Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Marius Weidner, Hannes Saffrich
Lukas Kleinert, Timpe Hörig

Universität Freiburg
Institut für Informatik
Wintersemester 2023

Übungsblatt 3

Abgabe: Montag, 06.11.2023, 9:00 Uhr morgens

Bitte beachten Sie weiterhin die Regeln für die Abgabe auf dem vorherigen Übungsblatt, da Verstoße ggf. zu Punktabzug führen.
Werfen Sie auch gelegentlich einen Blick in den Ankündigungschannel des Chats [1], da dort wichtige Hinweise zu Tutoraten, Übungen usw. veröffentlicht werden.

Importieren von eigenen Modulen

In der Vorlesung wurde gezeigt wie Sie Definitionen aus dem `math`-Modul von Python's Standardbibliothek importieren können:

```
from math import sin, pi
```

Sie können aber auch Definitionen aus eigenen Python-Dateien importieren. Angenommen Sie haben zwei Dateien `foo.py` und `bar.py` und möchten die Definitionen aus `foo.py` in `bar.py` verwenden. Sofern die Dateien im gleichen Verzeichnis liegen, können Sie hierzu in der `bar.py` Folgendes schreiben:

```
from foo import some_function
```

Der Modulname `foo` ergibt sich also aus dem Dateiname `foo.py` durch Weglassen der Dateiendung `.py`

Vorsicht beim Importieren

Importiert man eine Datei `foo.py` in eine andere Datei `bar.py`, dann werden in `bar.py` nicht nur die Definitionen von `foo.py` verfügbar gemacht, sondern auch alle Anweisungen aus `foo.py` direkt ausgeführt.

Beispiel: Angenommen die Datei `foo.py` hat den Inhalt

```
def some_function(x):
    print(x)

print("Hi! My name is foo.py!")
```

und die Datei `bar.py` hat den Inhalt

```
from foo import some_function

some_function(42)
```

dann erzeugt das Ausführen von `bar.py` folgende Ausgabe:

[1] https://chat.laurel.informatik.uni-freiburg.de/group/2023WS-EiP

```
Hi! My name is foo.py!
42
```

Um das zu verhindern, können wir die `foo.py` hierzu wie folgt umschreiben:

```
def some_function(x):
    print(x)

if __name__ == "__main__":
    print("Hi! My name is foo.py!")
```

Die Variable `__name__` wird von Python automatisch gesetzt:

- Wird `foo.py` von einer anderen Python-Datei importiert, dann hat die Variable `__name__` den Wert `"foo"` - also den Name des Moduls.
- Wird `foo.py` aber als Programm ausgeführt, z.B. mit `python3.12 foo.py`, dann hat die Variable `__name__` den Wert `"__main__"`.

Die `if`-Verzweigung führt also dazu, dass die `print`-Anweisung nur dann ausgeführt wird, wenn `foo.py` als Programm ausgeführt wird, und ansonsten ignoriert wird.

Verwenden Sie in diesem und allen folgenden Übungsblättern diese Technik, um dafür zu sorgen, dass alle Anweisungen, die keine Definitionen sind, nur dann ausgeführt werden, wenn die Python-Datei auch als Programm ausgeführt wird. Dies ist eine verbreitete Konvention in Python und erlaubt es auch unseren Tutorinnen Ihre Abgaben einfacher zu testen.

Aufgabe 3.1 (Winkel konvertieren; 3 Punkte; Dateien: `conversion.py`, `converter.py`)
In dieser Aufgabe sollen Sie ein Programm schreiben, welches einen Winkel im Gradmaß (degrees, D), Bogenmaß (radians, R) oder Gon (gradians, G) entgegen nimmt und diese zu einem anderen Winkelmaß konvertiert und ausgibt.

Ruft man das Programm auf, um 90 Grad zu Gon zu konvertieren, soll dabei *exakt* die folgende Ein- und Ausgabe erscheinen (Eingaben in blau hervorgehoben):

```
Enter source unit [D / R / G]: D
Enter source value: 90.0
Enter target unit [D / R / G]: G

90.0 D corresponds to 100.0 G
```

Gehen Sie dabei wie folgt vor:

(a) (1 Punkte) Definieren Sie die folgenden Funktionen:

- `degrees_to_radians`
- `radians_to_degrees`
- `degrees_to_gradians`
- `gradians_to_degrees`

in der Datei `conversion.py`. Die Funktionen sollen die Winkel als Argument vom Typ `float` entgegen nehmen und die entsprechend konvertierten Winkel als Wert vom Typ `float` zurückgeben (`return`).

Berücksichtigen Sie, dass die Maße in den folgenden Intervallen gültig sind: degrees : $[0; 360)$, radians : $[0; 2\pi)$, gradians : $[0; 400)$ Verwenden Sie den %-Operator, um das Argument der Funktionen auf diese Intervalle zu begrenzen.

(b) (1 Punkte) Definieren Sie die Funktionen

- `gradians_to_radians`
- `radians_to_gradians`

ebenfalls in der `conversion.py`. Rufen Sie hierzu mehrere Funktionen aus dem vorherigen Aufgabenteil auf, anstatt die mathematischen Formeln zur Konvertierung direkt zu verwenden.

Sie können einige Fälle testen, indem Sie `math` importieren und die folgenden Assertions am Ende der Datei unter `if __name__ == "__main__"` einfügen:

```
assert math.isclose(degrees_to_radians(45), math.pi / 4)
assert math.isclose(radians_to_degrees(math.pi), 180)
assert math.isclose(radians_to_degrees(3 * math.pi), 180)
assert math.isclose(degrees_to_gradians(270.0), 300.0)
assert math.isclose(gradians_to_degrees(100.0), 90.0)
assert math.isclose(gradians_to_degrees(500.0), 90.0)
assert math.isclose(gradians_to_degrees(-300.0), 90.0)
assert math.isclose(gradians_to_radians(300), 3 * math.pi / 2)
assert math.isclose(radians_to_gradians(math.pi / 2), 100)
```

`math.isclose`[2] prüft, ob 2 float-Werte fast gleich sind. Der `==`-Operator ist aufgrund der Ungenauigkeit der float-Repräsentation oft nicht richtig. (Erinnern Sie sich an den Ausdruck `2 - 2.1` aus der Vorlesung 02, welcher nicht gleich `-0.1` ist.)

(c) (1 Punkte) Schreiben Sie ein Skript `converter.py`, das die Funktionen aus `conversion.py` importiert und zusammen mit `input`, `print` und `if`-Verzweigungen das gewünschte Verhalten erzeugt, wie im Einleitungstext der Aufgabe beschrieben.

Aufgabe 3.2 (Quadratische Gleichungen; 3 Punkte; Datei: `quadratic.py`)

Schreiben Sie eine Funktion `quadratic` mit 4 Parametern: `a, b, c, sol`. Das bedeutet: Verwenden Sie *kein* `input`/`print`! Die ersten 3 Parameter (jeweils `float`) stellen die gleichnamigen Koeffizienten einer allgemeinen Gleichung der Form

$$ax^2 + bx + c = 0$$

[2] https://docs.python.org/3/library/math.html#math.isclose

dar. Die Funktion soll reelle Lösungen der Gleichung bestimmen. Hat die Gleichung keine reelle Lösung, soll `None` zurückgegeben werden. Hat sie zwei Lösungen, dann entscheidet `sol` (von Typ `bool`), welche zurückgegeben werden soll. Ist `sol True`, soll die größere der beiden Lösungen zurückgegeben werden, ansonsten die kleinere.

Hinweis: Beachten Sie insbesondere auch die unterschiedlichen Fälle, wenn `a` gleich `0` ist. Sie können einige Fälle testen, indem Sie die folgenden Assertions am Ende der Datei unter `if __name__ == "__main__"` einfügen:

```
assert quadratic(1, -2, -3, False) == -1.0
assert quadratic(1, -2, -3, True) == 3.0
assert quadratic(-1, -2, 3, False) == -3.0
assert quadratic(-1, -2, 3, True) == 1.0
assert quadratic(1, 2, 3, True) is None
assert quadratic(1, 2, 3, False) is None
assert quadratic(0, 1, 2, True) == quadratic(0, 1, 2, False) == -2.0
assert quadratic(0, 0, 0, True) == quadratic(0, 0, 0, False) == 0
assert quadratic(1, 2, 1, True) == quadratic(1, 2, 1, False) == -1.0
```

Aufgabe 3.3 (Gruppenarbeit: Akinator; 4 Punkte; Datei: `akinator.py`)

In dieser Aufgabe sollen Sie eine kleine Version des Akinator-Spiels nachprogrammieren. Akinator ist ein Ratespiel, bei dem die Spieler an eine fiktive oder reale Figur denken. Der Akinator stellt dann eine Reihe von Ja- oder Nein-Fragen, um die Identität der Person oder Figur zu erraten. Durch die Antworten schränkt der Akinator die Möglichkeiten ein und gibt schließlich eine Vermutung über die gewählte Person oder Figur ab. Am besten verschaffen Sie sich selbst einen Eindruck, indem Sie ein paar Runden spielen: https://en.akinator.com/

Ihre Version soll sich auf eine kleine Anzahl an möglichen Antworten begrenzen, die der Benutzerin zu Beginn mitgeteilt werden. Mithilfe von `input`, `print` und `if`-Verzweigungen soll das Programm dann abwechselnd Fragen stellen, die von der Benutzerin mit 'Ja' oder 'Nein' beantwortet werden, und dementsprechend folgende Fragen stellen.

Hier ist ein Beispiel:

```
Hallo! Denke an einen Hauptcharakter aus "Harry Potter".
Ich versuche, den Charakter zu erraten!

Fangen wir an:

Ist dein Charakter ein Mensch? [Ja/Nein]: Ja

Kann deine Person zaubern? [Ja/Nein]: Ja

Hat die Person eine Nase? [Ja/Nein]: Nein

Ich hab's! Es ist Lord Voldemort!
```

Oder noch ein Aufruf des gleichen Beispielprogramms:

```
Hallo! Denke an einen Hauptcharakter aus "Harry Potter".
Ich versuche, den Charakter zu erraten!

Fangen wir an:

Ist dein Charakter ein Mensch? [Ja/Nein]: Nein

Ist dein Charakter eine Eule? [Ja/Nein]: Ja

Dann muss es Hedwig sein!
```

Die Themen dürfen Sie sich aussuchen. Sie können Personen, Tiere, Objekte oder etwas ganz anderes auswählen. Lassen Sie Ihrer Kreativität freien Lauf.

Sie dürfen diese Aufgabe in Gruppen bis zu 3 Personen abgeben. Die Bedingungen sind jedoch:

- Alle Gruppenmitglieder müssen der selben Tutorin zugewiesen sein
- Bei einer Gruppe mit `n` Personen muss Ihr Programm mindestens $2n^2 - n + 5$ verschiedene Charaktere/Tiere/Objekte/... besitzen. Mehr sind natürlich auch erlaubt.
- Laden Sie *alle* die Abgabe hoch und schreiben Sie die Kürzel *aller* Gruppenmitglieder in die `NOTES.md`
- Da wir die besten Spiele gern auf der Vorlesungswebsite veröffentlichen möchten, geben Sie uns dafür in der `NOTES.md` die Erlaubnis, wenn das für Sie in Ordnung ist.

Aufgabe 3.4 (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 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.

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.

Vergessen Sie nicht, Ihre Gruppenmitglieder der Aufgabe 3.3 anzugeben!