Einführung in das Programmieren

Übungsblatt 1

Christoph Draxler, Eva Thoma, Vanessa Reichel

In der letzten Vorlesung hast du bereits einen ersten Einblick in *Visual Studio Code* bekommen. In dieser Programmierumgebung werden wir die Übungsaufgaben lösen. Es empfiehlt sich, für jedes Übungsblatt ein neues File zu erstellen, auf der alle Übungsaufgaben eines Blattes gelöst werden. Öffne deshalb ein neues File (oben links unter 'File' → 'New File'). Gib dem Dokument den Namen *Uebung1.py*, um dieses später leicht wiederzufinden.

1 Grundlegende Datentypen und Variablen

Wir arbeiten in Python mit verschiedenen Datentypen:

- Zahlen (numerische Werte: `int` (integer) und `float`
- Zeichen (Zeichenketten: `str` (string)
- Wahrheitswerte: `bool` (boolean)

Der Datentyp einer Variable kann mit bestimmten Befehlen geändert werden: `str()` konvertiert den Datentyp einer Variablen in einen string. `float()` und `int()` ändern den Datentyp zu einer Gleitkommazahl bzw. einer numerischen Zahl.

Eine **Variable** ist ein **benannter Platzhalter** für Werte. Mit dem Operator `=` wird einer Variable ein Wert zugewiesen.

1.1 Bestimmen des Datentyps

Mithilfe des Befehls `type()` lässt sich der Datentyp eines Elements bestimmen. Überlege Dir, welchen Datentyp die Elemente in der Tabelle haben. Überprüfe deine Vermutung anschließend mit dem `type()` Befehl.
Zur Auswahl stehen: *int, float, str* und *bool*

Element	**Datentyp:** *int, float, str* und *bool*
"Python"	
3	
3.3	
"Python3"	
"Python" == "Python3"	
"Python" + 3	
"Python" + "3"	
7 - 4	
"7 -4"	
7 - 4 == 6	

2 Funktionen

In Funktionen können **Folgen von Befehlen** zusammengefasst werden. Über die **Argumente**, die in den runden Klammern nach dem Funktionsnamen stehen, können Werte in die Funktion eingebracht werden. Es ist wichtig, sich den Aufbau von Funktionen einzuprägen: eine Funktion beginnt immer mit `def` gefolgt von dem **Funktionsnamen**, den **Argumenten** in runden Klammer und einem Doppelpunkt. Nach einem Doppelpunkt muss die nächste Zeile immer nach rechts eingerückt werden. Am Ende der Funktion werden Werte mit `return` zurückgegeben.

2.1 Vorstellen

Mit der Funktion `vorstellen(name, fach)` soll sich ein Nutzer vorstellen können. Wird die Funktion zum Beispiel mit `print(vorstellen('Vanessa', 'Phonetik'))` ausgeführt, soll der Satz `'Hallo, ich heiße Vanessa und studiere Phonetik'` ausgegeben werden. Definiere die Funktion `vorstellen(name, fach)` und überprüfe im Anschluss mit deinem Namen und deinem Studiengang, ob die Ausgabe korrekt ist.

2.2 Verdoppeln

Erstelle eine Funktion `verdoppeln(zahl)`, die die eingegebene Zahl verdoppelt und diese an den Nutzer zurückgibt.

2.3 Fehlersuche

In der folgenden Funktion haben sich 7 Fehler eingeschlichen. Finde die Fehler und bessere sie aus:

```
Def bestellen[anzahl, backware]
ausgabe = Ich hätte bitte gerne + anzahl + " " + Backware
  return(ausgabe)

print(bestellen(2, "Brezen"))
```

3 Was passiert hier?

Beim Programmieren arbeiten wir mit Variablen und Funktionen. Dabei ist es wichtig **sprechende Namen** für Funktionen und Variablen zu wählen. Dies erleichtert das Lesen und Verstehen des Codes. Im folgenden Beispiel wurden keine sprechenden Variablen gewählt:

```
def blabla(hi, ho):
  xyz = ho / 100 * hi
  aaa = str(hi) + " % von " + str(ho) + " sind " + str(xyz)
  return(aaa)
```

- Versuche nachzuvollziehen, was im Code passiert, **ohne den Code auszuführen**.
- Was würde folgender Aufruf zurückgeben?
  `print(blabla(50, 800))`
- Überprüfe deine Vermutung anschließend, indem du die Funktion in Visual Studio Code ausführst.
- Du kannst das besser: überlege dir neue - und vor allem `sprechende` - Namen für die Funktion und deren Variablen.

4 Fallunterscheidungen

Fallunterscheidungen sind Kontrollstrukturen, die den Programmablauf sicherer gestalten. In Python werden Fallunterscheidungen mit `if, elif` und `else` umgesetzt. Nach jedem `if` und `elif` folgt eine **Bedingung** (z.B. das eine Zahl größer als ein bestimmter Wert sein muss). Danach kommt ein Doppelpunkt. Der Code nach dem Doppelpunkt wird nur ausgeführt, wenn die Bedingung **wahr** ist. Erinnere dich an die Wahrheitswerte aus der Vorlesung zu Datentypen: wenn die Bedingung `True` ist wird der folgende Code ausgeführt, bei `False` wird dieser Codeabschnitt übersprungen und stattdessen wird (wenn vorhanden) der Code im `else`-Teil ausgeführt.
Woher weiß Python welcher Code zum `if`-Teil oder `else`-Teil gehört? Das wird über die **Einrückungen** festgelegt: alles was nach dem Doppelpunkt von `if, elif` oder `else` einmal nach rechts eingerückt wurde, gehört zusammen. Erst wenn eine Zeile auf der selben Höhe wie `if, elif` oder `else` beginnt, steht sie außerhalb der Fallunterscheidung.

4.1 Verzweigungen: Strings

Schreibe eine Funktion `harry_beziehungen(person)`, welche für die Personen `Albus Dumbledore, Ron Weasley` und `Lily Potter` die jeweilige Beziehung zu Harry Potter ausgibt.

Hier der jeweilige Funktionsaufruf mit gewünschter Ausgabe:

- `harry_beziehungen("Albus Dumbledore")` gibt `"Mentor"` zurück
- `harry_beziehungen("Ron Weasley")` gibt `"Bester Freund"` zurück
- `harry_beziehungen("Lily Potter")` gibt `"Mutter"` zurück

4.2 Mathespaß

Schreibe eine Funktion `rechnen(x, y, op)` die mit den zwei Zahlen `x` und `y` eine mathematische Operation durchführt und das Ergebnis zurückgibt. Zur Auswahl für `op` stehen: "summe", "multiplikation" und "durchschnitt".

Der Aufruf `rechnen(4, 3, "summe")` gibt 7 zurück, `rechnen(5, 10, "durchschnitt")` gibt `7.5` zurück.

Verwende dafür **nicht** die Funktionen `mean()` und `sum()`!

5 Rekursion

Schreibe eine rekursive Funktion `durch_drei(zahl)`, welche eine Zahl so lange durch drei teilt, bis dies ohne Rest nicht mehr möglich ist.

Der Aufruf der Funktion `durch_drei(351)` gibt folgendes auf dem Bildschirm aus:

```
117.0
39.0
13.0
"Ende!"
```

Folgender Operator wird für die Lösung dieser Aufgabe benötigt: **% (modulo)**

Probiere den Operator zunächst im Terminal aus, um seine Funktionsweise zu verstehen. Schreibe dafür `python3` in das Terminal. Damit wird Python geöffnet. Die folgenden Zeilen beginnen nun mit >>>. Hier kannst Du den Befehl ausprobieren.

Tipp: vor und hinter dem modulo-Zeichen muss jeweils ein numerischer Wert stehen.