Einführung in die Programmierung

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

Universität Freiburg
Institut für Informatik
Wintersemester 2024

Übungsblatt 3

Abgabe: Montag, 04.11.2024, 9:00 Uhr

Ankündigungs-Channel

In dem Chat zu dieser Vorlesung gibt es einen speziellen Channel[1], in dem wichtige Ankündigungen zu Tutorien, Übungen, usw. veröffentlicht werden. Um nichts zu verpassen, empfehlen wir Ihnen dort regelmäßig hineinzuschauen.

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 Definitionen in `bar.py` folgendermaßen schreiben:

```
from foo import some_function
```

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

Top-Level-Statemets 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)
```

[1] https://chat.laurel.informatik.uni-freiburg.de/group/2024WS-EidP

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

```
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 diesen in ein anderes Winkelmaß konvertiert und ausgibt.

Ruft man das Programm auf, um 45.0 Grad zu Gon zu konvertieren, soll dabei *exakt* die folgende Ein- und Ausgabe erscheinen (Eingaben in blau hervorgehoben):

```
Source unit [D / R / G]: D
Source value: 45.0
Target unit [D / R / G]: G

90.0 D corresponds to 50.0 G
```

Gehen Sie dabei wie folgt vor:

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

- `degrees_to_radians`
- `radians_to_degrees`
- `gradians_to_radians`

- `radians_to_gradians`

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:

- Gradmaß: $[0, 360)$
- Bogenmaß: $[0, 2\pi)$
- Gon: $[0, 400)$

Verwenden Sie den `%`-Operator, um das Argument der Funktionen auf diese Intervalle zu begrenzen.

(b) (1 Punkte) Definieren Sie die Funktionen

- `degrees_to_gradians`
- `gradians_to_degrees`

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 ihre Funktionen testen, indem Sie das `math` Modul importieren und die folgenden Tests am Ende der Datei hinter `if __name__ == "__main__"` einfügen und dann die Datei ausführen. Beachten Sie, dass ihre Funktionen nicht zwangsläufig korrekt sein müssen, wenn alle Tests erfolgreich durchlaufen.

```
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)
```

Die Funktion `math.isclose`[2] prüft, ob zwei `floats` *fast* gleich sind. Ein Vergleich mit dem `==` Operator ist aufgrund der Ungenauigkeit der `float`-Repräsentation oft nicht ausreichend (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. Da diese Datei als Programm ausgeführt wird, müssen Sie *in dieser*

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

Datei die Top-Level-Statements nicht in einer `if __name__ == "__main__"` Verzweigung schreiben.

Aufgabe 3.2 (Punkte berechnen; 3 Punkte; Datei: `points.py`)

In dieser Aufgabe übernehmen Sie die Rolle einer Tutorin und berechnen die Punkte einer Abgabe basierend auf der Anzahl der Fehler. Nehmen Sie an, dass in den Übungsblättern zwischen einfachen und schweren Aufgaben unterschieden wird, deren Punkte unterschiedlich berechnet werden.

(a) Schreiben Sie zunächst zwei Funktionen `points_easy` und `points_hard`, die die Punkte für den jeweiligen Aufgabentyp berechnet. Die Funktionen nehmen dabei die Punktanzahl `points` (`int`) und die Anzahl der Fehler `mistakes` (`int`) entgegen und berechnen die Anzahl der Punkte, die erreicht wurden. Diese werden wie folgt berechnet:

- Für beiden Aufgabentypen gibt es volle Punktzahl, wenn kein Fehler aufgetreten ist. Außerdem gibt es keine negativen Punkte.
- Für den leichten Aufgabentyp gibt es pro Fehler zwei Punkte Abzug. Sind hingegen mehr als 7 Fehler in der Aufgabe, so gibt es keine Punkte mehr.
- Für den schweren Aufgabentyp gibt es bereits volle Punktanzahl, wenn weniger als 3 Fehler aufgetreten sind. Ansonsten gibt es pro weiterem Fehler (ab dem 3. Fehler) einen Punkt Abzug.

Hinweis: Verwenden Sie `if`-Verzweigungen und/oder die `max`-Funktion[3].

(b) Nehmen Sie an, dass eine Abgabe aus einer leichten und einer schweren Aufgabe besteht, die entsprechend 16 und 14 Punkte wert sind. Schreiben Sie eine Funktion `calculate_points`, die die Anzahl der Fehler `mistakes_easy`, `mistakes_hard` als Argument nimmt und Gesamtpunktzahl der Abgabe berechnet. Verwenden Sie dazu die Funktionen aus Aufgabenteil (a).

Hinweis: Keine Sorge – für die Korrektur Ihrer Abgaben wenden wir natürlich ein anderes Schema an! :)

Aufgabe 3.3 (Gruppenarbeit: Textadventure; 4 Punkte; Datei: `adventure.py`)

In dieser Aufgabe werden Sie als Team ein eigenes Textbasiertes Abenteuerspiel (Text-Adventure[4]) entwickeln. Dabei soll das komplette Spielgeschehen, sowie die Spielaktionen nur durch Text repräsentiert werden.

Ein typisches Text-Adventure beginnt mit einer Einführung in die Spielwelt. Dies kann zum Beispiel eine Gruppe von Abenteurern sein, die sich in einer Höhle befinden und nach einem Ausweg suchen, aber auch ein einsamer Held, der sich auf eine Reise begibt, um ein mächtiges Artefakt zu finden. Ihrer Kreativität ist hier keine Grenzen gesetzt.

[3] https://docs.python.org/3/library/functions.html#max
[4] https://de.wikipedia.org/wiki/Adventure#Textadventures

Im weiteren Verlauf des Spiels soll die Spielerin Entscheidungen treffen können, die den Verlauf der Geschichte beeinflussen. Zum Beispiel:

```
1. Dem Höhleneingang folgen.
2. Über die frei schwingende Hängebrücke laufen.
3. Gemeinsam die steile Felswand erklimmen.
```

Implementieren Sie ein solches Text-Adventure in Python. Verwenden Sie hierzu die `input` und `print` Funktionen zur Interaktion mit der Spielerin, sowie `if`-Verzweigungen zum Implementieren der Spiellogik.

Hier ist ein Beispiel:

```
Gerade seid ihr noch mit Fackeln bewaffnet tiefer in den Dungeon vorgedrungen
und es schien eigentlich gut zu laufen, doch plötzlich hört ihr nur ein zischen
und plötzlich bricht der Boden unter euch zusammen. Im freien Fall schafft ihr
es gerade noch euch aneinander zu klammern und so den Sturz halbwegs unbeschadet
zu überstehen, doch aus der Kundschaft wurde schnell ein Überlebenskampf, denn
in der Höhle lauert es nur so vor gefahren. Versuche als Gruppe so schnell wie
möglich dem Dungeon zu entfliehen!

Was möchtest du tun? [1 / 2 / 3]
1. Dem linken Höhleneingang folgen.
2. Über die frei schwingende Hängebrücke laufen.
3. Gemeinsam die steile Felswand erklimmen.

Entscheide dich für eine Option: 1

Ihr lauft durch einen dunklen Pfad und nach kurzer Zeit kommt ihr an eine Treppe,
die zwar ziemlich rutschig ist, aber zumindest nicht so aussieht als würde sie
gleich einbrechen. Jetzt stellt sich nur die Frage, ob ihr lieber hoch oder runter
geht.

Was möchtest du tun? [1 / 2]
1. Der Treppe nach oben folgen
2. Der Treppe nach unten folgen

Entscheide dich für eine Option: 2

Ihr geht also nach unten ...
```

Das Setting und die Charaktere können Sie sich selbst aussuchen, lassen Sie Ihrer Kreativität freien lauf. Außerdem dürfen (und sollen) Sie diese Aufgabe in Gruppen von *bis zu 3 Personen* abgeben. Die Bedingungen sind jedoch:

- Alle Gruppenmitglieder müssen der selben Tutorin zugewiesen sein. Zur Gruppenfindung empfehlen wir Ihnen in die Tutorate zu kommen oder im öffentlichen Chat nachzufragen.
- Bei einer Gruppe mit `n` Personen muss Ihr Programm mindestens $2n^2 - n + 5$ verschiedene Abzweigungen haben. Außerdem soll Ihr Programm mindestens

eine geschachtelte `if`-Verzweigung haben. Mehr sind natürlich auch erlaubt.

- *Alle* Gruppenmitglieder müssen die Abgabe hochladen.
- Schreiben Sie außerdem die RZ-Kürzel (z.B.: xy123) *aller* Gruppenmitglieder (auch Ihres) in die `NOTES.md`. Dies ist auch erforderlich, falls Sie alleine abgeben.
- 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 Webplattform. 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.

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.

Vergessen Sie nicht, Ihre Gruppenmitglieder der Aufgabe 3.3 anzugeben!