Informatik I: Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Hannes Saffrich, Michael Uhl
Wintersemester 2022

Universität Freiburg
Institut für Informatik

Übungsblatt 10

Abgabe: Montag, 9.01.2023, 9:00 Uhr morgens

Hinweis

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

- Dateien und Definitionen nicht so benannt sind, wie im Aufgabentext gefordert;
- Dateien falsche Formate haben, z.B. PDF statt plaintext;
- Pythonskripte wegen eines Syntaxfehlers nicht ausführbar sind.

Es gibt **Punktabzug** wenn:

- Funktionen keine oder falsche Typannotationen aufweisen. Ausnahme: Bei Funktionen, die stets `None` zurückgeben, kann der Rückgabetyp weggelassen werden.

Gruppenaufgaben müssen von allen Mitgliedern abgegeben werden und in der ersten Zeile müssen die Mitglieder in einem Kommentar vermerkt werden, z.B:

```
# Gruppe: xy123, yz56, zx934
```

Aufgabe 10.1 (Klausur; Datei: `exam.py`; Punkte: 10 = 3+4+3)

Die Studierenden des Kurses *Einführung in die Programmierung* haben eine Klausur geschrieben. Um die Auswertung der Noten zu automatisieren, werden die Punkte jedes einzelnen Studierenden in einem **dictionary** als ganze Zahl gespeichert. Dieses Dictionary speichert die Namen der Studierenden in den Schlüsseln (keys) und die erreichten Punkte in den zugehörigen Werten (values). Beispiel:

```
>>> student_points = {"Adam": 63, "John": 112, "Donald": 43}
```

(a) Bei der Erstkorrektur sind den Tutoren ein paar Fehler unterlaufen. So kann es sein, dass manche Studierenden noch Zusatzpunkte bekommen, manche jedoch auch Punkte verlieren. Ein übermotivierter Assistent der Vorlesung legt aus diesem Grund ein zweites Dictionary an, das die Änderungen (sowohl negative wie auch positive) speichert. Schreiben Sie eine Funktion `update_points` die beide Dictionary als Eingabe bekommt und die neue Gesamtpunktzahl berechnet. Beispiel:

```
>>> changes = {"Adam": 3, "John": -7}
>>> update_points(student_points, changes)
>>> student_points
{'Adam': 66, 'John': 105, 'Donald': 43}
```

Nehmen Sie an die maximale Punktzahl sei 120. Wenn die neu berechneten Punkte über 120 oder unter 0 liegen sollten, soll ein ValueError mit der Fehlermeldung *"Die Gesamtpunktzahl muss zwischen 0 und 120 liegen"* geworfen werden. Sollte in `changes` ein Name vorkommen, der nicht in den `student_points` vorkommt, soll ein `KeyError` mit der Fehlermeldung *"[Name] wurde nicht gefunden"* geworfen werden. Hierbei ist [Name] der Platzhalter für den Namen des Studierenden.

(b) Schreiben Sie eine Funktion `compute_grade(student_points: dict[str, int], max_points: int, student_name: str) -> int`, welche die Note für einen beliebigen Studierenden berechnet. Es gibt die Noten 1, 2, 3, 4 und 5. Diese werden wie folgt aus der Maximalpunktzahl der Klausur (`max_points`), der gesondert zu berechnenden Mindestpunktzahl zum Bestehen (`pass_points`) und den vom Studierenden erreichten Punkte (`student_points`) berechnet:

- Der Studierende hat nicht bestanden (Note 5), wenn weniger Punkte erreicht wurden als die Mindestpunktzahl (`pass_points`) angibt.
- Die anderen 4 Noten sind gleich unter den restlichen Punkten verteilt, sprich, weniger als 25% der Punkte entsprechen einer 4. Mehr, aber weniger als 50% einer 3 usw... Beispiel: Bei 120 maximalen Punkten und einer Mindestpunktzahl von 60, bekommt man für 60 - 74 Punkte eine 4, für 75 - 89 Punkte eine 3, für 90 - 104 Punkte eine 2 und für mehr als 104 Punkte eine 1.
- Die Mindestpunktzahl wird wie folgt berechnet:
  `pass_points = max_points // 2`

- Zur Unterstützung seines Wahlkampfs fordert Ex-Präsident T., dass die Durchfallquote aller Klausuren bei höchstens 40% liegen darf. Sollte das in dieser Klausur nicht der Fall sein, werden die `pass_points` so weit nach unten angepasst, bis die Forderung erfüllt ist.

Als Eingabe soll die Funktion wie gehabt das `student_points` Dictionary bekommen, ebenso wie die `max_points` und den Namen des Studierenden als `String`.

```
>>> student_points = {"Adam": 63, "John": 112, "Donald": 43}
>>> compute_grade(student_points, 120, "Adam")
4
```

Hinweis: Um den Code übersichtlich zu gestalten ist es hier sinnvoll Hilfsfunktionen zu schreiben. Diese können dann auch in der nächsten Teilaufgabe wiederverwendet werden.

(c) Implementieren Sie die Funktion `cluster_by_grade`, welche ein Dictionary mit Noten (keys) und Listen von Studierenden (values) zurückgibt. Die Eingabeparameter sollen `student_points` und `max_points` sein. `pass_points` werden hier auch wieder initial auf `max_points // 2` gesetzt und bei einer zu hohen Durchfallquote angepasst.

```
>>> student_points = {"Mira": 80, "Olivia": 95, "Emily": 83}
>>> cluster_by_grade(student_points, 120)
{3: ['Mira', 'Emily'], 2: ['Olivia']}
```

Hinweis: Die Reihenfolge, wie die Noten in der Ausgabe erscheinen, spielt keine Rolle. Noten sollen aber nur dann angezeigt werden, wenn es mindestens einen Studierenden gibt der diese erzielt hat.

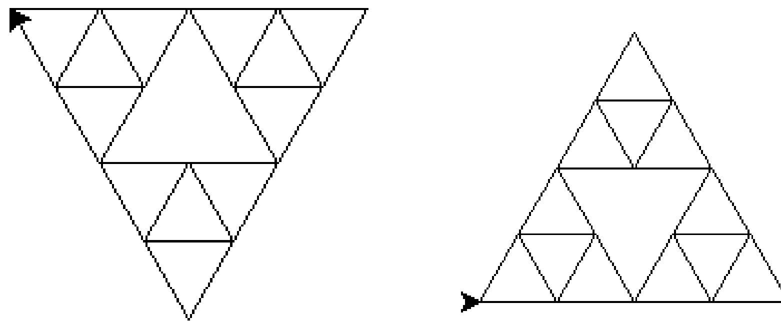
Aufgabe 10.2 (Sierpiński-Dreieck; Datei: `sierpinski.py`; Punkte: 5)

Das *Sierpiński-Dreieck* ist ein Fraktal, welches als $0L$-System wie folgt beschrieben werden kann:

$$V = \{F, G, +, -\}$$
$$\omega = F - G - G$$
$$P = \{F \mapsto F - G + F + G - F,$$
$$G \mapsto GG\}$$

Hierbei entsprechen F und G dem Vorwärtszeichnen einer Strecke, $+$ entspricht einer Drehung um 120 nach links und $-$ entspricht einer Drehung um 120 nach rechts.

Implementieren Sie eine Funktion `sierpinski(size: int, n: int)` zum Zeichnen eines Sierpiński-Dreiecks mit Hilfe des `turtle`-Moduls (siehe Vorlesung), wobei `size` die jeweilige Streckenlänge und `n` die Rekursionstiefe (bzw. Anzahl Generationen) angibt (analog zur Vorlesung). Das Ergebnis für $n = 2$ sollte so, oder so ähnlich aussehen (die Ausrichtung spielt keine Rolle):



Aufgabe 10.3 (**Knobelaufgabe** Polynom raten; Datei: `polynom.py`; Punkte: 3)

In dieser Aufgabe betrachten wir Polynomfunktionen über ganzen Zahlen. Das heißt, Funktionen der Form

$$f(x) = \sum_{i=0}^{n} a_i x^i,$$

wobei $n \geq 0$ ist, a_i eine ganze Zahl für $0 \leq i \leq n$ und $a_n \neq 0$. Auch für x sollen nur ganze Zahlen eingesetzt werden. Die Zahl n bezeichnet den Grad der Polynomfunktion.

Die Aufgabe ist nun wie folgt: Gegeben eine geheime Funktion `f: int → int`, von der nur bekannt ist, dass sie eine Polynomfunktion vom Grad n ist. Finden Sie die Koeffizienten $a_0, a_1, \ldots, a_n$.

Diese Aufgabe gehen wir Schritt für Schritt an:

(a) Ein (ganzzahliges) Polynom vom Grad 1 ist eine lineare Funktion der Form

$$f(x) = a_0 + a_1 x,$$

wobei a_0 und a_1 ganze Zahlen sind. Hier sind zwei Beispiele für solche Funktionen:

```
def f10(x: int) → int:
    return 1 + 2 * x

def f11(x: int) → int:
    return -1 + 3 * x
```

Schreiben Sie eine Funktion `def crack_1(f) → list[int]:`, sodass

```
assert crack_1(f10) == [1, 2]
assert crack_1(f11) == [-1, 3]
```

Hinweis: Die `crack_1` Funktion kann ihre Argumentfunktion `f` nur auf ausgewählte Argumente anwenden. Überlegen Sie, welche Argumente Ihnen weiterhelfen. Was ergibt sich z.B., wenn Sie die Polynomfunktion `f` auf Null anwenden? Betrachten Sie die Differenzen der Funktionswerte.

(b) Betrachten Sie nun ganzzahlige Polynomfunktionen vom Grad 2 wie die folgenden Beispiele:

```
def f20(x: int) → int:
    return 1 + 2*x + x*x

def f21(x: int) → int:
    return -1 - 4*x + 2 * x * x

def f22(x: int) → int:
    return (x + 1) * (x - 1)
```

Schreiben Sie eine Funktion `def crack_2(f) → list[int]:`, sodass

```
assert crack_2(f20) == [1, 2, 1]
assert crack_2(f21) == [-1, -4, 2]
assert crack_2(f22) == [-1, 0, 1]
```

Hinweis: Die `crack_2` Funktion muss ihre Argumentfunktion `f` auf mindestens drei verschiedene, ausgewählte Argumente anwenden.

(c) Erweitern Sie den Ansatz auf Polynomfunktionen vom Grad 3, d.h., implementieren Sie `crack_3 (f) → list[int]`, sodass

```
def f30(x: int) → int:
    return x * (x + 10) * (x - 5)

assert crack_3(f30) == [0, -50, 5, 1]
```

Zur Kontrolle Ihrer Lösungen verwenden Sie die Funktionen in der zur Verfügung gestellten Datei `polynom_test.py` wie folgt:

```
>>> from polynom_test import  test_many_cracker_n
>>> test_many_cracker_n(1, crack_1)
Passed 1000 tests
>>> test_many_cracker_n(2, crack_2)
Passed 1000 tests
>>> test_many_cracker_n(3, crack_3)
Passed 1000 tests
```

Aufgabe 10.4 (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 `4.5 h` steht dabei für 4 Stunden 30 Minuten.

Bei Gruppenaufgaben müssen alle Gruppenmitglieder die Abgabe hochladen. Ferner müssen Sie die RZ-Accounts aller Gruppenmitglieder angeben, sodass wir Sie bei der Korrektur zuordnen können und unsere Tutoren die Aufgabe nicht mehrfach korrigieren müssen. Fügen Sie hierzu in der `NOTES.md`-Datei unter der Zeile für den Zeitbedarf eine weitere Zeile hinzu die exakt das folgende Format hat:

```
Gruppe: xy123, xy234, xy345
```

Hierbei stehen `xy123`, `xy234`, und `xy345` für die RZ-Accounts der Gruppenmitglieder. 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.