Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Marius Weidner, Hannes Saffrich
Simon Dorer, Sebastian Klähn

Universität Freiburg
Institut für Informatik
Wintersemester 2024

Übungsblatt 2

Abgabe: Montag, 28.10.2024, 9:00 Uhr

Vorab beachten Sie bitte folgendes:

- Falls Sie sich noch nicht für ein Tutorat in HisInOne angemeldet haben, tun Sie dies bitte **umgehend** hier. Schreiben Sie aufgrund der verspäteten Anmeldung zudem eine Email an Marius Weidner (weidner@cs.uni-freiburg.de) mit der Übungsgruppe, der Sie beigetreten sind und ihrem RZ-Kürzel.
- Falls Sie diese Woche zum ersten Mal eine Abgabe machen, schreiben Sie ebenfalls eine Mail an Marius Weidner (weidner@cs.uni-freiburg.de), damit Sie einer Tutorin zugewiesen werden.
- Es gelten weiterhin alle Abgaberegeln, wie sie in der ersten Übung beschrieben wurden. Zusätzlich finden Sie diese auch hier.
- Sollten Sie Probleme haben, dann können Sie gerne im Chat oder in den Tutorien fragen. Wir sind auch immer noch ganz lieb und beißen auch nicht :)

Aufgabe 2.1 (Typen; 4 Punkte; Datei: `types.txt` oder `types.md`)
Bestimmen Sie nach jeder der folgenden Wertzuweisungen an die Variable `res` den Typ von `res`. Geben Sie jeweils eine kurze Erläuterung, warum das so ist.

(a)
```
>>> res = (37 - 12) * '13' + '1' * 1337
```

(b)
```
>>> res = float(str(int('2' + '52')) + """2""") + 5
```

(c)
```
>>> res = 12 * int(34 * float("." + "56")) % 78
```

(d)
```
>>> res = (0x1b ^ 0b1110) // 3.1415e3 ** 2.0
```

Hinweis zu Aufgabe 2.2, 2.3 und 2.4: `input`-Funktion
Wie in der Vorlesung gezeigt, können Sie mit der `input`-Funktion Benutzereingaben in Ihr Python-Programm einlesen. Im folgenden wird die Benutzereingabe in einer blauen Schrift dargestellt und die Ausgabe des Programms in schwarzer Schrift.

Ein Beispiel für die Verwendung der Funktion `input` ist das folgende:

```
>>> num = input("Bitte geben Sie hier eine Zahl ein: ")
Bitte geben Sie hier eine Zahl ein: 42
>>> num
'42'
```

Die Funktion `input` gibt immer einen String zurück, auch wenn die Benutzerin eine Zahl eingibt. Sie können einen String in eine Zahl umwandeln, indem Sie die `int` oder `float` Funktion verwenden:

```
>>> int('1337')
1337
>>> float('3.1415')
3.1415
```

Versucht man einen String, der keiner Zahl entspricht, zu konvertieren, wird die Ausführung des Programms durch eine Ausnahme zum Absturz gebracht:

```
>>> int('not a number')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: could not convert string to int: 'not a number'
```

In Ihrem Python-Skript dürfen Sie dies ignorieren. Sie können also annehmen, dass die Benutzerin korrekt formatierte Eingaben macht.

Aufgabe 2.2 (Rahmen drucken; 2 Punkte; Datei: `border.py`)

Erstellen Sie ein Python-Skript `border.py`, das die Benutzerin dazu auffordert, nacheinander die Höhe und Breite eines Rechtecks sowie ein Zeichen für den Rand einzugeben. Sie dürfen davon ausgehen, dass die Benutzerin stets ganze Zahlen größer oder gleich 2 für die Höhe und Breite, sowie ein einzelnes Zeichen für den Rand eingibt.

Im Anschluss soll ein Rechteck mit der angegebenen Höhe und Breite ausgeben werden. Dabei bestehen die Ränder des Rechtecks aus dem eingegebenen Zeichen, während der Innenraum mit Leerzeichen gefüllt ist.

Beispielsweise soll ein Aufruf des Skriptes mit den Eingaben `4`, `12` und `$` *exakt* folgende Ausgabe erzeugen:

```
Höhe: 4
Breite: 12
Zeichen: $
$$$$$$$$$$$$
$          $
$          $
$$$$$$$$$$$$
```

Hinweis: Die Funktion `print` fügt am Ende der Ausgabe standardmäßig einen Zeilenumbruch ein. Um dies zu verhindern, können Sie `print(..., end='')` verwenden. Zum Beispiel [1]:

```
>>> print("Hallo", end=''); print("Welt!")  # Siehe erstes print
HalloWelt!
>>> print("Hallo"); print("Welt!")
Hallo
Welt!
```

Aufgabe 2.3 (PQ-Formel; 2 Punkte; Datei: `pq.py`)

In dieser Aufgabe schreiben Sie ein Python-Skript, das die PQ-Formel für zwei beliebige Gleitkommazahlen p und q berechnet:

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$$

Ihr Skript soll dabei die Benutzerin zunächst auffordern die Werte für p und q einzugeben. Anschließend soll es die beiden möglichen Lösungen x_1 und x_2 der PQ-Formel berechnen und ausgeben.

Sie dürfen dabei annehmen, dass die Benutzerin stets Werte für p und q eingibt, die für positive Werte unter der Wurzel sorgen. Wenn es eine eindeutige Lösung gibt, soll diese trotzdem doppelt ausgegeben werden.

[1] Die ‘;‘ Schreibweise zum Trennen von Anweisungen in einer Zeile dient hier nur zur Erklärung und ist in der Praxis kein guter Stil. Verwenden Sie dies also *nicht* in Ihren eigenen Programmen.

Verwenden Sie in Ihrem Programm Variablen, um mehrfach vorkommende Berechnungen zwischenzuspeichern, statt diese neu zu berechnen.

Ein Aufruf des Skripts mit den Eingaben `-4.0` und `-5.0` soll *exakt* diese Ausgabe liefern:

```
p = -4.0
q = -5.0
x1 = 5.0
x2 = -1.0
```

Aufgabe 2.4 (Ausdrücke optimieren; 2 Punkte; Datei: `optimize.py`)

Betrachten Sie das folgende Python-Programm:

```
x = input()
n = float(x)
a = n * 2
b = a + 3
c = b ** 2
d = c - n
e = d / 4
print(e)
```

Schreiben Sie ein Python-Skript `optimize.py`, das *exakt* das gleiche Verhalten wie das obige Programm hat, aber so wenige Variablenzuweisungen wie möglich verwendet!

Aufgabe 2.5 (Erfahrungen; Datei: `NOTES.md`)

Notieren Sie Ihre Erfahrungen mit diesem Übungsblatt (benötigter Zeitaufwand, Probleme, Bezug zur Vorlesung, Interessantes, etc.).

Editieren Sie hierzu die Datei `NOTES.md` im Abgabeordner dieses Übungsblattes auf unserer Webplattform. Halten Sie sich an das dort vorgegebene Format, da wir den Zeitbedarf mit einem Python-Skript automatisch statistisch auswerten. Die Zeitangabe `4.5 h` steht dabei für 4 Stunden 30 Minuten.

Der Build-Server überprüft ebenfalls, ob Sie das Format korrekt angegeben haben. Prüfen Sie, ob der Build-Server mit Ihrer Abgabe zufrieden ist, so wie es im Video zur Lehrplattform gezeigt wurde.