Mathehelfer Bruchrechnen

Ein Hackathon für Kinder und Jugendliche

Norbert Seulberger

Mai 2023

Inhaltsverzeichnis

1 Thema und Planung

1.1 Programmiere den Mathehelfer für das Bruchrechnen!

Zukünftig wird der Computer deine Hausaufgaben im Bruchrechnen kontrollieren. Na ja, fast . . .

Wir bringen dem Computer die ersten Schritte im Bruchrechnen bei, nämlich das Kürzen, Erweitern und Multiplizieren von Brüchen. Dazu formulieren wir die Algorithmen (also das Kochrezept, wie es geht) und programmieren Funktionen in der Programmiersprache Python, die diese Berechnungen durchführen.

Damit die Rechnungen schön aussehen, schreiben wir die Gleichungen mit MathML (das ist eine Erweiterung von HTML) und prüfen das Ergebnis im Browser.

Schwierigkeitsstufen

Wir bearbeiten Aufgaben in verschiedenen Schwierigkeitsstufen, von einfach bis sehr schwierig. Du musst also keine Programmierkenntnisse haben, um mit Spaß mitmachen zu können.

Richtige Programmierfreaks kommen auch auf ihre Kosten! Wir werden euch Dinge zeigen, die ihr vermutlich noch nicht gesehen habt. Die Wette gilt.

Vorkenntnisse

Du solltest relativ flüssig Text in einem Editor auf dem Computer bearbeiten können.

Von Vorteil ist es, wenn du bestimmte Vorkenntnisse mitbringst:

- Du weißt, wie eine HTML-Datei aufgebaut ist und hast schon Text in HTML geschrieben.
- Du hast schon kleine Programme in Python programmiert.

Wenn du diese Vorkenntnisse besitzt, kannst du schwierigere Aufgaben lösen!

1.2 Die Aufgabenstellung

Der Mathehelfer Bruchrechnen kann in der Grundversion Brüche kürzen, erweitern und miteinander multiplizieren.

Die Darstellung der Umrechnungen erfolgt in einem Browser. Dabei werden die Brüche grafisch schön mit Zähler, Nenner und Bruchstrich dargestellt wie in einem Mathematikbuch.

$$\frac{6}{8} = \frac{3}{4}$$

$$\frac{3}{4} = \frac{9}{12}$$

$$\frac{3}{4} \cdot \frac{2}{3} = \frac{1}{2}$$

Dabei geben wir die linke Seite vor und berechnen die rechte Seite jeweils mit einem Programm. Wir schreiben jeweils eine eigene Funktion, also drei Funktionen, die

- einen Bruch kürzt,
- einen Bruch mit einem beliebigen Faktor erweitert oder
- zwei Brüche miteinander multipliziert.

Als Programmiersprache verwenden wir Python.

Die Formelzeile tragen wir automatisch in eine kleine Web-Seite ein, die wir uns im Browser ansehen. Dabei beschreiben wir die Formel in einer speziellen HTML-Sprache für mathematische Formeln: MathML.

Die Sprache MathML ist sehr einfach. Allerdings muss man sehr viel Text eingeben, um eine Formel mit MathML zu beschreiben. Daher schreiben wir ein Programm, das die Formel automatisch in MathML ausdrückt.

1.3 Technische Rahmenbedingungen

Hardware und Betriebssystem

Die Beispiele wurden auf einem Raspberry Pi, Version 4, mit 8 GB Ram entwickelt. Damit lässt sich stabil und flüssig arbeiten.

Vermutlich reicht ein Raspi 4 mit mindestens 2 GB Ram aus.

Ein Raspi 3 mit 1 GB Ram läuft nicht mehr performant mit einer anspruchsvollen IDE wie z.B. VS Code.

Als Betriebssystem wurden verschiedene Debian-/Ubuntu-Derivate für arm64-Prozessoren genutzt. Diese arbeiten stabil und flüssig. Unter Windows oder MacOS sollte alles genauso gut klappen.

Programmiersprache

Der Mathehelfer Bruchrechnen wird in Python programmiert. Da die Klasse für Brüche als Dataclass definiert werden, ist mindestens Version 3.7 erforderlich. Jede aktuelle Linux-Distribution enthält diese oder eine neuere Python-Version.

IDE

Für Python existieren viele verschiedene Entwicklungsumgebungen. Um eine sinnvolle Unterstützung durch Code-Vervollständigung zu erhalten, sind folgende IDEs geeignet:

- Visual Studio Code (Python-Plugin, Pylance-Plugin von Microsoft)
- Pycharm
- eclipse mit dem Pydev-Plugin

Die Online-IDE repl.it funktioniert ebenfalls gut. Allerdings ist der Vorteil der fehlenden Installation gegenüber der Online-Abhängigkeit abzuwägen.

Browser

MathML wird aktuell unmittelbar nur von Firefox dargestellt.

Die Chromium-/Chrome-Familie stellt MathML nicht (mehr) dar.

1.4 Zeitliche Planung

Der zeitliche Rahmen besteht aus einer zweitägigen Veranstaltung mit jeweils vier Stunden Dauer.

Die TeilnehmerInnen arbeiten in Zweier-Teams an einem Rechner. Sie dürfen entscheiden, wie selbstständig sie arbeiten möchten. Ein möglichst eigenständiges Vorgehen ist wünschenswert.

Es sind ausreichend Pausen vorzusehen. Die TeilnehmerInnen dürfen Pausen selbst bestimmen, werden von den Betreuern daran erinnert.

Tag 1

- Begrüßung durch einen Inspirer der Hacker School (15 min, 15/240)
- Begrüßung durch das Veranstalterteam / die Inspirer des Veranstalters. Vorstellung der TeilnehmerInnen einschließlich Vorkenntnisse und Erwartungen (30 min, 45/240)
- MathML: Durchlesen der Unterlagen zu MathML, Lösen der Aufgabe (45 min, 90/240)
- Pause (15 min, 105/240)
- Algorithmen und Funktionen: Durchlesen der Unterlagen (20 min, 125/240)
- Nachvollziehen und Umsetzen des Beispiels Kürzen (20 min, 145/240)
- Pause (15 min, 160/240)
- Bearbeiten der Aufgaben zum Erweitern und der Multiplikation (40 min, 200/240)
- Vorstellen von Teamlösungen, Code-Reviews (30 min, 230/240)
- Feedback-Runde (10 min, 240/240)

Tag 2

- Ankommen, Feedback zu Vortag (15 min, 15/240)
- Optional: Fertigstellen der Aufgaben vom Vortag (30 min, 45/240)
- Zeichenketten: Durchlesen der Unterlagen, Nachvollziehen und Umsetzen des Beispiels *Bruch in MathML* (30 min, 45/240)
- Nachvollziehen und Umsetzen des Beispiels *Kürzen eines Bruchs in MathML* (60 min, 105/240)
- dazwischen: Pause (15 min, 120/240)
- Weitere Aufgaben: Erweitern oder Multiplizieren in MathML (60 min, 180/240)
- dazwischen: Pause (15 min, 195/240)
- Vorstellen von Teamlösungen, Code-Reviews (30 min, 225/240)
- Feedback-Runde und Abschluss (15 min, 240/240)

1.5 Vorkenntnisse

Das Thema setzt nur wenige Vorkenntnisse zwingend voraus. Allerdings wächst der Spaß mit steigendem Schwierigkeitsgrad der Aufgaben.

Notwendige Vorkenntnisse

Unverzichtbar ist eine gewisse Fertigkeit im Umgang mit einem PC oder Laptop, u.a.

- flüssige Bedienung von Maus und Tastatur,
- Aufruf und Nutzung des Dateimanagers,
- das Betrachten von HTML-Dateien im Browser,
- das Editieren von Text in einem Editor.

Unbedingt bekannt sollte das Bruchrechnen sein. Für das Formulieren der Algorithmen ist die Kenntnis der Rechenregeln und von einschlägigen Merksätzen wichtig.

Hilfreiche Vorkenntnisse

Grundsätzlich wird in Form der Funktion ein einfaches Sprachelement von Python genutzt, das allerdings in Lehrtexte häufig sehr spät eingeführt wird. Daher erklärt der Kurs, wie Funktionen in Python aufgebaut sind. Vorerfahrung in der Programmierung mit Python ist natürlich hilfreich, weil dann die anspruchsvolleren Aufgaben bewältigt werden können.

Bei der Generierung von MathML sind Vorkenntnisse in HTML hilfreich, etwa der grundsätzliche Aufbau mit öffnenden und schließenden Tags.

Abfrage der Vorkenntnisse und Erwartungen

Bei der Vorstellung der TeilnehmerInnen werden die Vorkenntnisse und Erwartungen besprochen. Mögliche Fragen sind:

- Nutzt du regelmäßig den Computer zum Schreiben von Texten, beispielsweise für die Schule?
- Hast du schon einmal eine Web-Seite erstellt? Kennst du dich mit HTML aus?
- Hast du schon einmal mit Python programmiert? Was hast du dort kennengelernt?
- Was wünscht du dir von diesem Workshop?

2 Zum Warmwerden: MathML

2.1 MathML

MathML ist eine einfache Sprache, die in einem HTML-Dokument dazu verwendet werden kann, um mathematische Formeln darzustellen.

Genauso wie in HTML werden die Daten in sogenannte Tags eingeschlossen, d.h. ein Element beginnt mit einen Tag, dann stehen die Daten und ein schließendes Tag beendet den Ausdruck.

MathML: Anfang und Ende einer Formel

Eine Formeln beginnt mit `<math>` und endet mit `</math>`.

Für die verschiedenen Teile einer mathematischen Formel gibt es unterschiedliche Elemente.

- `<mn>42</mn>` stellt die Zahl 42 dar.
- `<mo>=</mo>` beschreibt ein Gleichheitszeichen.
- `<mo>+</mo>` schreibt ein Pluszeichen.
- `<mo>&middot;</mo>` schreibt das Multiplikationszeichen.

Die Formel $2 + 3 = 5$ lautet also in MathML:

```
<math>
    <mn>2</mn>
    <mo>+</mo>
    <mn>3</mn>
    <mo>=</mo>
    <mn>5</mn>
</math>
```

Tipp

Der Browser ignoriert die Zeilenumbrüche in MathML. Wir können die Formel also auch in eine Zeile schreiben.

```
<math><mn>2</mn><mo>+</mo><mn>3</mn><mo>=</mo><mn>5</mn></math>
```

Brüche in MathML

Ein Bruch beginnt mit `<mfrac>` und endet mit `</mfrac>`. Den Zähler und den Nenner schreiben wir zwischen `<mn>` und `</mn>`.

Der Bruch $\frac{2}{3}$ sieht also so aus:

```
<math>
    <mfrac>
        <mn>2</mn>
        <mn>3</mn>
    </mfrac>
</math>
```

Eine Formelzeile

Damit jede Formel in eine eigene Zeile geschrieben wird, verwenden wir das Absatz-Tag von HTML: ...

Also zusammen:

```
<p>
    <math><mn>2</mn><mo>+</mo><mn>3</mn><mo>=</mo><mn>5</mn></math>
</p>
```

Eine Anleitung zu MathML

Eine einfache, aber umfassende Anleitung zu MathML findest du auf folgender Internetseite: MathML-Tutorial

2.2 Aufgabenblatt MathML

Aufgabe: Anzeige der HTML-Datei

Schau dir im Firefox-Browser die Datei `manuell.html` an. Die Anzeige sollte etwa so aussehen:

$$\frac{6}{8} = \frac{3}{4}$$

„Hier kannst du die Formeln für das Umrechnen eines Bruchs in eine Dezimalzahl und umgekehrt einfügen.“

Aufgabe: Ansicht der HTML-Datei im Editor

Schau dir jetzt dieselbe Datei im Editor der Entwicklungsumgebung an. Der Text sieht etwa so aus:

```
<!doctype html>
<html lang="de">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Bruchrechnen</title>
    </head>
    <body>
        <p>
            <math>
                <mfrac>
                    <mn>6</mn>
                    <mn>8</mn>
                </mfrac>
                <mo>=</mo>
                <mfrac>
                    <mn>3</mn>
                    <mn>4</mn>
                </mfrac>
            </math>
        </p>
        <p>
            Hier kannst du die Formeln für das Umrechnen ...
        </p>
    </body>
</html>
```

Kannst du die Element entdecken, die für die Darstellung des Kürzens der beiden Brüche stehen?

Aufgabe: Das Erweitern eines Bruchs

Füge in die Datei

`manuell.html`

den Text in der MathML-Sprache ein, so dass das Erweitern eines Bruchs dargestellt wird. Der Text „Hier kannst du ...“ darf überschrieben werden.

Die letzte Zeile soll so aussehen:

$$\frac{3}{4} = \frac{9}{12}$$

Überprüfe das Ergebnis im Firefox-Browser. Die Datei `manuell.html` sollte jetzt so dargestellt werden:

$$\frac{6}{8} = \frac{3}{4}$$

$$\frac{3}{4} = \frac{9}{12}$$

Aufgabe: Das Multiplizieren von zwei Brüchen

Füge in die Datei

`manuell.html`

den Text in der MathML-Sprache ein, so dass diese Zeile als letzte Zeile angezeigt wird:

$$\frac{3}{4} \cdot \frac{2}{3} = \frac{1}{2}$$

Überprüfe das Ergebnis im Firefox-Browser. Die Seite sollte jetzt ungefähr so aussehen:

$$\frac{6}{8} = \frac{3}{4}$$

$$\frac{3}{4} = \frac{9}{12}$$

$$\frac{3}{4} \cdot \frac{2}{3} = \frac{1}{2}$$

3 Algorithmen und Funktionen

3.1 Algorithmen

Ein Algorithmus ist die eindeutige Beschreibung eines Verfahrens, um eine bestimmte Aufgabe zu lösen.

Beispiel: Das Erweitern eines Bruches

Ein Bruch wird erweitert, indem der Zähler und der Nenner mit demselben Faktor multipliziert werden. Wird beispielsweise der Bruchs $\frac{3}{4}$ mit dem Faktor 3 erweitert, so ergibt sich diese Gleichung:

$$\frac{3}{4} = \frac{3 \cdot 3}{4 \cdot 3} = \frac{9}{12}$$

Der Algorithmus für das Erweitern

- Nimm einen Bruch und den Faktor, mit dem du den Bruch erweitern willst.
- Merke dir den Zähler des Bruchs in einer Variablen.
- Merke dir den Nenner des Bruchs in einer zweiten Variablen.
- Multipliziere die Variable für den Zähler mit dem Erweiterungsfaktor.
- Verfahre genauso mit dem Nenner.
- Erstelle einen neuen Bruch mit dem neuen Zähler und dem neuen Nenner.
- Gib den neuen Bruch zurück.

3.2 Variablen

Einfache Variable

Eine Variable dient dazu, sich einen Wert, einen Text oder das Ergebnis einer Rechnung zu merken. Wir geben einer Variablen einen sinnvollen Namen.

```
wichtige_zahl = 42
summe = 2 + 3
```

Mit Variablen können wir auch rechnen.

```
summe = summe + 6
```

Strukturierte Variablen

Wir können Variablen auch komplizierter aufbauen und einen Wert in einer Variablen merken, der sich aus mehreren Teilwerten zusammensetzt. Ein Beispiel ist der Bruch:

```
class Bruch():
    zaehler: int
    nenner: int
```

Ein Bruch ist ein Wert, der sich aus zwei Teilwerten zusammensetzt, nämlich dem Zähler und dem Nenner. Den Bruchstrich müssen wir uns nicht merken, weil er immer da ist. Zähler und Nenner sind ganze Zahlen (`int`).

```
bruch = Bruch(3,4)
```

Jetzt können wir die Variable, die ja einen Bruch enthält, nach dem Zähler und dem Nenner fragen. Wir schreiben die beiden Werte in eigene Variablen.

```
zaehler = bruch.zaehler
nenner = bruch.nenner
```

Wir können auch zwei Ganzzahlen jeweils in eine Variable schreiben und damit eine Bruchvariable erzeugen.

```
neuer_zaehler = 7
neuer_nenner = 8
```

```
bruch = Bruch(neuer_zaehler, neuer_nenner)
```

3.3 Funktionen

Eine Funktion ist ein kleines Teilprogramm, das eine spezifische Aufgabe löst. Eine Funktion arbeitet mit folgenden Schritten:

- Die Funktion nimmt einen oder mehrere Eingabewerte entgegen.
- Die Funktion führt eine Berechnung oder andere Aktivität durch.
- Die Funktion gibt das Ergebnis der Berechnung zurück.

Für die gleichen Eingabewerte erhält man immer dasselbe Ergebnis. Die Eingabewerte heißen Parameter. Es kann einen oder mehrere Parameter geben, manchmal sogar gar keinen.

Der Aufbau einer Funktion in Python

Der Aufbau einer Funktion orientiert sich an den drei Schritte, die wir eben kennen gelernt haben:

- die Signaturzeile
- der Rumpf der Funktion
- der Rückgabewert

Die Signaturzeile

Die Signaturzeile nennt den Namen und beschreibt die Parameter.

```
def kuerzeBruch(bruch):
```

- Zuerst steht immer das Wort `def`.
- Dann folgt der Name der Funktion. Der Name sollte ausdrücken, welche Berechnung die Funktion durchführt.
- In den runden Klammer stehen die Eingabewerte für die Funktion. Hier ist es ein Parameter mit dem Namen `bruch`. Die Funktion kann den Wert des Parameters für die Berechnung nutzen.

Tipp: Datentypen angeben

Wer möchte, kann die Typen der Parameter und des Ergebnisses angeben. Es hilft, Fehler zu vermeiden. Programmierprofis tun das.

Die Signaturzeile für die Funktion zum Kürzen lautet dann:

```
def kuerzeBruch(bruch: Bruch) -> Bruch:
```

Die Funktion nimmt also einen Parameter entgegen, dessen Name `bruch` lautet (klein geschrieben!) und dessen Datentyp ein `Bruch` ist (groß geschrieben!).

Die Funktion führt eine Berechnung durch, bei der ein Rückgabewert berechnet wird, der vom Datentyp `Bruch` ist. Das steht hier: `-> Bruch`.

Der Rumpf einer Funktion

Der Rumpf der Funktion ist eingerückt. Wir benutzen dafür vier Leerzeichen.

Hier wird die Berechnung durchgeführt. Das erklären wir ausführlich im nächsten Beispiel.

Der Rückgabewert

Die letzte Zeile im Rumpf gibt das Ergebnis zurück. Sie beginnt mit dem Wort

```
return
```

3.4 Beispiel: Das Kürzen eines Bruchs

Im folgenden Beispiel lernst du das Programmieren einer Funktion in Python.

Aufgabe: Der Algorithmus zum Kürzen eines Bruchs

Lies den Algorithmus zum Kürzen eines Bruchs durch:

- Nimm einen Bruch.
- Merke dir den Zähler des Bruchs in einer Variablen.
- Verfahre ebenso mit dem Nenner.
- Berechne den größten gemeinsamen Teiler (ggT) von Zähler und Nenner. Es gibt dafür eine fertige Funktion.
- Berechne den gekürzten Zähler, indem du den alten Zähler durch den ggT teilst.
- Berechne ebenso den gekürzten Nenner.
- Erstelle einen neuen Bruch aus dem gekürzten Zähler und dem gekürzten Nenner.
- Gib den neuen Bruch zurück.

Verstehst du das Vorgehen? Andernfalls stelle deine Fragen dem Betreuer.

Die Funktion zum Kürzen von Brüchen

Die Berechnung des ggT erfolgt mittels der Funktion

`mathefunktionen.berechne_ggT`.

In der Funktion zum Kürzen wird also eine Zeile enthalten sein:

```
ggT = mathefunktionen.berechne_ggT(alterZaehler, alterNenner)
```

Beachte, dass beim Teilen durch den ggT die Division `//` benutzt wird, damit der neue Zähler und der neue Nenner ganze Zahlen werden!

Aufgabe: Programmieren der Funktion

Gib die Funktion in den Editor der Entwicklungsumgebung (IDE) ein. Zeigt die IDE noch Fehler an? Vielleicht hast du dich vertippt? Falls du den Fehler nicht finden kannst, sprich den Betreuer an.

```
def kuerzeBruch(bruch: Bruch) -> Bruch:
    alterZaehler = bruch.zaehler
    alterNenner = bruch.nenner
    ggT = mathefunktionen.berechne_ggT(alterZaehler, alterNenner)
    neuerZaehler = alterZaehler // ggT
    neuerNenner = alterNenner // ggT
    neuerBruch = Bruch(neuerZaehler, neuerNenner)
    return neuerBruch
```

Überprüfung des Ergebnisses

Wir überprüfen das Ergebnis mit Hilfe folgender Datei:

`uebungsplatz.py`

Wir tragen dort die folgenden Zeilen ein:

```
bruch = Bruch(6,8)
gekuerzter_bruch = mathehelfer.kuerzeBruch(bruch)
print(gekuerzter_bruch)
```

Jetzt führen wir `uebungsplatz.py` als Python-Skript aus. Das Ergebnis sollte jetzt diese Zeile enthalten:

```
Bruch(zaehler=3, nenner=4)
```

Sieht das Ergebnis bei dir genauso aus?

3.5 Aufgabe: Das Erweitern eines Bruchs

Der Algorithmus zum Erweitern

Wie lautet der Algorithmus zur Erweitern eines Bruchs mit einem vorgegebenen Faktor?

- Nimm einen Bruch und den Faktor, mit dem du den Bruch erweitern willst.
-
-
-
-
-
-

Die Funktion zum Erweitern

Programmiere die Funktion zum Erweitern eines Bruchs. Die Funktion erhält als Parameter einen Bruch sowie den Faktor zum Erweitern. Die Signatur lautet:

```
def erweitereBruch(bruch: Bruch, faktor: int) -> Bruch:
```

Vergiss nicht die letzte Zeile mit dem `return`-Befehl!

Überprüfung des Ergebnisses

Überprüfe das Ergebnis. Trage dazu die notwendige Befehle in die Datei

`uebungsplatz.py`

ein. Du kannst dich daran orientieren, wie wir es beim Kürzen gemacht haben. Hier möchten wir den Bruch $\frac{3}{4}$ mit dem Faktor 3 erweitern.

Es geht los mit

```
bruch = Bruch(3,4)
```

Wie geht es weiter?

Danach führen wir `uebungsplatz.py` als Python-Skript aus. Das Ergebnis sollte jetzt diese Zeile enthalten:

```
Bruch(zaehler=9, nenner=12)
```

Sieht das Ergebnis bei dir genauso aus?

3.6 Aufgabe: Das Multiplizieren von Brüchen

Der Algorithmus zum Multiplizieren von Brüchen

Wie lautet der Algorithmus zum Multiplizieren von Brüchen?

- Nimm zwei Brüche entgegen.
-
-
-
-
-
-
-
-

Hast du daran gedacht, den neuen Bruch zu kürzen?

Die Funktion zum Multiplizieren von Brüchen

Wie lautet die Funktion?

Überprüfung der Funktion zum Multiplizieren

Wir möchten in der Datei

`uebungsplatz.py`

die beiden Brüche $\frac{3}{4}$ und $\frac{2}{3}$ miteinander multiplizieren. Es beginnt also mit

```
erster_faktor = Bruch(3,4)
zweiter_faktor = Bruch(2,3)
```

Wie geht es weiter?

Führe `uebungsplatz.py` als Python-Skript aus. Die Ausgabe muss folgende Zeile enthalten:

Produkt: Bruch(zaehler=1, nenner=2)

4 Zeichenketten

4.1 Zeichenketten

Was ist eine Zeichenkette?

Eine Zeichenkette enthält ein oder mehrere beliebige Zeichen, also Buchstaben, Zahlen oder Sonderzeichen. Die Zeichen stehen hintereinander. Eine Zeichenkette wird eingeschlossen in Hochkommata (') oder hochgestellte Anführungszeichen (").

```
eine_zeichenkette = "r2d2"
berg = '8000er-Gipfel'
ein_ganzer_satz = "Python ist toll!"
```

Häufig benutzen wir den englischen Begriff für Zeichenkette: String. Mit Strings kann man viele interessante Dinge anstellen.

Das Zusammenfügen von Zeichenketten

In Python können zwei Strings mit dem Pluszeichen zu einem langen String zusammengefügt werden:

```
gruss = "Hallo " + "Welt!"
```

In `gruss` steht jetzt der String `"Hallo Welt!"`.

Dass das Pluszeichen in diesem Fall nicht das Addieren von Zahlen, sondern das Zusammenfügen von Strings bedeutet, daran gewöhnt man sich schnell!

Wir möchten einen String zusammenbauen, der in MathML eine Rechenformel beschreibt.

Die Formel

$$2 + 3 = 5$$

lautet in MathML:

```
<math><mn>2</mn><mo>+</mo><mn>3</mn><mo>=</mo><mn>5</mn></math>
```

Diese Formel können wir in Python schrittweise aus kleinen Strings zu einem langen String zusammensetzen.

```
formel = "<math>"
formel = formel + "<mn>2</mn>"
formel = formel + "<mo>+</mo>"
formel = formel + "<mn>3</mn>"
formel = formel + "<mo>=</mo>"
formel = formel + "<mn>5</mn>"
formel = formel + "</math>"
```

Dabei nehmen wir die bereits erstellte Formel, fügen den nächsten kurzen String an und merken uns das Ergebnis wieder in der Variablen `formel`.

4.2 Beispiel: Die Darstellung eines Bruchs in MathML

Einen Bruch in MathML zu schreiben, ist nicht schwierig, aber mühsam. Daher möchten wir eine Funktion ein Python programmieren, die das automatisch erledigt.

Der Algorithmus zur Darstellung eines Bruchs in MathML

- Nimm einen Bruch entgegen.
- Wandle den Zähler des Bruchs in eine Zeichenkette und speichere diese Zeichenkette in einer Variablen.
- Verfahre genauso mit dem Nenner.
- Baue die Zeichenkette für das Ergebnis aus Einzelteilen zusammen, wie in den nächsten Schritten beschrieben.
- Beginne die Zeichenkette mit `<mfrac>`.
- Rahme die Zeichenkette, die den Zähler beschreibt, mit den Tags `<mn>` und `</mn>` ein.
- Verfahre ebenso mit der Zeichenkette für den Nenner.
- Beende die Zeichenkette mit `</mfrac>`.
- Gib die Zeichenkette zurück.

Die Funktion zur Darstellung eines Bruchs in MathML

```
def schreibeBruch(bruch: Bruch) -> str:
    zaehlerAlsString = str(bruch.zaehler)
    nennerAlsString = str(bruch.nenner)
    ergebnis = "<mfrac>"
    ergebnis = ergebnis + "<mn>" + zaehlerAlsString + "</mn>"
    ergebnis = ergebnis + "<mn>" + nennerAlsString + "</mn>"
    ergebnis = ergebnis + "</mfrac>"
    return ergebnis
```

5 Das Erstellen der Formeln

5.1 Beispiel: Die Formel zum Kürzen eines Bruchs in MathML

Die Vorarbeiten

- Wir verfügen über die Funktion `kuerzeBruch`. (siehe 3.4)
- Außerdem haben wir die Funktion `schreibeBruch`, die einen Bruch in MathML darstellt. (siehe 4.2)

Der Algorithmus zur Darstellung der Formel zum Kürzen

- Nimm einen Bruch.
- Erzeuge aus diesem Bruch einen gekürzten Bruch. Verwende dazu die Funktion, die du gerade programmiert hast.
- Erzeuge aus dem ursprünglichen Bruch die Beschreibung in MathML. Verwende dazu die Funktion, die einen Bruch in MathML beschreibt.
- Erzeuge aus dem gekürzten Bruch ebenfalls die Beschreibung in MathML. Das geht genauso wie eben.
- Setze jetzt den MathML-Text aus den einzelnen Angaben zusammen.
- Die Zeichenkette beginnt mit dem MathML-Text für den ungekürzten Bruch.
- Dann kommt das Gleichheitszeichen.
- Es folge der MathML-Text für den gekürzten Bruch.

Die Funktion zur Darstellung der Formel zum Kürzen

In der Datei **`mathehelfer.py`** programmieren wir folgende Funktion:

```
def schreibeKuerzen(bruch: Bruch) -> str:
    bruchGekuerzt = kuerzeBruch(bruch)
    textUngekuerzt = schreibeBruch(bruch)
    textGekuerzt = schreibeBruch(bruchGekuerzt)
    ergebnis = textUngekuerzt + "<mo>=</mo>" + textGekuerzt
    return ergebnis
```

Überprüfung des Ergebnisses

Überprüfe das Ergebnis in der Datei **`uebungsplatz.py`**.

```
bruch = Bruch(6,8)
formel = mathehelfer.schreibeKuerzen(bruch)
print(formel)
```

Der Bruch $\frac{6}{8}$ muss gekürzt den Bruch $\frac{3}{4}$ ergeben. In einer Zeile steht:

```
<mfrac><mn>6</mn><mn>8</mn></mfrac><mo>=</mo>
                                    <mfrac><mn>3</mn><mn>4</mn></mfrac>
```

Jetzt arbeiten wir wieder in der Datei **mathehelfer.py**.

Ergänze in der Funktion `schreibeMathML()` die Zeilen für Kürzen für den Bruch $\frac{6}{8}$.

```
bruch = Bruch(6,8)
inhalt = inhalt + "\n\t\t<p><math>"
inhalt = inhalt + schreibeKuerzen(bruch)
inhalt = inhalt + "</math></p>"
```

Führe die Datei `htmlErzeugung.py` aus.

Prüfe im Firefox-Browser, wie die Datei `index.html` aussieht. Dort sollte eine Zeile stehen, die so aussieht:

$$\frac{6}{8} = \frac{3}{4}$$

5.2 Aufgabe: Die Formel zum Erweitern eines Bruchs in MathML

Erledigte Vorarbeiten

Welche bereits erstellten Funktionen können wir sinnvoll nutzen?

-
-

Der Algorithmus zur Darstellung der Erweiterungsformel in MathML

Wie lautet der Algorithmus?

- Nimm einen Bruch und einen Faktor zur Erweiterung entgegen.
-
-
-
-
-
-
-

Die Funktion zur Darstellung der Formel in MathML

Wie lautet die Funktion? Die Signatur sieht so aus:

```
def schreibeErweitern(bruch: Bruch, faktor: int) -> str:
```

Wie geht es weiter?

Überprüfung des Ergebnisses

Überprüfe das Ergebnis in der Datei

`uebungsplatz.py`.

Wir erweitern beispielsweise den Bruch $\frac{3}{4}$ mit dem Faktor 3.

```
bruch = Bruch(2,3)
faktor = 3
```

Wie geht es weiter? Wie sieht das Ergebnis aus?

Darstellung der Formel im Browser

Wir können die Formel automatisch in die HTML-Seite übertragen lassen und anschließend im Browser anschauen.

Ergänze in der Funktion

`schreibeMathML()`

die entsprechenden Zeilen analog zum Beispiel Kürzen.

Dann führe die Datei

`htmlErzeugung.py`

als Python-Skript aus.

Prüfe im Firefox-Browser, wie die Datei `index.html` aussieht. Dort sollte eine Zeile stehen, die so aussieht:

$$\frac{3}{4} = \frac{9}{12}$$

5.3 Aufgabe: Die Formel zum Multiplizieren von Brüchen in MathML

Der Algorithmus zur Darstellung der Multiplikationsformel

Wie lautet der Algorithmus, um die Multiplikation zweier Brüche in MathML darzustellen?

- Nimm zwei Brüche entgegen.
-
-
-
-
-
-
-
-

Die Funktion zur Darstellung der Multiplikationsformel

Wie lautet die Funktion?

Überprüfung der Darstellung

Ergänze in der Funktion `mathehelfer.schreibeMathML()` die Zeilen für das Multiplizieren. Wie lauten sie?

Führe die Datei `htmlErzeugung.py` aus.

Prüfe im Firefox-Browser, wie die Datei `index.html` aussieht. Dort sollte eine Zeile stehen, die so aussieht:

$$\frac{3}{4} \cdot \frac{2}{3} = \frac{1}{2}$$

6 Weitere Aufgaben

6.1 Aufgabe: Das Dividieren von Brüchen

Diese Aufgabe ist einfach.

Überlege: Wie werden Brüche dividiert?

Brüche werden dividiert, indem ...

Formuliere den Algorithmus!

Programmiere die Funktion, die den Teiler-Bruch umwandelt.

Das ist eine sehr einfache Funktion.

Programmiere jetzt die Division der Brüche.

Benutze dabei die Funktionen, die du für das Multiplizieren der Brüche programmiert hast.

6.2 Aufgabe: Das Addieren von Brüchen

Diese Aufgabe ist mittelschwer.

Überlege: Wie werden zwei Brüche addiert?

Zwei Brüche werden addiert, indem ...

Formuliere den Algorithmus zum Addieren von zwei Brüchen.

Programmiere die Funktion, die die beiden Brüche für das Addieren vorbereitet.

Diese Funktion darf sehr einfach sein, weil wir bereits sehr gut Brüche kürzen können.

Könner dürfen eine anspruchsvollere Lösung mit dem kleinsten gemeinsamen Vielfachen (kgV) programmieren!

Programmiere das Addieren von zwei Brüchen.

Natürlich möchten wir das Ergebnis in MathML im Browser anschauen!

Programmiere das Subtrahieren von zwei Brüchen.

Wenn wir zwei Brüche addieren können, dann ist das Subtrahieren sehr einfach.

6.3 Aufgabe: Das Umwandeln eines Bruchs in eine Kommmazahl

Diese Aufgabe ist einfach.

Überlege: Wie wird ein Bruch in eine Kommazahl umgewandelt?

Formuliere den Algorithmus!

Programmiere die Funktion zum Umwandeln.

Zusatz: Eine Datenklasse für Kommazahlen

Diese Zusatzaufgabe ist mittelschwer.

Grundsätzlich können wir mit dem Datentyp `float` arbeiten. Schöner ist es, eine eigene Datenklasse für Kommazahlen zu haben, ähnlich zur Datenklasse für Brüche.

Wie sieht eine Datenklasse für eine Kommazahl aus? Bitte nicht wundern - das Ergebnis ist sehr einfach.

6.4 Aufgabe: Die Umwandlung einer Kommazahl in einen Bruch

Diese Aufgabe ist mittelschwer.

Überlege: Wie wird eine Kommazahl in einen Bruch umgewandelt?

Formuliere den Algorithmus.

Programmiere die Funktion zur Umwandlung einer Kommazahl in einen Bruch

Tipp: Wir haben eine Hilfsfunktion vorbereitet, die den richtigen Nenner berechnet. Vergiss danach das Kürzen nicht.

6.5 Aufgabe: Umwandeln in eine gemischte Zahl

Diese Aufgabe ist schwer.

Der Wert eines unechten Bruchs, also eines Bruchs, dessen Zähler größer als der Nenner ist, ist oft erst nach etwas Kopfrechnen zu erkennen. Daher schreiben wir unechte Brüche als gemischte Zahl, also als Summe einer ganzen Zahl und einem echten Bruch. Beispiel:

$$\frac{70}{12} = 5\frac{5}{6}$$

Einen echten Bruch können wir ebenfalls als gemischte Zahl schreiben. Dann ist die ganze Zahl einfach gleich Null und wird gar nicht hingeschrieben.

Programmiere die Datenklasse für eine gemischte Zahl

Tipp: Benutze dafür die Datenklasse `Bruch`.

Überlege: Wie wird ein Bruch in eine gemischte Zahl umgewandelt?

Überlege: Wie wird eine gemischte Zahl in einen Bruch umgewandelt?

Formuliere die beiden Algorithmen zur Umwandlung zwischen Bruch und gemischter Zahl.

Programmiere die Funktionen zur Umwandlung.

Sieht deine Darstellung einer gemischten Zahl in MathML schön aus?

6.6 Aufgabe: Objektorientierte Programmierung

Diese Aufgabe ist sehr schwierig. Es werden gute Kenntnisse der Programmierung in Python vorausgesetzt.

Ziel der Aufgabe

Wir möchten nur noch **eine** Funktion haben, die das Ergebnis einer Berechnung von zwei Zahlen liefert, unabhängig davon, ob es Brüche, Kommazahlen oder gemischte Zahlen sind, auch wild durcheinander. Das ist sehr anspruchsvoll!

Tipp: Intern rechnen wir alle Zahlen in Brüche um und rechen dann mit Brüchen.

Die Datenklassen für Bruch, Kommazahl, gemischte Zahl

Betrachte die Datenklassen für Bruch, Kommazahl, gemischte Zahl. Welche Funktionen, die wir bisher programmiert haben, können wir als Methoden in die Klassen ziehen?

Der Operator

Wie können wir den Operator als Klasse beschreiben? Tipp: Es gibt den Aufzählungstyp `enum`.

Die Superklasse

Programmiere eine Superklasse für Bruch, Kommazahl und gemischte Zahl. Diese Superklasse sollte eine abstrakte Methode deklarieren, die ein Objekt in einen Bruch umwandelt.

Implementiere in den drei Subklassen diese Methode.

7 Lösungen

7.1 Lösung: Das Kürzen eines Bruchs

Der Algorithmus zum Kürzen eines Bruchs

- Nimm einen Bruch.
- Merke dir den Zähler des Bruchs in einer Variablen.
- Verfahre ebenso mit dem Nenner.
- Berechne den größten gemeinsamen Teiler (ggT) von Zähler und Nenner. (Es gibt dafür eine fertige Funktion.)
- Berechne den gekürzten Zähler, indem du den alten Zähler durch den ggT teilst.
- Berechne ebenso den gekürzten Nenner.
- Erstelle einen neuen Bruch aus dem gekürzten Zähler und dem gekürzten Nenner.
- Gib den neuen Bruch zurück.

Die Funktion zum Kürzen von Brüchen

Die Berechnung des ggT erfolgt mittels der Funktion `mathefunktionen.berechne_ggT`. In der Funktion zum Kürzen wird also eine Zeile enthalten sein:

```
ggT = mathefunktionen.berechne_ggT(alterZaehler, alterNenner)
```

Beachte, dass beim Teilen durch den ggT die Division `//` benutzt wird, damit der neue Zähler und der neue Nenner ganze Zahlen werden!

```
def kuerzeBruch(bruch: Bruch) -> Bruch:
    alterZaehler = bruch.zaehler
    alterNenner = bruch.nenner
    ggT = mathefunktionen.berechne_ggT(alterZaehler, alterNenner)
    neuerZaehler = alterZaehler // ggT
    neuerNenner = alterNenner // ggT
    neuerBruch = Bruch(neuerZaehler, neuerNenner)
    return neuerBruch
```

Die Algorithums zur Darstellung der Formel zum Kürzen

- Nimm einen Bruch.
- Erzeuge aus diesem Bruch einen gekürzten Bruch. Verwende dazu die Funktion, die du gerade programmiert hast.
- Erzeuge aus dem ursprünglichen Bruch die Beschreibung in MathML. Verwende dazu die Funktion, die einen Bruch in MathML beschreibt.

- Erzeuge aus dem gekürzten Bruch ebenfalls die Beschreibung in MathML. Das geht genauso wie eben.
- Setze jetzt den MathML-Text aus den einzelnen Angaben zusammen.
- Die Zeichenkette beginnt mit dem MathML-Text für den ungekürzten Bruch.
- Dann kommt das Gleichheitszeichen.
- Es folge der MathML-Text für den gekürzten Bruch.

Die Funktion zur Darstellung der Formel zum Kürzen

```
def schreibeKuerzen(bruch: Bruch) -> str:
    bruchGekuerzt = kuerzeBruch(bruch)
    textUngekuerzt = schreibeBruch(bruch)
    textGekuerzt = schreibeBruch(bruchGekuerzt)
    ergebnis = textUngekuerzt + "<mo>=</mo>" + textGekuerzt
    return ergebnis
```

Überprüfung des Ergebnisses

Überprüfe das Ergebnis in der Datei `uebungsplatz.py`.

```
bruch = Bruch(6,8)
formel = schreibeKuerzen(bruch)
print(formel)
```

Der Bruch $\frac{6}{8}$ muss gekürzt den Bruch $\frac{3}{4}$ ergeben. In einer Zeile steht:

```
<mfrac><mn>6</mn><mn>8</mn></mfrac><mo>=</mo>
    <mfrac><mn>3</mn><mn>4</mn></mfrac>
```

Ergänze in der Funktion `schreibeMathML()` die Zeilen für Kürzen für den Bruch $\frac{6}{8}$.

```
bruch = Bruch(6,8)
inhalt = inhalt + "\n\t\t<p><math>"
inhalt = inhalt + schreibeKuerzen(bruch)
inhalt = inhalt + "</math></p>"
```

Führe die Datei `htmlErzeugung.py` aus.

Prüfe im Firefox-Browser, wie die Datei `index.html` aussieht. Dort sollte eine Zeile stehen, die so aussieht:

$$\frac{6}{8} = \frac{3}{4}$$

7.2 Lösung: Das Erweitern eines Bruchs

Der Algorithmus zum Erweitern

- Nimm einen Bruch und den Faktor, mit dem du den Bruch erweitern willst.
- Merke dir den Zähler in einer Variablen.
- Merke dir den Nenner des Bruchs in einer zweiten Variablen.
- Multipliziere die Variable für den Zähler mit dem Erweiterungsfaktor. Merke dir das Ergebnis in einer Variablen.
- Verfahre genauso mit dem Nenner.
- Erstelle einen neuen Bruch mit dem neuen Zähler und dem neuen Nenner. Merke dir das Ergebnis in einer Variablen.
- Gib die Variable zurück, die den neuen Bruch enthält.

Die Funktion zum Erweitern

```
def erweitereBruch(bruch: Bruch, faktor: int) -> Bruch:
    alterZaehler = bruch.zaehler
    alterNenner = bruch.nenner
    neuerZaehler = alterZaehler * faktor
    neuerNenner = alterNenner * faktor
    neuerBruch = Bruch(neuerZaehler, neuerNenner)
    return neuerBruch
```

Der Algorithmus zur Darstellung der Erweiterungsformel in MathML

- Nimm einen Bruch und einen Faktor zur Erweiterung entgegen.
- Erzeuge aus diesem Bruch einen erweiterten Bruch. Verwende dazu die Funktion `erweitereBruch`. Merke das Ergebnis in einer Variablen.
- Erzeuge aus dem ursprünglichen Bruch die Beschreibung in MathML. Verwende dazu die Funktion `schreibeBruch`. Merke das Ergebnis in einer Variablen.
- Erzeuge aus dem erweiterten Bruch die Beschreibung in MathML. Verwende dazu ebenfalls die Funktion `schreibeBruch`. Merke das Ergebnis in einer Variablen.
- Setze jetzt den MathML-Text aus den einzelnen Angaben zusammen wie in den folgenden Schritten angegeben. Verknüpfe die einzelnen Strings mit dem Pluszeichen.
- Es beginnt mit dem MathML-Text für den ungekürzten Bruch.
- Dann kommt das Gleichheitszeichen.
- Es folgt der MathML-Text für den gekürzten Bruch.

Die Funktion zur Darstellung der Formel in MathML

```
def schreibeErweitern(bruch: Bruch, faktor: int) -> str:
    bruchErweitert = erweitereBruch(bruch, faktor)
    textAlterBruch = schreibeBruch(bruch)
    textNeuerBruch = schreibeBruch(bruchErweitert)
    ergebnis = textAlterBruch + "<mo>=</mo>" + textNeuerBruch
    return ergebnis
```

Überprüfung des Ergebnisses

Überprüfe das Ergebnis in der Datei `uebungsplatz.py`. Wir erweitern beispielsweise den Bruch $\frac{2}{3}$ mit dem Faktor 3.

```
bruch = Bruch(2,3)
formel = schreibeErweitern(bruch, 3)
print(formel)
```

Die resultierende Formel lautet (allerdings in einer Zeile):

```
<mfrac><mn>2</mn><mn>3</mn></mfrac><mo>=</mo>
                                        <mfrac><mn>6</mn><mn>9</mn></mfrac>
```

Darstellung der Formel im Browser

Wir können die Formel automatisch in die HTML-Seite übertragen lassen und anschließend im Browser anschauen.

Ergänze dazu in der Funktion `schreibeMathML()` folgende Zeilen:

```
inhalt = inhalt + "\n\t\t<p><math>"
inhalt = inhalt + schreibeErweitern(Bruch(2,3), 3)
inhalt = inhalt + "</math></p>"
```

Jetzt führen wir die Datei `htmlErzeugung.py` aus.

Prüfe im Firefox-Browser, wie die Datei `index.html` aussieht. Dort sollte eine Zeile stehen, die so aussieht:

$$\frac{2}{3} = \frac{6}{9}$$

7.3 Lösung: Das Multiplizieren von Brüchen

Der Algorithmus zum Multiplizieren von Brüchen

- Nimm zwei Brüche entgegen.
- Merke dir den Zähler des ersten Bruchs in einer Variablen.
- Merke dir den Nenner des ersten Bruchs in einer zeiten Variablen.
- Verfahre ebenso mit dem zweiten Bruch. Bisher hast du vier Variablen.
- Multipliziere die beiden Zähler-Variablen und merke dir das Ergebnis in einer Variablen.
- Multipliziere die beiden Nenner-Variablen.
- Erzeuge einen Bruch aus den Produkten der Zähler und der Nenner und merke dir das Ergebnis in einer Variablen.
- Wichtig! Kürze den soeben erstellten Bruch.
- Gib den neuen, gekürzten Bruch zurück.

Die Funktion zum zum Multiplizieren von Brüchen

```
def multipliziereBrueche(erster_faktor: Bruch, zweiter_faktor: Bruch)
   -> Bruch:
    erster_zaehler = erster_faktor.zaehler
    erster_nenner = erster_faktor.nenner
    zweiter_zaehler = zweiter_faktor.zaehler
    zweiter_nenner = zweiter_faktor.nenner
    produkt_zaehler = erster_zaehler * zweiter_zaehler
    produkt_nenner = erster_nenner * zweiter_nenner
    produkt = Bruch(produkt_zaehler, produkt_nenner)
    produkt = kuerzeBruch(produkt)
    return produkt
```

Überprüfung der Funktion

Ergänze in der Datei `uebungsplatz.py`:

Führe `uebungsplatz.py` als Python-Skript aus. Die Ausgabe muss folgende Zeile enthalten:

> Produkt: Bruch(zaehler=1, nenner=2)

```
    erster_faktor = Bruch(3,4)
    zweiter_faktor = Bruch(2,3)
    produkt = mathehelfer.multipliziereBrueche(erster_faktor,
       zweiter_faktor)
    print("Produkt: " + str(produkt))
```

Der Algorithmus zur Darstellung der Multiplikationsformel

- Nimm zwei Brüche entgegen.
- Berechne das Produkt der beiden Brüche mit der Funktion `multipliziereBrueche`.
- Erstelle für den ersten Bruch die MathML-Darstellung.
- Erstelle für den zweiten Bruch die MathML-Darstellung.
- Erstelle für das Produkt die MathML-Darstellung.
- Setze jetzt den MathML-Text aus den drei MathML-Texten der Brüche zusammen.
- Zwischen dem ersten und dem zweiten Bruch steht noch das Multiplikationszeichen.
- Zwischen dem zweiten Bruch und dem Produkt steht das Gleichheitszeichen.
- Gib den gesamten MathML-Text zurück.

Die Funktion zur Darstellung der Multiplikationsformel

```
def schreibeMultiplikation(erster_faktor: Bruch, zweiter_faktor: Bruch)
    -> str:
    text_erster_faktor = schreibeBruch(erster_faktor)
    text_zweiter_faktor = schreibeBruch(zweiter_faktor)
    produkt = multipliziereBrueche(erster_faktor, zweiter_faktor)
    text_produkt = schreibeBruch(produkt)
    ergebnis = text_erster_faktor + "<mo>&middot;</mo>" +
       text_zweiter_faktor + "<mo>=</mo>" + text_produkt
    return ergebnis
```

Überprüfung der Darstellung

Ergänze in der Funktion `mathehelfer.schreibeMathML()` die Zeilen für das Multiplizieren:

Führe die Datei `htmlErzeugung.py` aus.

```
erster_faktor = Bruch(3,4)
zweiter_faktor = Bruch(2,3)
inhalt = inhalt + "\n\t\t<p><math>"
inhalt = inhalt + schreibeMultiplikation(erster_faktor, zweiter_faktor)
inhalt = inhalt + "</math></p>"
```

Prüfe im Firefox-Browser, wie die Datei `index.html` aussieht. Dort sollte eine Zeile stehen, die so aussieht:

$$\frac{3}{4} \cdot \frac{2}{3} = \frac{1}{2}$$