Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Marius Weidner, Hannes Saffrich
Lukas Kleinert, Timpe Hörig

Universität Freiburg
Institut für Informatik
Wintersemester 2023

Übungsblatt 4

Abgabe: Montag, 13.11.2023, 9:00 Uhr morgens

Wichtig

Aufgabenteile werden mit **0 Punkten** bewertet, wenn:

- Dateien und Funktionen nicht so benannt sind, wie im Aufgabentext gefordert;
- Dateien falsche Formate haben, z.B. PDF statt plaintext;
- Pythonskripte wegen eines Syntaxfehlers nicht ausführbar sind.
- Verwenden Sie nur Befehle und Programmiertechniken, die Inhalt der bisherigen Vorlesungen (bis zum Abgabetermin) und Übungsblättern waren. (Ausnahme: f-Strings)

Ein Syntaxfehler tritt zum Beispiel auf, wenn ein Doppelpunkt vergessen wurde:

```
# Die nächste Zeile sollte mit einem ':' enden.
def add_one(x: int) -> int
    return x + 1
```

Beim Versuch das Programm auszuführen wird Python direkt mitteilen, ob solch ein Fehler vorliegt:

```
$ python3.12 example.py
  File "example.py", line 2
    def add_one(x: int) -> int
                           ^
SyntaxError: expected ':'
```

Syntaxfehler sind also einfach zu entdecken und zu reparieren. Sollten Sie einen Syntaxfehler trotz längerem Anstarren Ihres Codes nicht repariert bekommen, fragen Sie bitte rechtzeitig in den Tutoraten oder im Chat.

Hinweis

Verwenden Sie Typannotationen für Ihre Funktionen, fehlende Typannotationen führen zu Punktabzug.

Aufgabe 4.1 (Funktionen über Listen; 3 Punkte; Datei: `lists.py`)
Implementieren Sie folgende Funktionen:

(a) (1 Punkte) Schreiben Sie eine Funktion `even`, die eine Liste von ganzen Zahlen nimmt und für jede Zahl überprüft, ob diese gerade oder ungerade ist. Dafür soll `even` eine Liste von Tupeln zurückgeben, wobei das Tupel aus einer Zahl und einem Wahrheitswert besteht. Letzterer beschreibt, ob die Zahl gerade ist oder nicht.

Ein Aufruf von `even` mit `[1, 3, 2, -6]` soll Folgendes zurückgeben:

```
>>> even([1, 3, 2, -6])
[(1, False), (3, False), (2, True), (-6, True)]
```

(b) (1 Punkte) Schreiben Sie eine Funktion `min`[1], die eine Liste von ganzen Zahlen nimmt und davon die kleinste Zahl zurückgibt. Ist die Liste leer, soll `None` zurückgegeben werden.

Hinweis: In der Typannotation kann dies mit `Optional[int]` als Rückgabetyp angegeben werden. `Optional` muss aus dem Module `typing` importiert werden. `Optional` bedeutet, dass der Typ entweder der angegebene Typ oder `None` ist.

Sie können `min` mit folgenden Asserts testen[2]:

```
assert min([1, 2, 3]) == 1
assert min([-1, 2, -3]) == -3
assert min([]) is None
```

(c) (1 Punkte) Schreiben Sie eine Funktion `max`, die eine Liste von ganzen Zahlen nimmt und die davon größte Zahl zurückgibt. Ist die Liste leer, soll `None` zurückgegeben werden.

Verwenden Sie hierzu Ihre selbst definierte `min` Funktion.

Hinweis:

In Python kann man wie folgt prüfen, ob etwas `None` ist:

```
>>> None is None
True
>>> 1 is None
False
>>> 0 is None
False
```

Hinweis: Überlegen Sie, wie Sie die Elemente in einer Liste so verändern können, damit Sie `min` zum Lösung der Aufgabe benutzen können.

Sie können `max` mit folgenden Asserts testen:

[1] Sie dürfen hierzu NICHT die `min` Funktion aus der Standardlibrary benutzen.
[2] Bei Vergleichen mit `None` verwendet man `is`, nicht `==`.

```
assert max([1, 2, 3]) == 3
assert max([-1, -2, -3]) == -1
assert max([]) is None
```

Aufgabe 4.2 (Euler'sche Zahl e; 3 Punkte; Datei: `euler.py`)

Die Euler'sche Zahl e beträgt ungefähr 2.718. Diese Zahl ist irrational und kann daher nicht exakt von einem Computer dargestellt werden. Es gilt:

$$e = 1 + \frac{1}{1} + \frac{1}{1*2} + \frac{1}{1*2*3} + ... = \sum_{k=0}^{\infty} \frac{1}{k!},$$

sodass wir die Summe $\sum_{k=0}^{n} \frac{1}{k!}$ als $n$te Approximation der Zahl e bezeichnen können.

Schreiben Sie eine Funktion `approx_e`, die eine nicht negative ganze Zahl n nimmt und die $n$te Approximation von e zurückgibt.

Schreiben Sie dafür die Hilfsfunktion `fac` die eine ganze Zahl n nimmt und die Fakultät dieser Zahl berechnet.

Sie können Ihre Funktion `approx_e` mit den folgenden Asserts testen. Hierfür müssen Sie das Modul `math` importieren.

```
assert approx_e(0) == 1
assert approx_e(1) == 2
assert math.isclose(approx_e(2), 2.5)
assert math.isclose(approx_e(3), 8 / 3)
assert math.isclose(approx_e(100), math.e)
```

Aufgabe 4.3 (Binärzahlen; 4 Punkte; Datei: `binary.py`)

Implementieren Sie folgende Funktionen:

(a) (2 Punkte) Schreiben Sie eine Funktion `to_num`, die einen String von Nullen und Einsen nimmt. Diese Abfolge repräsentiert eine Zahl in Binärdarstellung. Die Funktion `to_num` soll diese Zahl in einen Integer konvertieren und zurückgeben.

Wichtig: Das erste Bit soll das Vorzeichen repräsentieren. `"1"` steht für negativ und `"0"` für positiv. Der String `"11"` steht also für `-1` und NICHT `3`. Hingegen steht der String `"01"` für `1`. Besteht der String aus weniger als zwei Zeichen, so soll `0` zurückgeben werden.

Der Wert lässt sich wie folgt berechnen:

$$1110 = -1 * (1 * 2^2 + 1 * 2^1 + 0 * 2^0) = -6_{10}$$
$$0111 = 1 * (1 * 2^2 + 1 * 2^1 + 1 * 2^0) = 7_{10}$$

Sie können `to_num` mit folgenden Asserts testen:

```
assert to_num("01") == 1
assert to_num("11") == -1
assert to_num("111") == -3
assert to_num("0101") == 5
assert to_num("10") == -0
assert to_num("00") == 0
assert to_num("") == 0
```

(b) (2 Punkte) Schreiben Sie eine Funktion `stream_to_nums`. Diese Funktion soll zwei Argumente nehmen: Einen Stream `stream` als String und einen Separator[3] `sep` ebenfalls als String. Der Stream stellt beliebig viele Binärzahlen getrennt durch den Separator dar. Die Binärzahlen `0111`, `11` und `011010` mit dem Separator `"#"` resultieren in folgendem Stream `"0111#11#011010"`. Die Funktion `stream_to_nums` soll aus einem solchen Stream die Binärzahlen auslesen, mithilfe Ihrer eigenen Funktion aus Aufgabenteil (a) (`to_num`) zu Integern umwandeln und die Liste dieser Zahlen zurückgeben.

Sie können `stream_to_nums` mit folgenden Asserts testen:

```
assert stream_to_nums("01", "#") == [1]
assert stream_to_nums("01#11", "#") == [1, -1]
assert stream_to_nums("01#11#000", "#") == [1, -1, 0]
assert stream_to_nums("01#11#000#", "#") == [1, -1, 0, 0]
assert stream_to_nums("01#11#000#1111", "#") == [1, -1, 0, -7]
```

Aufgabe 4.4 (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 `3.5 h` steht für 3 Stunden 30 Minuten.

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

[3] Wenn Sie einen Separator wählen, der nur aus Nullen und Einsen besteht entsteht ein Problem. Wissen Sie welches? In der Rechnernetze VL stoßen Sie ebenfalls auf dieses Problem und werden dies dort lösen. Hier dürfen Sie das Problem ignorieren.