Informatik I: Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Hannes Saffrich, Marc Fuchs
Wintersemester 2020

Universität Freiburg
Institut für Informatik

Übungsblatt 10

Abgabe: Montag, 25.01.2021, 9:00 Uhr morgens, über git[1]

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; oder
- 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
```

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

Aufgabe 10.1 (Rechtecke; Datei: `rectangle.py`; Punkte: 5)

Betrachten Sie die in der Vorlesung vorgestellte Datenklasse Point2D, welche einen zweidimensionalen Vektor repräsentiert.

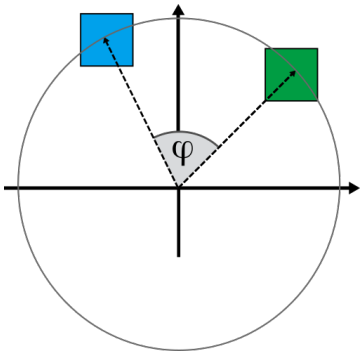
```
from dataclasses import dataclass
from math import sin, cos, pi

@dataclass
class Point2D:
x: float
y: float

def __eq__ (self, other):
    return (isinstance(other , Point2D) and
            self.x ==  other.x and self.y ==  other.y)

def __add__(self, other: 'Point2D') -> 'Point2D':
    return Point2D(self.x + other.x, self.y + other.y)

def turn(self, phi: float):
    self.x, self.y = (self.x * cos(phi) - self.y * sin(phi)
                      ,self.x * sin(phi) + self.y * cos(phi))
```

Ihre Aufgabe besteht darin, die Datenklasse Rectangle zu implementieren, welche ein Rechteck im kartesischen Koordinatensystem darstellen soll. Dabei müssen folgende Eigenschaften gelten:

- Das Rechteck soll durch 2 Vektoren definiert werden. Diese Vektoren sollen die Ecken *links-unten* (`bottom_left: Point2D`) und *rechts-oben* (`top_right: Point2D`) repräsentieren (Wir betrachten hier nur Rechtecke, deren Seiten parallel zu den Achsen sind).
- `bottom_left` und `top_right` sollen als *Properties* mit *Getter*-Methoden gemäß Vorlesung definiert werden.
- Jedes Rechteck besitzt eine Invariante: Der `bottom_left` Vektor soll sowohl in der x wie auch der y Koordinate einen kleineren Wert als sein Gegenüber besitzen. Schreiben Sie, wie in der Vorlesung erläutert, die korrekte Invariante in den *Docstring* der Klasse und sorgen Sie dafür, dass der Konstruktor die Einhaltung der Invariante prüft.
- Implementieren Sie eine Methode `area`, welche die Fläche des Rechtecks als float berechnet.
- Definieren Sie den `==` Vergleich für Rechtecke. Es soll True zurückgegeben werden, wenn für zwei Rechtecke alle Eckpunkte die selben Koordinaten haben.
- Definieren Sie den `+` Operator für Rechtecke. Dabei soll für `r1 + r2` ein neues Rechteck mit minimaler Ausdehnung entstehen, sodass es `r1` und `r2` vollständig

enthält.

- Schreiben Sie eine Methode `turn_center`, welche ein Rechteck auf einem Kreis um den Ursprung dreht. Berechnen Sie dafür zuerst die Koordinaten des Mittelpunktes des Rechtecks (also des Punktes der genau zwischen `top_right` und `bottom_left` liegt) und drehen Sie diesen Punkt anschließend um den Winkel `phi` (`float`). Das Resultat ergibt den Mittelpunkt des neuen, verschobenen Rechtecks. Passen Sie nun die Koordinaten von `bottom_left` und `top_right` den neuen Mittelpunkt an.
  Hinweis: Diese Methode soll kein neues Rechteck-Objekt erzeugen, sondern nur die Koordinaten von `bottom_left` und `top_right` verändern. Länge und Breite (und somit auch Fläche) des Rechtecks bleiben dabei unverändert. In folgendem Bild sieht man ein Beispiel. Hier wird das grüne Rechteck mittels `turn_center` auf die Position des blauen Rechtecks verschoben.



Beispiel:

```
>>> rect = Rectangle(Point2D(0, -1), Point2D(2, 1))
>>> rect.bottom_left
Point2D(x=0, y=-1)
>>> rect.area()
4
>>> rect2 = Rectangle(Point2D(0, -1), Point2D(2, 2))
>>> rect2 == rect
False
>>> (rect + rect2).area()
6
>>> rect.turn_center(pi/2)
>>> rect.top_right
Point2D(x=1.0, y=2.0)
```

Änderungen der Vektoren z.B. `rect.top_right = Point2D(3, 1)` sollen zu Fehlern führen.

Aufgabe 10.2 (Klausur; Datei: `exam.py`; Punkte: 8 = 2+4+2)

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** 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 Gesamtpunkzahl 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`, 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 Mindestpunktzahl zum Bestehen (`pass_points`) und den vom Studierenden erreichten Punkten (`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, die unteren 25% der Punkte entsprechen einer 4. Das nächst höhere Viertel einer 3 usw... Beispiel: Bei 120 maximalen Punkten und einer Mindestpunktzahl von 60, bekommt man für 60 - 74 Punkten eine 4, für 75 - 89 Punkten eine 3, für 90 - 104 Punkten eine 2 und für mehr als 104 Punkte eine 1.
- Die Mindespunktzahl wird wie folgt berechnet:
  `pass_points = max_points // 2`

– Zur Unterstützung seines Wahlkampfs fordert Präsident T., dass die Durchfallquote aller Klausuren bei höchstens 30% 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")
3
```

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 inzial 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 min. einen Studierenden gibt der diese erziehlt hat.

Aufgabe 10.3 (Potenzmenge; Datei: `powerset.py`; Punkte: 5)

Die Potenzmenge $\mathcal{P}(X)$ einer Menge X ist eine neue Menge, die aus allen Teilmengen von X besteht.

Definieren Sie eine Funktion `powerset(s: list) -> list`, die für eine beliebige Menge s, rekursiv die Potenzmenge berechnet und zurückgibt. Verwenden Sie dabei Python-Listen zur Darstellung von Mengen. Wir nehmen an, dass keine Wiederholungen in den Eingabelisten auftreten. Beispiel:

```
>>> powerset([1, 2, 3])
[[], [1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3]]
```

Hinweis: Das Importieren von weiteren Modulen ist bei der Bearbeitung dieser Aufgabe nicht erlaubt. Die Reihenfolge der Elemente spielt im Ergebnis keine Rolle.

Aufgabe 10.4 (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 ;-)