Informatik I: Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Hannes Saffrich, Simon Ging
Wintersemester 2021

Universität Freiburg
Institut für Informatik

Übungsblatt 4

Abgabe: Montag, 15.11.2021, 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; oder
- Bei Gruppenarbeiten die RZ-Kürzel der Mitglieder fehlen.

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

```
# Die nächste Zeile sollte mit einem ':' enden.
def add_one(x)
    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 2
    def add_one(x)
                  ^
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 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.

Hinweis

Verwenden Sie den folgenden Aufbau für alle Dateien in diesem Blatt:

- Alle imports stehen ganz am Anfang der Datei.
- Danach definieren Sie die Funktionen.
- Dann verwenden Sie `if __name__ == "__main__":`, um den danach folgenden Code nur auszuführen, wenn das Skript direkt gestartet wird. Die Erklärung hierzu befindet sich im vorherigen Übungsblatt.
- Innerhalb dieses ifs schreiben Sie nun allen restlichen Code (z.B. die assert-Befehle).

Tipp: Verwenden Sie während der Entwicklung print-statements, um die Ausgabe ihrer Funktion zu testen. Dies gibt Ihnen ggf. mehr Hilfestellung als ein fehlgeschlagener assert. Bitte löschen Sie die print-statements wieder, bevor Sie ihre Lösung einreichen.

Eine korrekte Datei sieht damit beispielsweise so aus:

```
from math import isclose


def duplicate_input(x: float) -> float:
    return x * 2


if __name__ == "__main__":
    eps = 1e-4
    assert isclose(duplicate_input(2.4), 4.8, rel_tol=eps)
    assert isclose(duplicate_input(-1.7), -3.4, rel_tol=eps)
```

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

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

(b) Testen Sie die Funktion wie folgt:

```
from math import isclose

eps = 1e-4
assert isclose(average([]), 0.0, abs_tol=eps, rel_tol=eps)
assert isclose(average([1.0]), 1.0, rel_tol=eps)
assert isclose(average([5.0, 10.0, 15.0, 20.0]), 12.5, rel_tol=eps)
```

(c) Schreiben Sie eine Funktion `reverse`, die eine Liste als Argument nimmt und eine Liste mit den gleichen Elementen in umgedrehter Reihenfolge zurückgibt.[1]

(d) Testen Sie die Funktion wie folgt:

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

(e) Schreiben Sie eine Funktion `only_positive`, die eine Liste von ganzen Zahlen als Argument nimmt und eine Liste der Zahlen zurückgibt, die positiv sind. Positiv bedeutet, dass die Zahl größer als 0 ist.

(f) Testen Sie die Funktion wie folgt:

```
assert only_positive([]) == []
assert only_positive([1, 2, 3]) == [1, 2, 3]
assert only_positive([-8, 1, -5, -9, 2, -7, 3, -6, 0]) == [1, 2, 3]
```

Aufgabe 4.2 (Pi berechnen; Datei: `calculate_pi.py`; 4 Punkte)

Schreiben Sie eine Funktion `calculate_pi`, welche eine ganze Zahl n als Argument nimmt und die Kreiszahl π anhand der folgenden Formel annähert:

$\pi^2/6 \approx \sum_{i=1}^{n}(1/i^2)$

Beachten Sie, dass die obige Formel alle Zahlen von 1 bis n einschließt, während in python `range(n)` alle Zahlen von 0 bis $n-1$ einschließt. Vergessen Sie nicht, die Formel nach π umzustellen. Die Formel geht auf den Mathematiker Leonard Euler zurück.

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

Testen Sie die Funktion wie folgt:

```
from math import isclose

eps=1e-4
assert isclose(calculate_pi(-3), 0.0, abs_tol=eps, rel_tol=eps)
assert isclose(calculate_pi(1), 2.44948, rel_tol=eps)
assert isclose(calculate_pi(7), 3.01177, rel_tol=eps)
assert isclose(calculate_pi(1000), 3.14063, rel_tol=eps)
assert isclose(calculate_pi(10000), 3.14149, rel_tol=eps)
```

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 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`.

Testen Sie die Funktion wie folgt:

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

Aufgabe 4.4 (Erfahrungen; 2 Punkte; 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 Webplatform. 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 dabei für 3 Stunden 30 Minuten.