Einführung in die Programmierung

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

Universität Freiburg
Institut für Informatik
Wintersemester 2024

Übungsblatt 4

Abgabe: Montag, 11.11.2024, 9:00 Uhr

Abgaberegeln

Zusätzlich zu den bisherigen Abgaberegeln (siehe hier) gelten ab diesem Übungsblatt folgende Regeln:

(a) **Syntax-Fehler**

Ab diesem Übungsblatt werden Python-Skripte, die aufgrund eines Syntax-Fehlers nicht ausführbar sind, mit 0 Punkten bewertet. Syntax-Fehler treten auf, wenn Ihr Programm nicht den Regeln der Python-Syntax entspricht. Ein Beispiel für einen Syntax-Fehler ist das Vergessen eines Doppelpunkts am Ende einer Funktionsdefinition:

```python
# Am Ende der Zeile fehlt ein ':'
def add_one(x: int) -> int
    return x + 1
```

Wenn Sie versuchen, dieses Programm auszuführen, erhalten Sie die folgende Fehlermeldung:

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

Dabei zeigt die Fehlermeldung sowohl die fehlerhafte Zeile als auch eine kurze Fehlerbeschreibung an. In diesem Fall wird in Zeile 2 ein Doppelpunkt erwartet.

Syntax-Fehler sind oft leicht zu finden und zu beheben. Wenn Sie einen Syntax-Fehler nach längerem Überprüfen Ihres Codes dennoch nicht lösen können, wenden Sie sich bitte rechtzeitig an Ihre Tutorin oder fragen Sie im Chat.

(b) **Typannotationen**

Zusätzlich sollen Sie ab diesem Übungsblatt Typannotationen in Ihren Funktionen verwenden. Geben sie dabei den Typen so präzise wie möglich an. Gibt eine Funktion zum Beispiel eine Liste von ganzen Zahlen zurück, so sollte der Rückgabetyp `list[int]` sein.

Eine vollständige Liste aller gültigen Abgaberegeln finden Sie hier auf unserer Website.

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

(a) (1 Punkt) Schreiben Sie eine Funktion `count_occurrences`, die eine Liste von Strings `xs` und einen String `x` nimmt und die Anzahl der Vorkommnisse von `x` in `xs` zählt.

Zum Beispiel:

```
>>> assert count_occurrences([], "jemand da?") == 0
>>> assert count_occurrences(
...     ["Python", "C++", "Java", "Python"], "Python") == 2
>>> assert count_occurrences(
...     ["Python", "C++", "Java", "Python"], "JavaScript") == 0
```

(b) (1 Punkte) Schreiben Sie eine Funktion `min`[1], die eine Liste von ganzen Zahlen nimmt und die kleinste Zahl daraus 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.

Zum Beispiel:

```
>>> assert min([]) is None
>>> assert min([1, 2, 3]) == 1
>>> assert min([-2, 20, -4, 19]) == -4
```

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

Verwenden Sie hierzu Ihre selbst definierte `min` Funktion. Überlegen Sie hierzu, wie Sie die Elemente in der Eingabeliste so verändern können, dass sie `min` zum Lösung der Aufgabe benutzen können.

Zum Beispiel:

```
>>> assert max([]) is None
>>> assert max([1, 2, 3]) == 3
>>> assert max([-2, 20, -4, 19]) == 20
```

Hinweis:

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

[1] Sie dürfen hierzu NICHT die `min` Funktion aus der Standardlibrary benutzen.

```
>>> None is None
True
>>> "None" is None
False
>>> 0 is None
False
```

Aufgabe 4.2 (Mathematische Konstante π; 3 Punkte; Datei: `approx_pi.py`)
Die Zahl π (Pi) beträgt ungefähr 3.14159. Diese Zahl ist irrational und kann daher nicht exakt von einem Computer dargestellt werden. Es gilt:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1},$$

sodass wir die Summe[2]

$$S_n = 4 \sum_{k=0}^{n} \frac{(-1)^k}{2k+1}$$

als $n$te Approximation von π bezeichnen können.

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

Zum Beispiel:

```
>>> assert math.isclose(approx_pi(0), 4)
>>> assert math.isclose(approx_pi(1), 2.666666666666667)
>>> assert math.isclose(approx_pi(2), 3.466666666666667)
>>> assert math.isclose(approx_pi(78), 3.1542503744801236)
>>> assert math.isclose(approx_pi(1000), 3.1425916543395442)
>>> assert math.isclose(approx_pi(10000), 3.1416926435905346)
```

Aufgabe 4.3 (Zahlen und Farben; 4 Punkte; Datei: `colors.py`)
Implementieren Sie folgende Funktionen:

(a) (2 Punkte) Hexadezimalzahlen sind Zahlen im Zahlensystem zur Basis 16. Dabei entsprechen die Ziffern von 0 bis 9 den gleichen Werten wie in Dezimalzahlsystem. Die Werte von 10 bis 15 werden hingegen durch die Buchstaben A bis F dargestellt. Um eine Hexadezimalzahl $h = h_n h_{n-1} \ldots h_1 h_0$ in eine Dezimalzahl d zu konvertieren, wird die folgende Formel verwendet:

$$d = \sum_{i=0}^{n} h_i \cdot 16^i$$

[2] https://de.wikipedia.org/wiki/Summe#Notation_mit_dem_Summenzeichen

Beispielsweise entspricht die Hexadezimalzahl 3AF2 der Dezimalzahl $3 \cdot 16^3 + 10 \cdot 16^2 + 15 \cdot 16^1 + 2 \cdot 16^0 = 15090$.

Schreiben Sie eine Funktion `hex_to_dec`, die eine Hexadezimalzahl als Zeichenkette (`str`) entgegennimmt und diese in eine Dezimalzahl (`int`) umwandelt und zurückgibt. Sie können dabei annehmen, dass die Eingabe aus mindestens einem Zeichen besteht und nur die Zeichen `0-9` und `A-F` vorkommen.

Verwenden Sie bei Ihrer Implementierung **nicht** die `int`-Funktion aus der Standardlibrary!

Zum Beispiel:

```
>>> assert hex_to_dec("0") == 0
>>> assert hex_to_dec("17") == 23
>>> assert hex_to_dec("FF") == 255
>>> assert hex_to_dec("00FF") == 255
>>> assert hex_to_dec("3AF2") == 15090
```

Hinweis: Sie *können* die Funktion `ord`[3]-Funktion verwenden, um die Zeichen A-F in Dezimalzahlen umzuwandeln, ohne eine Fallunterscheidung zu verwenden:

```
>>> ord("A")  # ASCII Wert von A
65
>>> ord("C")  # ASCII Wert von C
67
>>> ord("C") - ord("A") + 10  # Differenz plus 10
12
```

(b) (2 Punkte) Farben werden häufig im dreidimensionalen Rot-Grün-Blau (RGB)-Farbraum dargestellt. Dieser Farbraum basiert auf einem 3-Tupel, das jeweils die Intensität der Farben Rot, Grün und Blau im Bereich von 0 bis 255 beschreibt. Statt das 3-Tupel direkt zu verwenden, wird es oft als sechsstellige Hexadezimalzahl dargestellt. Die ersten beiden Ziffern repräsentieren die Intensität von Rot, die nächsten beiden die Intensität von Grün und die letzten beiden die Intensität von Blau.

Zum Beispiel:

$$ACB5F2 \leftrightarrow (\underbrace{172}_{=AC_{16}}, \underbrace{181}_{=B5_{16}}, \underbrace{242}_{=F2_{16}})$$

Schreiben Sie eine Funktion `parse_color`, die eine RGB-Hexadezimalzahl entgegennimmt und ein RGB-Tupel in Dezimaldarstellung zurückgibt. Verwenden Sie zur Umwandlung der Hexadezimalzahlen in Dezimalzahlen die Funktion `hex_to_dec` aus Aufgabenteil (a).

[3] https://docs.python.org/3/library/functions.html#ord

Zum Beispiel:

```
>>> assert parse_color("000000") == (0, 0, 0)
>>> assert parse_color("FFFFFF") == (255, 255, 255)
>>> assert parse_color("FA7BCC") == (250, 123, 204)
>>> assert parse_color("123456") == (18, 52, 86)
```

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 `5.5 h` steht für 5 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.