Informatik I: Einführung in die Programmierung

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

Universität Freiburg
Institut für Informatik

Übungsblatt 11

Abgabe: Montag, 17.01.2022, 9:00 Uhr morgens

Hinweis: Es gelten die selben Regeln wie bisher, diese können in Blatt 8 eingesehen werden. ,

Aufgabe 11.1 (Geometrische Formen; Datei: `shapes.py`; Punkte: $9 = 3 + 3 + 3$)

Betrachten Sie die vorgegebene Datei `geo.py`.

- `GuiWrapper` ist eine Datenklasse, die ein Fenster mit einer Zeichenfläche öffnet. Dafür wird das Standardmodul `tkinter` verwendet.
- `Vector2D` beschreibt einen 2-dimensionalen Vektor. Beachten sie die Operator-Überladung: ein solcher Vektor kann negiert (`-`), zwei Vektoren können addiert (`+`) oder subtrahiert (`-`) werden.
- `Object2D` beschreibt schließlich ein zu zeichnendes Objekt mit einer Position.

  Die Methode `draw` ist mithilfe des Pakets `abc` als sogenannte "abstrakte Methode" deklariert, die von abgeleiteten Klassen implementiert werden muss. Damit dies zur Laufzeit überprüft wird, wird die Klasse mit `metaclass=abc.ABCMeta` deklariert und die Methode mit `@abc.abstractmethod` dekoriert.

(a) Erstellen Sie die Datenklasse `Circle` nach den folgenden Vorgaben. Importieren Sie die benötigten Datenklassen aus der Datei `geo.py`. Beachten Sie die in der Vorlesung beschriebenen Regeln für Invarianten und Properties. Insbesondere sollen alle Properties read-only sein.

- Superklasse ist `Object2D`.
- Zusätzlich zur Position (Mittelpunkt) aus `Object2D` hat der Kreis einen Radius > 0. Definieren Sie den Radius als `property` und stellen Sie sicher, dass die Invariante nicht verletzt wird.
- Erstellen Sie zusätzlich die beiden Properties `top_left` und `bottom_right`, diese sind vom Typ `Vector2D` und beschreiben die Eckpunkte des umschließenden Quadrats. Diese Properties sollen einmalig aus Radius und Position berechnet werden.
- Implementieren Sie die Methode `draw`, verwenden Sie hierfür `gui.canvas.create_oval`. Diese Methode benötigt 4 Koordinaten für die zwei Eckpunkte sowie die Farbwerte mit den keywords `fill` und `outline`.

Testen Sie die Implementierung wie folgt. Beachten Sie die Reihenfolge der Aufrufe: Zunächst wird das GUI erstellt, dann werden alle Objekte erstellt

und gezeichnet, danach wird das GUI gestartet.

```
if __name__ == "__main__":
    gui = GuiWrapper(width=800, height=600)
    circle = Circle(Vector2D(100, 200), 75)
    circle.draw(gui, fillcolor="lightblue")
    gui.start()
```

(b) Erstellen Sie die Datenklasse `RotatableEllipse` wie folgt:

- Superklasse ist `Object2D`.
- Zusätzlich zur Position (Mittelpunkt) aus `Object2D` hat die Ellipse die folgenden Properties: `size` definiert die Breite und Höhe der Ellipse. Der Typ ist `Vector2D`, beide Koordinaten müssen größer als 0 sein. `angle` definiert den Winkel der Ellipse im Bogenmaß. Die Ellipse wird im Uhrzeigersinn um den Mittelpunkt rotiert. Der Typ ist `float`, der Wert muss innerhalb des Wertebereichs von 0 (eingeschlossen) bis 2 pi (ausgeschlossen) liegen.
- Implementieren Sie die Methode `draw`, verwenden Sie hierfür `gui.canvas.create_polygon`. Erstellen Sie eine Liste mit 360 Punkten `[x1, y1, ..., x360, y360]`, um die Ellipse zu beschreiben. Verwenden Sie den star-operator `*liste` um die Koordinaten der Punkte der Funktion zu übergeben. Dieser Operator wandelt die Liste in einzelne Argumente um. Beispiel: `funktion(*[1, 2, 3])` entspricht `funktion(1, 2, 3)`. Übergeben Sie die Farbwerte mit den keywords `fill` und `outline`.
- Eine Lösungsmöglichkeit ist, die Punkte der Ellipse jeweils um den Nullpunkt zu berechnen, danach um den Nullpunkt zu drehen und dann zu verschieben.

Zeichnen Sie die Ellipse wie folgt:

```
ellipse = RotatableEllipse(Vector2D(300, 250), Vector2D(200, 50),
                           3 * pi / 5)
ellipse.draw(gui, fillcolor="pink")
```

(c) Schlussendlich erstellen Sie noch die Datenklasse `Triangle`:

- Superklasse ist `Object2D`.
- Zusätzlich zur Position (eine Ecke des Dreiecks) aus `Object2D` hat das Dreieck die Properties `edge1` und `edge2` um die absolute Position der beiden anderen Ecken zu beschreiben.
- Zeichnen Sie das Dreieck mit `gui.canvas.create_polygon`. Übergeben Sie die Farbwerte mit den keywords `fill` und `outline`.

Zeichnen Sie das Dreieck wie folgt:

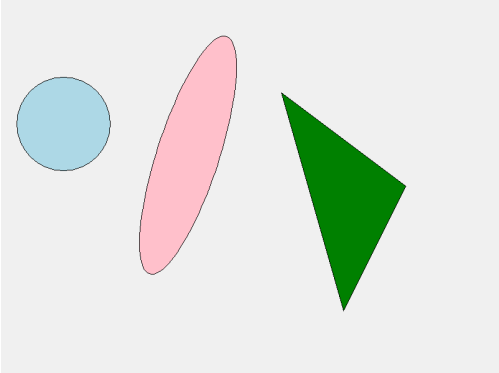

Abbildung 1

```
triangle = Triangle(Vector2D(450, 150), Vector2D(650, 300),
                    Vector2D(550, 500))
triangle.draw(gui, fillcolor="green", outlinecolor="black")
```

Das komplette Bild sehen Sie in Abbildung 1.

Aufgabe 11.2 (Kunstwerk; Datei: `face.py`; Punkte: 3)

Verwenden Sie die Klassen aus der vorherigen Aufgabe, um ein Gesicht zu zeichnen. Beispiel: Abbildung 2. Laden Sie einen Screenshot davon in der Datei `face.png` hoch.

Die besten Bilder werden wir auf unserer Webseite veröffentlichen.

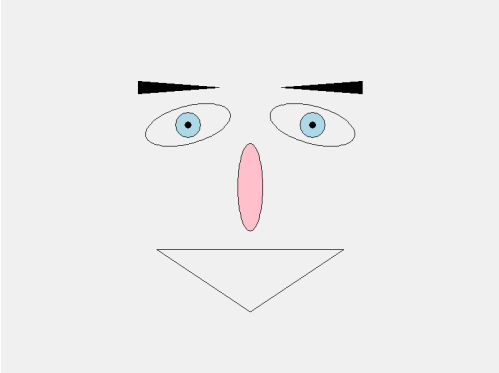

Abbildung 2

Aufgabe 11.3 (Vektoren; Datei: `vector.py`; Punkte: 6)

Hier sollen sie die Datenklasse `Vector` implementieren, welche einen Vektor mit beliebiger Dimension abbilden kann.

- Annotatieren Sie reelle Zahlen mit `float`.
- Vektorinstanzen sollen unveränderbar sein, zum Ändern eines Wertes muss eine neue Instanz erzeugt werden.
- Für Invarianten und Properties gelten die selben Regeln wie in der ersten Aufgabe des Blattes.

(a) Erstellen Sie zunächst die Hilfsfunktion `apply_binary_operator`. Diese Funktion soll einen Operator als string und zwei reelle Zahlen als Argumente nehmen. Je nach Operator (`+`, `-` oder `*`) soll die entsprechende Berechnung ausgeführt und das Ergebnis zurückgegeben werden. Bei fehlerhaftem Operator soll eine Fehlermeldung ausgegeben werden. Verwenden Sie Pattern Matching. Verwenden Sie insbesondere **nicht** `eval`.

```
>>> print(apply_binary_operator("-", 5, 3))
2
```

(b) Erstellen Sie die Datenklasse Vektor mit den folgenden Eigenschaften:

- Property `values`, eine Liste reeller Zahlen, die den Vektor beschreibt.
- Properties `dim` (Dimension des Vektors) und `length` (Euklidische Länge des Vektors). Diese Properties sollen einmalig automatisch berechnet werden. Die Dimension muss größer als 0 sein.
- Überladen der Methode `__str__`: `print(Vector([1, 2, 3]))` soll ausgegeben werden als `3D vector: [1, 2, 3]`.
- Überladen der folgenden unären Operatoren: `__pos__` (`+`) gibt den Vektor selbst zurück. `__neg__` (`-`) wendet Negation auf jedes Element an.
- Methode `operate_binary`: Diese nimmt einen Operator als string und ein weiteres Argument `other` und berechnet das Ergebnis der Operation mit der obigen Hilfsfunktion. `other` kann ein Vektor oder eine reelle Zahl (d.h. `int` oder `float`) sein. Im Falle eines Vektors soll sichergestellt werden, ob beide Vektoren die selbe Dimension haben. Anschließend soll die Berechnung elementweise durchgeführt werden. Im Falle einer reellen Zahl soll die Berechnung zwischen jedem dem Element des Vektors und der Zahl durchgeführt werden. Rückgabe ist in beiden Fällen ein neuer Vektor. Für alle anderen Eingaben soll ein Fehler ausgegeben werden.
- Überladen der folgenden binären Operatoren: `__add__` (`+`), `__sub__` (`-`), und `__mul__` (`*`). Es soll jeweils die Methode `operate_binary` aufgerufen werden.

– Überladen der Operatoren, wenn ein Skalar links und der Vektor rechts steht, in der selben Weise wie zuvor: `__radd__` (+), `__rsub__` (-) und `__rmul__` (*).

Testen Sie Ihre Implementierung mit der Datei `test_vector.py`.

Aufgabe 11.4 (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 `7.5 h` steht dabei für 7 Stunden 30 Minuten.