: Wie im Kapitel zur fortgeschrittenen Aufgabenerstellung ausgeführt wurde, muss das Übungssystem auf Einsendeformulare vorbereitet werden. Zu jeder Aufgabe muss das Übungssystem die genaue Anzahl ihrer Teilaufgaben und die Definition der Eingabefelder jeder Teilaufgabe kennen. Dazu erfolgt eine statische Analyse der Aufgaben-HTML-Datei (und ggf. der weiteren Formulare in von dieser direkt oder indirekt erreichbaren Quittungsseiten), die der Aufgabenautor bei der fortgeschrittenen Aufgabenerstellung manuell anstoßen muss (»Teilaufgaben und Felder erzeugen«). Diese Funktion kann natürlich (leider) keine dynamisch von Vorkorrekturmodulen generierten Formulare analysieren! Als Ersatz sind statt dessen statisch Hidden-Fields in ein HTML-Formular einzufügen, eines für jedes Eingabefeld, das in dem vom Vorkorrekturmodul erzeugten Formular vorkommen kann! Dabei müssen diese Hidden-Fields zwar in einem Form-Element stehen, dürfen jedoch *nicht im selben* Form-Element stehen wie die vom Vorkorrekturmodul erzeugten „echten“ Formularelemente[27]! In obigem Codeauszug z.B. dürfen diese Hidden Fields *nicht* im selben `form`-Element wie die `$Vorkorrektur`-Variable stehen.

Da die Hidden Fields nur der statischen Codeanalyse zur Aufgabeneinrichtung dienen und ihre Inhalte (typischerweise leere Strings) niemals eingesendet werden sollen, können für diese „Dummy-Felder“ auch einfache, nicht-einsendbare „Dummy-Formulare“ angelegt werden – mit einfachen Form-Tags ohne jegliche Attribute und natürlich ohne Submit-Button im Formular. Falls Sie Embedding verwenden, bietet es sich an, dieses Dummy-Formular auch außerhalb des Embedding-Bereichs einzufügen, so dass es (außer in der Fallback-Darstellung bei fehlgeschlagenem Embedding) auch niemals mit an die Browser der Studenten übertragen wird. (Sobald sich in einer Webseite mit Embedding mehrere Formulare befinden, ist zu beachten, dass diese jeweils vollständig innerhalb oder außerhalb des Embedding-Bereichs liegen müssen, also z.B. das öffnende Form-Tag des auszuliefernden Formulars nicht bereits vor der `$EMBED`-Marke liegen darf!)

Greifen wir nochmals das obige Codebeispiel auf und nehmen wir an, dass die Ausgabe des Vorkorrekturmoduls dynamisch (abhängig von den Eingaben des Studenten im vorherigen Schritt) entweder ein Formularfeld `FeldB1` oder die beiden Formularfelder `FeldC1` und `FeldC2` (sowie dazu passend einen Submit-Buttons entweder mit Namen `einsendenB` oder `einsendenC`) ins Formular einfügt. Dann könnte die Einbindung besagter Hidden-Fields zur Aufgabeneinrichtung in etwa wie folgt aussehen:

```
$EMBED
<form method="POST"
action="$WebAssignServer/$Veranstaltername/Einsendung/$KursNr/$VersionsNr/$Aufg
abenheftNr/$AufgabenNr/">
  …
  $Vorkorrektur
  <!--| Vorkorrektur fügt dynamisch Formularelemente und Submit-Button ein |-->
  …
</form>
$/EMBED
<form>
  <!-- Aufgabeneinrichtung für ggf. in Vorkorrektur ausgegebene Formularelemente
-->
  <input type="hidden" name="FeldB1">
  <input type="hidden" name="FeldC1">
  <input type="hidden" name="FeldC2">
</form>
```

Teilaufgabeneinsendungen per Ajax

Streng genommen ist es dem Übungssystem egal, ob eine Einsendung zu einer Teilaufgabe ein herkömmlicher Browser-Submit und die Quittungsseite eine „echte“ HTML-Seite ist oder nicht. Statt dessen könnte eine Einsendung auch z.B. als asynchroner sog. Ajax-Request erfolgen (d.h. per JavaScript über ein `XMLHttpRequest`-Objekt gesendet). Im Browser bliebe dann weiterhin die Aufgabenseite geladen, während der JavaScript-Code im Hintergrund auf die Antwort des Übungssystems wartet. Letztere ist – wie gehabt – die ausgefüllte Quittungsseite, aber diese könnte natürlich auch nur aus einem HTML-Fragment statt einer kompletten HTML-Seite bestehen oder sogar XML-Code enthalten. Das empfangende JavaScript im Aufgabenformular kann die Antwort (Quittung) dann verarbeiten und daraufhin dynamisch das Aufgabenformular modifizieren.

So lange die Quittung nur eine reine Einsendebestätigung darstellt, also die eingesendeten Daten wieder zurücksendet, mag das allein noch wenig nützlich sein, aber in Verbindung mit einem Vorkorrekturmodul <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#vkmodule> , dessen Ausgaben über die Quittungs-Antwort wieder an das JavaScript der Aufgabenseite zurückgegeben werden, sind damit sehr dynamische Aufgabenformulare realisierbar.

Dabei ist jedoch immer im Kopf zu behalten, dass jede Teilaufgaben-Einsendung in erster Linie dazu dient, Eingabe des Studenten zu speichern. Dementsprechend entsteht auch bei einer solchen Ajax-Einsendung ein gewisser Overhead: Die eingesendeten Daten werden zuerst in der Datenbank als Einsendung des Studenten zur Aufgabe gespeichert, bevor das Vorkorrekturmodul ausgeführt wird. Auch dessen Rückgabe wird wiederum in der Datenbank gespeichert, bevor sie in die Quittungsseite eingefügt und so an den aufrufenden Browser gesendet wird.

HTTP-Proxy mit Autorisierungsprüfung

Greifen wir noch einmal das Szenario aus dem vorangehenden Abschnitt auf, nehmen also an, Sie wollen ein dynamisches Aufgabenformular entwickeln, das sich mit Hilfe von Ajax-Requests abhängig von Formulareingaben verändert. Nun seien jedoch die vom externen Server dynamisch zu verarbeitenden Daten rein transienter Natur, so dass weder für die Speicherung der Eingaben noch der berechneten „Vorkorrektur" ein Bedarf besteht, die oben beschriebene Ajax-Einsendung mit Vorkorrektur also nicht optimal ist.

Nun können Sie natürlich beispielsweise selbst einen kleinen Webservice oder ähnliches entwickeln und Ihren Server *direkt* ansprechen, statt die Daten über das Übungssystem durchzuschleifen. Das wiederum kann aber auch einige Nachteile haben. So müsste Ihr Server insbesondere öffentlich im Internet erreichbar sein. Setzen Sie dann für diese dynamische Verarbeitung eventuell noch lizenzpflichtige Software ein, die Sie nur FernUni-Studenten zugänglich machen, aber nicht Netz-öffentlich hosten dürfen, dann kommt dieser Ansatz nicht mehr in Frage.

Für derartige Anwendungszwecke bietet das Online-Übungssystem einen speziellen HTTP-Proxy mit zwei wesentlichen Kernfeatures:

- Er erlaubt es, Ihren Server, den Sie z.B. per Ajax vom Browser des Studenten ansprechen wollen, nur lokal im Netzwerk der FernUni erreichbar zu halten, denn die Requests gehen nicht direkt vom Browser (übers Internet) direkt an Ihren Server, sondern zunächst ans Übungssystem. Anders als Einsendungen wird das Übungssystem diese Daten jedoch nicht verarbeiten, speichern oder konvertieren, sondern der Request wird vom Proxy nahezu unverändert an Ihren Server weitergeleitet. Auch die Antwort geht dann von Ihrem Server zunächst ans Übungssystem und wird von dort zurück an den Browser übertragen.
- Weiterhin bietet der Proxy eine transparente Autorisierungsprüfung: Nur Studenten, die überhaupt zur Bearbeitung der Aufgabe autorisiert sind – typischerweise durch Belegung des zugehörigen Kurses –, können auch auf Ihren Server zugreifen: Der Browser muss die Ajax-Requests mit denselben Logindaten versehen, mit denen er auch die Aufgabenseite selbst abgerufen hat. Ohne Autorisierung wird der Proxy den Request nicht an Ihren Server weiterleiten.

Dieser Proxy wird in einem gesonderten Handbuch <https://online-uebungssystem.fernuni-hagen.de/download/AuthProxy/UebungssystemAuthProxy.html> genauer beschrieben.

Fußnoten

1. Das Einsenden ist natürlich nur möglich, so lange der Einsendeschluss noch nicht verstrichen ist. Nach Einsendeschluss hat der Student Zugriff aufs Aufgabenformular und kann weiterhin (z.B. zum Abgleich mit einer bereits vorliegenden Korrektur) die Aufgabenstellung einsehen sowie seine Eingaben sehen, aber keine Änderungen mehr einsenden.
2. Unterschied: Eine Musterlösungsfreigabe »nur mit Korrektur« erzwingt, dass nur Studenten, die auch etwas eingesendet haben, Zugriff auf die Musterlösung erhalten, sobald auch ihre Korrektur freigegeben wird. »Nicht vor Korrektur« erlaubt auch Studenten ohne aktive Beteiligung den Musterlösungszugriff, sobald alle Korrekturen (der aktiven Teilnehmer) zur Aufgabe freigegeben wurden.
3. Auch bei automatischer Korrektur kann dieser Zustand eintreten, nämlich für die Dauer zwischen Einsendeschluss und Heft-Schließen. Ein automatisches Heft-Schließen z.B. findet normalerweise in der Nacht nach dem Einsendeschluss statt. Kursbetreuer können das Heft-Schließen auch manuell sehr kurz nach Einsendeschluss auslösen, um die Wartezeit auf die

Korrekturen zu minimieren.

4. Zu beachten: Im Online-Übungssystem müssen Aufgabenhefte und Aufgaben innerhalb eines Hefts lückenlos nummeriert sein. Falls Sie nun zu einem Kurs, der noch gar keine Aufgaben und Aufgabenhefte enthält, z.B. eine Datei namens `aufgabe3.2.html` hochladen, also eine Aufgabenstellung zu Aufgabe 2 aus Heft 3, so werden im Online-Übungssystem drei Aufgabenhefte angelegt und im dritten werden zwei Aufgaben angelegt, wobei die hochgeladene Aufgabenstellung der zweiten Aufgabe zugeordnet wird und die erste Aufgabe noch „leer" ist.
5. Der Text „[leer]" lässt sich über das `data-if-empty`-Attribut durch einen beliebigen anderen Text, z.B. „keine Dateieinsendung vorhanden" austauschen, siehe Das HTML-Formular/Eingabefelder
6. Alternativ könnte man statt `$IfExistsA2` auch `$IfNotEmptyA2` verwenden, um diesen Text nur dann anzuzeigen, wenn wirklich eine nicht-leere Einsendung, in diesem Fall also eine Dateieinsendung vorliegt. Das wird aber *nicht* empfohlen: Wenn bei leerer Einsendung überhaupt keine Informationen zur vorherigen Einsendung angezeigt werden, kann ein Student nicht mehr sofort erkennen, dass von ihm keine Dateieinsendung vorliegt und er in diesem Feld am besten noch eine Einsendung vornehmen sollte! Insbesondere, falls er schon einmal eine Datei hochgeladen hatte und diese nur versehentlich durch eine leere Einsendung überschrieb, ist es sinnvoll, ihm dies explizit anzuzeigen. (Ohne Anzeige könnte er immer noch der Meinung sein, dass seine Dateieinsendung vorliegt und das lediglich (für beliebige Einsendungen) im Aufgabenformular nicht erkennbar ist.)
7. Die Studenten-Heftstartseite ist streng genommen kein Thema der Aufgabenerstellung an sich und wird daher in diesem Handbuch nicht ausführlich behandelt, aber teilweise wird Bezug darauf genommen. Normalerweise wird diese Darstellung nicht verwendet: Im normalen Studentenzugang wird den Teilnehmern die sog. Studenten-Startseite angezeigt, die eine tabellarische Übersicht über die *gesamte* Kursumgebung, also sämtliche Aufgabenhefte und die darin enthaltenen Aufgaben zeigt. Von dieser Gesamt-Aufgabenübersicht aus navigieren die Teilnehmer direkt über Links zu den einzelnen Aufgaben, es ist nicht nötig, erst ein Aufgabenheft zu öffnen und dann darin eine Aufgabe auszuwählen.
   Die Aufgabenheft-Startseite ist eine (im normalen Studentenzugang nirgends verlinkte) Alternativdarstellung mit der Aufgabenübersicht für *genau ein* Aufgabenheft. Sie wird typischerweise verwendet, wenn ein ganzes Aufgabenheft per LTI <https://online-uebungssystem.fernuni-hagen.de/download/LTI_Moodle/Uebungssystem_LTI_Moodle.html> in eine externe Lernumgebung wie Moodle eingebunden wird.
   Bei Bedarf können Sie auch selbst HTTP-Links zu Heft-Startseiten verwenden, der URL-Aufbau entspricht dem der normalen Studentenzugang-Startseite, nur dass der Servicename `StudentenStartSeite` durch `StudentenHeftStartSeite` ersetzt und hinter der Semesterangabe, per Slash/Querstrich getrennt, die Aufgabenheftnummer angehängt wird. Beispiel: `https://online-uebungssystem.fernuni-hagen.de/xyz/StudentenHeftStartSeite/12345/WS20/2/`, wobei die abschließende `2` die Übersichtsseite zum zweiten Aufgabenheft öffnet (und natürlich auch die Dummy-Platzhalter `xyz` durch den korrekten Veranstalter, `12345` durch die korrekte Kursnummer und `WS20` durch die gewünschte Semesterkennung zu ersetzen sind).
   Es gibt eine Standardansicht der Studenten-Heftstartseite, die sich aber auch durch eine eigens angepasste Seitenvorlage ersetzen lässt. Mehr dazu finden Sie im Designvorlagenpaket <https://online-uebungssystem.fernuni-hagen.de/download/Designvorlagen2018/Designvorlagen2018OnlineUebungssystem.html#studentenheftstartseite> .
8. Die Option ist nur sichtbar und aktivierter, wenn einige Grundvoraussetzungen für Selbstkontrollarbeiten gegeben sind. Insbesondere darf das Heft keine manuell korrigierten Aufgaben besitzen: Es wäre alles andere als sinnvoll, menschliche Korrektoren mit der

Korrekturarbeit zu beauftragen, wenn Studierende dann (vor, während oder gar nach Abschluss der Korrekturarbeiten) einfach das Heft wieder öffnen und damit diese Korrekturen wieder löschen könnten, nur um das Heft später wieder zu schließen, und der Korrektor müsste die Einsendung (ob nun verändert oder nicht) erneut korrigieren. Außerdem ist die Option nur verfügbar, wenn in den Kursparametern der Heft-Schließen-Knopf für Studierende aktiviert ist, denn der Heft-Öffnen-Knopf ist eine Funktion*sergänzung* zum Heft-Schließen-Knopf und hat nur eine Funktion, wenn Hefte schon manuell vor Einsendeschluss geschlossen wurden.

9. Prinzipiell werden auch mehrere `form`-Elemente unterstützt, so lange die Formularelemente jeder Teilaufgabe <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#ta> (separat einzusenden) alle im selben `form`-Element liegen. Falls ein Formular nur eine Teilmenge der Formularfelder einer Teilaufgabe einsendet, verhalten sich die beiden Teile der Teilaufgabe exklusiv: Angenommen eine Teilaufgabe A bestehe aus drei Eingabefeldern `FeldA1`, `FeldA2` und `FeldA3`, wovon `FeldA1` und `FeldA2` im selben Formular liegen, `FeldA3` jedoch in einem anderen Formular. Dann kann jeder Student nur *entweder* in den Feldern `FeldA1` und `FeldA2` etwas einsenden *oder* in `FeldA3`, nie in allen dreien! (Jede Einsendung zu `FeldA3` im zweiten Formular würde ggf. zuvor über das erste Formular eingesendete Eingaben zu `FeldA1` und `FeldA2` wieder löschen – und umgekehrt.) Das kann im Ausnahmefall natürlich gewollt sein, z.B. falls ein Student *entweder* direkt einen Text in ein Textfeld eingeben darf *oder* eine PDF-Datei einsenden darf. **Wichtig bei Verwendung mehr als eines `form`-Elements in Kombination mit Embedding (`$EMBED`):** Achten Sie in diesem Fall darauf, dass alle Formular-Tags komplett zwischen `$EMBED` und `$/EMBED` stehen! (Nur bei genau einem Formular ist das Übungssystem in der Lage, auch die Form-Tags außerhalb des Embed-Berechs zu berücksichtigen.)
10. Streng genommen ist der eingesendete *Wert*, wenn keine Checkbox angekreuzt ist, der *leere String*! Wird ein (Vor-)korrekturmodul eingesetzt, bekommt dieses entsprechend auch den leeren String als Eingabe zur Korrektur/Bewertung übermittelt. `---` ist lediglich der dem Studenten in der Quittung und der Korrekturseite angezeigte Wert, der im Gegensatz zum leeren String sichtbar ist und visualisieren soll, dass eine leere Eingabe erfolgt ist, hier also keine Checkbox angekreuzt wurde.
11. Wir empfehlen, diese hier wirkungslose Angabe von `data-ignore="empty"` dennoch vorzunehmen, denn sie wird im Zweifel wieder wirksam, falls bei einer späteren Überarbeitung der Aufgabe der `never`-Anteils der Teilaufgabe wegfallen sollte (z.B. durch Ändern der `ignore`-Einstellung für die Checkboxen, Zuordnen der Checkboxen zu einer anderen technischen Teilaufgabe oder komplettes Entfernen der Checkboxen aus der Aufgabe).
12. Die `$Randomize(...)`-Variable muss in jedem Fall vor der ersten `$Random...`-Variablen stehen, oder anders gesagt: Wird bei der Verarbeitung der Aufgabenseite eine `$Random`-Variable gefunden, ohne dass zuvor eine `$Randomize`-Variable auftrat, dann gelten die Grundeinstellungen für Randomisierung und jede folgende Randomize-Variable wird ignoriert. Dazu muss die Randomize-Variable nicht unbedingt *unmittelbar* vor der ersten Random-Variablen stehen (auch wenn das ein typischer und passender Ort ist), es kann auch noch Content zwischen beiden Variablen geben. Falls Sie Embedding nutzen, muss die Randomize-Variable irgendwo zwischen `$EMBED...` und der ersten `$Random...`-Variablen platziert werden.
13. Angenommen, Sie erstellen eine Aufgabe mit vier Fragen (in Form von vier Random-Blöcken) und erzeugen dann (per Aufgabenimport-Funktion) zwei Kopien dieser Aufgabe im selben Heft, so dass dieselbe Aufgabe also dreimal im Heft vorkommt. Wenn diese Aufgabenseite nun die Variable `$RandomizeHeft` enthält, so dass in jeder der Aufgaben eben genau eine der vier Fragen gestellt wird, dann bewirkt der Zusatz „Heft“ am Variablennamen, dass eben die Zufallsauswahl nicht in jeder der drei Aufgaben unabhängig voneinander stattfindet (also auch

zwei Aufgaben zufällig dieselbe Frage zeigen könnten), sondern dass in jeder der drei Aufgaben eine andere der vier Fragen gewählt wird. So lange die Fragenzahl größer als die Aufgabenzahl ist, funktioniert das hier.
Bei Angabe von `$RandomizeHeft(2)` dagegen ginge das Beispiel nicht mehr auf: Es würde versucht, in jeder der drei Aufgaben zwei andere aus den vier Fragen auszuwählen als in den anderen Aufgaben. Das liefe aber auf eine Auswahl von insgesamt sechs verschiedenen Fragen aus vier vorhandenen hinaus, ist also nicht möglich. Es fände also sehr wohl – trotz des `Heft`-Zusatzes an der Randomize-Variable – darauf hinaus, dass identische Fragen in mehr als einer der Aufgaben angezeigt würden. Eine Erhöhung der Fragenzahl auf sechs oder Reduzierung der Aufgabenzahl auf zwei würde den Konflikt beheben, dann würden wieder in jeder der Aufgaben garantiert unterschiedliche Fragen ausgewählt.

14. Das Dollarsymbol wurde gewählt, weil es weitgehend dem Buchstaben S wie Selektion ähnelt, aber selbst eben kein regulärer Buchstabe aus {A, …, Z} ist. Der Buchstabe S selbst ist bereits als reguläre Teilaufgabenkennung (für die neunzehnte Teilaufgabe) reserviert.
15. Tatsächlich *muss* der Teilaufgaben-Buchstabe nicht unbedingt entfallen, seine Angabe schadet nicht – ist aber unnötige Redundanz und deshalb nicht empfohlen, denn die Teilaufgabe steht ja schon fest. Es ist jedoch im Normalfall *nicht* möglich, Feld-Variablen von anderen Teilaufgaben in eine Quittung einzubunden, sondern eine Quittung darf nur die eben eingesendeten Daten quittieren und keine, die schon vor der Einsendung gespeichert waren! In `quittung1.1.a.html` z.B. ist `$FeldA1` gleichbedeutend mit `$Feld1`, während Variablen wie `$FeldB1` schlicht entfernt und nicht durch konkrete Werte ersetzt werden. Eine *Ausnahme* von dieser Regel gibt es jedoch: Im Spezialfall Multi-Step-Submit können tatsächlich auch in Quittungen über Feldvariablen der Art `$FeldB1` Einsendungen zu anderen Teilaufgaben mit eingebunden werden, um die in mehreren Schritten über mehrere Teilaufgaben gesammelten Einsendungen in derselben Quittungsseite aufführen zu können.
16. Erwarten Sie z.B. von Studenten die Einsendung bestimmter Quelltexte als Textdatei und möchten die Inhalte aber immer in Quittung und Korrektur direkt eingebettet haben, selbst wenn z.B. aufgrund einer dem Browser unbekannten Extension der Content-Type `application/octet-stream` o.ä. gesendet wird, können Sie dies nutzen (dann vorzugsweise in der `P`-Variante in einem `pre`-Element). Allerdings hat das ein paar Haken: Erstens funktioniert bei Dateieinsendungen, die keinen `text/*`-ContentType haben, die automatische Charset-Erkennung nicht, so dass Sonderzeichen wie Umlaute beim Einbetten fehlerhaft dargestellt werden könnten. Zweitens könnten die Studenten auch Binärdateien einsenden, und das sähe dann ziemlich „hässlich“ aus.
17. Ist eine automatische Vorkorrektur ohne Sofort-Feedback gewünscht, kann statt eines Vorkorrekturmoduls ein normales, nach Einsendeschluss ausgeführtes Korrektur-/Bewertermodul gesetzt werden – auch da ist ggf. manuelle Nachkorrektur noch möglich, siehe Korrekturmodus »Automatisch und von Hand« <https://online-uebungssystem.fernuni-hagen.de/download/Aufgabenerstellung/Aufgabenerstellung.html#autohand> .
18. Die Erkennung, ob die Vorkorrektur Plaintext oder HTML-Markup ist, erfolgt nicht durch Analyse des Contents selbst, sondern basiert auf einer Metainformation des Vorkorrekturmoduls selbst: dem zurückgelieferten Content-Type, der entweder `text/html` oder `text/plain` lauten muss.
19. Es wird jedoch vorausgesetzt, dass ein Teststudent mit der Matrikelnr 7777777 zum Kurs angemeldet ist.
20. Dies dient in erster Linie dem nachträglichen Aus-der-Wertung-nehmen von Antwortalternativen, wenn sich die Frage bzw. Antwortalternative z.B. als missverständlich erwiesen hat und man daher niemandem Punkte für diese Alternative abziehen möchte, unabhängig von der Antwort.
21. Ein bewusster Verzicht auf eine Antwort bietet einem Teilnehmer dann also keinerlei

(potentiellen) Vorteil gegenüber Raten mehr, Raten ist immer die erfolgsversprechendere Alternative. Einzige verbleibende Vorteile könnten sein, dass einerseits ein Teilnehmer selbst eine bessere Übersicht hat, welche Alternativen er schon beantwortet hat und welche noch nicht (da er „keine Antwort" von „Antwort ‚falsch'" unterscheiden kann), und dass andererseits für den Aufgabenautor die Wahl zwischen X-aus-N-Modus und 1-bis-X-aus-N-Modus wegfällt (s.o.), denn eine X-aus-N-3A-Frage gilt immer genau dann als unbearbeitet, wenn keine einzige Alternative als richtig oder falsch markiert wurde.

22. Ausnahmen sind natürlich, dass bei Wahl eines Bewerters `BegriffeIC` oder `BegriffeSmartIC` Abweichungen in der Groß-Kleinschreibung nicht gezählt werden, und dass bei `BegriffeSmart` und `BegriffeSmartIC` zu numerischen Musterbegriffen keine wörtliche, sondern eine numerische Übereinstimmung geprüft wird (siehe Smart Mode), wobei die Übereinstimmung-Einstellung keine Rolle spielt.
23. Vor Juni 2020 galt: Alle Eingaben, die länger als `maxlaenge` waren, wurden in der Statistik gar nicht mit berücksichtigt, also nicht mitgezählt. Ab Juni 2020 gilt hingegen: Die Anzahl aller Eingaben, die länger als `maxlaenge` sind, werden nun mitgezählt, aber nicht mehr einzeln nach Eingabe aufgeschlüsselt, sondern zu einem einzigen Statistikeintrag »Länger als xx Zeichen« zusammengefasst.
24. Bei Fragen, zu denen es alternative Lösungen gibt, hängt es von der Art der Statistik ab, ob es sinnvoll bzw. möglich ist, diese Musterlösungen hier zu hinterlegen. Nehmen wir wieder die eingangs gezeigte Multiple-Choice-Frage (X aus 5) als Beispiel: In obiger Beispielabbildung ist genau eine Musterlösung definiert: Die Antworten A, C und E sind korrekt. Nehmen wir nun an, alternativ sei auch die Musterlösung `A,C,D` korrekt. Für die zweite Statistik der kombinierten Antworten könnten dann problemlos beide Musterlösungen `A,C,D` als auch `A,C,E` angegeben werden, beide Antworten würden dann in der Statistik als (vollständig) korrekt markiert. Bei der Statistik der Einzelantworten dagegen könnte man zwar nun sowohl `A`, `C`, `D` als auch `E` als korrekte Antworten hinterlegen, aber dann würden laut der Statistik-Ausgabe alle diese vier Antworten als korrekte und lediglich `B` als falsche Antworten markiert, der Zusammenhang aber, dass nur bestimmte Kombinationen zusammen als korrekt gelten (also nicht die Nennung von D und E zusammen) ginge dabei verloren. Es ist in diesem Fall vielleicht ein wenig Geschmacksache, ob man diese Anzeige dann überhaupt noch wünscht oder bei der Statistik lieber auf die Markierung der korrekten und falschen Antworten ganz verzichtet. Der Aufgabenerstellungsassistent jedenfalls wird für X-aus-N-Aufgaben nur dann eine Musterlösung zur Einzelantwort-Statistik hinterlegen, wenn diese eindeutig ist.
25. Erläuterung des beispielhaften regulären Ausdrucks `Fern[Uu]ni(versit(ä|ae)t)? (in )?Hagen`: Zeichen in eckigen Klammern stehen für eine Aufzählung alternativer Zeichen (Character Classes), von denen genau eines an der entsprechenden Stelle stehen muss. Das `[Uu]` in obigem Ausdruck bedeutet also: Hier muss entweder der Großbuchstabe U oder der Kleinbuchstabe u stehen. Eine alternative Schreibweise für mehrere Alternativen ist die Aufzählung mit Pipe-Symbol, i.d.R. eingefasst in runde Klammern, wobei die Alternativen dann nicht nur einzelne Zeichen sein müssen, sondern eben auch aus Teilausdrücken / mehreren Zeichen bestehen können. So steht hier `(ä|ae)` für „entweder Buchstabe ‚ä' oder Zeichenkette ‚ae'". Ein Fragezeichen besagt, dass das vorangehende Zeichen bzw. der vorangehende in Klammern stehende Subausdruck optional ist, also auch Fehlern darf. In diesem Beispiel darf also der Name „FernUniversität" optional zu „FernUni" verkürzt werden, und das Wort „in" (gefolgt von einem Leerzeichen) vor „Hagen" darf ebenfalls entfallen.
26. Zu Beachten ist bei Erstellung des Vorkorrekturmoduls in diesem Fall, dass es seine produzierte Textausgabe als HTML (Content-Type: `text/html`) und *nicht* als Plain-Text (Content-Type `text/plain`) kennzeichnet.
27. Angenommen, Ihr Vorkorrekturmodul soll unter gewissen Bedingungen dynamisch ein Formularfeld namens `FeldC1` in das Aufgabenformular einfügen und zur Vorbereitung der

Datenbank fügen Sie in dasselbe Formular noch `<input type="hidden" name="FeldC1">` ein. Was passiert dann beim Submit? Falls das Vorkorrekturmodul sich gegen ein Einfügen des Formularfelds entschieden hat, wird dennoch eine (leere) Einsendung zum `FeldC1` vorgenommen. Hat das Vorkorrekturmodul dagegen ein Input-Element namens `FeldC1` eingefügt und der Student darin z.B. den Text `Hello World` eingegeben, so werden die Inhalte beider gleichnamiger Felder (leerer String aus Hidden Field sowie `Hello World` aus dynamisch eingefügtem Textfeld) eingesendet und vom Übungssystem durch Komma aufgezählt, d.h. die Einsendung lautet am Ende `,Hello World` (oder `Hello World,`, je nach Reihenfolge der Formularelemente). Um diese Effekte zu vermeiden, darf also das Hidden Field nicht im selben Formular stehen.