Informatik I: Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Hannes Saffrich
Wintersemester 2020

Universität Freiburg
Institut für Informatik

Übungsblatt 4

Abgabe: Montag, 30.11.2020, 9:00 Uhr morgens, über git[1]

Wichtig

Ab diesem Übungsblatt werden Aufgabenteile 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; oder
- Pythonskripte wegen eines Syntaxfehlers nicht ausführbar sind.

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

```
def add_one(x)     # Diese Zeile sollte mit einem `:` enden.
    return x + 1
```

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

```
$ python3 example.py
File "example.py", line 1
def add_one(x)
^
SyntaxError: invalid syntax
```

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 die Tutoren um Hilfe.

Hinweis

In den folgenden Aufgaben müssen Sie Stück für Stück eine Liste aufbauen. Hierzu müssen Sie zu einer bestehenden Liste neue Elemente hinzufügen. Dies geht z.B. durch Listenkonkatenation mit einer einelementigen Liste:

```
>>> xs = [1, 2, 3]
>>> xs = xs + [4]
>>> xs
[1, 2, 3, 4]
```

Hinweis

Versuchen Sie bei Ihren Funktionsdefinitionen Typannotationen zu verwenden. Ab nächstem Blatt werden diese verpflichtend.

[1] `https://inpro.informatik.uni-freiburg.de/`

Aufgabe 4.1 (Kurzaufgaben; Datei: `shorts.py`; 6 Punkte)

(a) Schreiben Sie eine Funktion `average`, die eine Liste von Gleitkommazahlen als Argument nimmt und den Durchschnitt der Zahlen zurückgibt.

```
>>> average([])
0.0
>>> average([1.0])
1.0
>>> average([5.0, 10.0, 20.0])
11.666666666666666
```

(b) Schreiben Sie eine Funktion `reverse`, die eine Liste als Argument nimmt und eine Liste mit den gleichen Elementen in umgedrehter Reihenfolge zurückgibt.[2]

```
>>> reverse([])
[]
>>> reverse([1, 2, 3])
[3, 2, 1]
>>> reverse([1, 2, 3, 4, 5])
[5, 4, 3, 2, 1]
```

(c) Schreiben Sie eine Funktion `only_positive`, die eine Liste von ganzen Zahlen als Argument nimmt und eine Liste der Zahlen zurückgibt, die positiv sind.

```
>>> only_positive([])
[]
>>> only_positive([1, 2, 3])
[1, 2, 3]
>>> only_positive([-8, 1, -5, -9, 2, -7, 3, -6])
[1, 2, 3]
```

[2] Verwenden sie dabei *nicht* die Funktion `reversed` die Python bereits zu Verfügung stellt, sondern schreiben Sie die Funktion selbst.

Aufgabe 4.2 (Faltung; Datei: `convolution.py`; Punkte: 4)

Seien xs und ys zwei Listen von Zahlen der Länge N. Die diskrete Faltung von xs und ys an der Stelle 0 ist definiert durch

$$(xs * ys)_0 := \sum_{n=0}^{N-1} xs[n] \cdot ys[N-1-n].$$

Implementieren Sie eine Funktion `convolute_0`, die zwei Listen von ganzen Zahlen `xs` und `ys` als Argumente nimmt und die diskrete Faltung von `xs` und `ys` an der Stelle 0 berechnet.

Schreiben Sie zusätzlich folgende `assert`-Befehle in ihren Code, um zu überprüfen, ob ihre Funktion die richtigen Ergebnisse berechnet:

```
assert convolute_0([1],       [10]         ) == 1*10
assert convolute_0([1,2],     [10,20]      ) == 1*20 + 2*10
assert convolute_0([1,2,3],   [10,20,30]   ) == 1*30 + 2*20 + 3*10
assert convolute_0([1,2,3,4], [10,20,30,40]) == 1*40 + 2*30 + 3*20 + 4*10
```

Aufgabe 4.3 (Primzahlen; Datei: `primes.py`; Punkte: 8)

Primzahlen sind natürliche Zahlen, die durch genau zwei Zahlen teilbar sind: durch 1 und durch sich selbst. Insbesondere ist 2 die kleinste Primzahl, eine größte Primzahl existiert nicht. Implementieren Sie eine Funktion `primes`, die eine ganze Zahl n als Argument nimmt, sukzessiv alle Primzahlen kleiner oder gleich n berechnet und diese in aufsteigender Reihenfolge als Liste zurückgibt. Implementieren Sie dazu die folgende Idee: Um zu überprüfen, ob eine Zahl n prim ist, reicht es, diese auf Teilbarkeit durch alle zuvor erzeugten Primzahlen $\leq n$ zu überprüfen. Bereits erzeugte Primzahlen können (und sollten) in einer Liste zwischengespeichert werden. Sie können Ihre Funktion z.B. wie folgt testen:

```
assert primes(1)  == []
assert primes(3)  == [2, 3]
assert primes(20) == [2, 3, 5, 7, 11, 13, 17, 19]
```

Sie können dabei wie folgt vorgehen:

(a) Implementieren Sie zunächst eine Funktion `is_prime`, die eine ganze Zahl `x` und eine Liste von ganzen Zahlen `ps` als Argumente nimmt und zurückgibt ob `x` eine Primzahl ist. Dabei wird angenommen, dass `ps` die Liste aller Primzahlen kleiner `x` ist.

(b) Implementieren Sie dann die Funktion `primes` unter Verwendung von `is_prime`.

Aufgabe 4.4 (Erfahrungen; Datei: `erfahrungen.txt`; 2 Punkte)

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

Schreiben Sie den groben Zeitaufwand, den Sie für das *gesamte* Blatt benötigt haben, bitte wie folgt in die erste Zeile:

```
Zeitaufwand: 3.5h

[... Freitext, wie bisher ...]
```

Wenn sich genug Leute daran halten, dann können wir ein Pythonskript schreiben, welches automatisiert die durchschnittliche Bearbeitungszeit berechnet ;-)