Informatik I: Einführung in die Programmierung

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

Universität Freiburg
Institut für Informatik

Übungsblatt 3

Abgabe: Montag, 7.11.2022, 9:00 Uhr morgens

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:

```
$ python3 bar.py
Hi! My name is foo.py!
42
```

Um dies zu verhindern, teilt man seinen Code meistens in zwei Arten von Python-Dateien auf:

(a) Python-Dateien, die selbst keine ausführbare Programme darstellen, sondern nur Funktionalität bereitstellen, die von anderen Python-Dateien importiert wird. Diese Dateien enthalten nur Definitionen und keine anderen Anweisungen, wie z.B. das `print("Hi! My name is foo.py!")` in der `foo.py`. Eine Ansammlung von einer oder mehreren solcher Dateien nennt man auch eine Bibliothek (englisch *Library*, kurz *lib*).

(b) Python-Dateien, die den Einstiegspunkt in ein Programm darstellen. Diese Dateien enthalten auch Anweisungen, die nicht Teil einer Definition sind, aber können dafür auch nicht sinnvoll in andere Dateien importiert werden. Solche Dateien nennt man auch Applikationen (englisch *applications*, kurz *app*).

Diese Aufteilung bringt den Vorteil, dass man die Funktionalität aus den Bibliotheks-Dateien potentiell in mehreren Anwendungen wiederverwenden kann, ohne das Verhalten der Anwendungen auf ungewünschte Weise zu beeinflussen. Ein Beispiel für eine Bibliotheks-Datei ist das `math`-Modul, dass Sie auch in der Vorlesung kennengelernt haben. Da diese Datei nur Definitionen enthält, kann z.B. `math.sin` oder `math.pi` in vielen verschiedenen Anwendungen und anderen Bibliotheken verwendet werden, ohne dass direkt irgendwelche Ein- oder Ausgaben erzeugt werden, die für die Anwendungen keinen Sinn ergeben würden.

Für sehr kleine Programme, wie in unseren Übungen, gibt es noch eine weitere Möglichkeit das Ausführen von Anweisungen beim Importieren zu verhindern. Wir können 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 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.

Dies kann z.B. nützlich sein, um eine Bibliotheks-Datei zu testen.

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 ein sinnvolle und weitverbreitete Konvention in

Python und erlaubt es auch unseren Tutoren Ihre Abgaben einfacher zu testen.

Aufgabe 3.1 (Mantelfläche; 4 Punkte; Dateien: `cone_area_lib.py`, `cone_area_app.py`)
In dieser Aufgabe sollen Sie Ihre Lösung zur Berechnung der Mantelfläche eines Kegels aus Aufgabe 2.3 des letzten Übungsblattes umschreiben und dabei Funktionen und mehrere Dateien verwenden. Falls Sie die Aufgabe auf dem vorherigen Übungsblatt nicht gelöst haben, können Sie die Musterlösung als Ausgangspunkt verwenden.

(a) (2 Punkte) Erstellen Sie die Datei `cone_area_lib.py` und definieren Sie dort eine Funktion `cone_area`, welche den Radius und die Höhe eines Kegels als Argumente vom Typ `float` entgegen nimmt und die Mantelfläche des Kegels als Wert vom Typ `float` zurückgibt.

Die Funktion soll keine Seiteneffekte haben, d.h. sie soll ausschließlich die Argumente verwenden um die Mantelfläche zu berechnen und diese zurückgeben, aber keine Funktionen wie `print` oder `input` aufrufen.

Beispielaufruf:

```
>>> cone_area(3.0, 5.0)
54.96
```

(b) (2 Punkte) Erstellen Sie die Datei `cone_area_app.py` und importieren Sie dort die Funktion `cone_area` aus der Datei `cone_area_lib.py`.

Verwenden Sie die Funktionen `cone_area`, `input` und `print` um das gleiche Verhalten wie bei Aufgabe 2.3 zu erzeugen, d.h. das Ausführen von `cone_area_app.py` soll für einen Kegel mit Radius 3 und Höhe 5 folgende Ausgabe erzeugen:

```
$ python3 cone_area_app.py
Radius: 3.0
Höhe: 5.0
Mantelfläche: 54.96
```

Die `if`-Verzweigung mit der Variable `__name__` ist hier nicht notwendig, da die `cone_area_app.py` keine Definitionen bereitstellt und es daher auch keinen Sinn ergibt sie in eine anderen Python-Datei zu importieren.

Aufgabe 3.2 (Temperatur-Konvertierung; 6 Punkte; Datei: `temperature.py`)

In dieser Aufgabe sollen Sie ein Programm schreiben, welches eine Temperatur in Celsius (`C`), Fahrenheit (`F`) oder Kelvin (`K`) entgegen nimmt und diese zu einer anderen Temperatureinheit konvertiert und ausgibt.

Ruft man das Programm auf, um 42 Grad Celsius nach Kelvin zu konvertieren, soll dabei *exakt* die folgende Ein- und Ausgabe erscheinen (Benutzereingaben in blau hervorgehoben):

```
$ python3 temperature.py
Enter source unit [C / F / K]: C
Enter source value: 42.0
Enter target unit [C / F / K]: K

42.0 C corresponds to 315.15 K.
```

Gehen Sie dabei wie folgt vor:

(a) (2 Punkte) Definieren Sie die folgenden Funktionen:

- `celsius_to_fahrenheit`
- `fahrenheit_to_celsius`
- `celsius_to_kelvin`
- `kelvin_to_celsius`

Die Funktionen sollen die Temperatur als Argument vom Typ `float` entgegen nehmen und die entsprechend konvertierte Temperatur als Wert vom Typ `float` zurückgeben.

Wie in der vorherigen Aufgabe, sollen diese Funktionen keine Seiteneffekte haben.

Beispielaufruf:

```
>>> celsius_to_fahrenheit(42.0)
107.6
```

(b) (2 Punkte) Definieren Sie die Funktionen

- `fahrenheit_to_kelvin`
- `kelvin_to_fahrenheit`

Rufen Sie hierzu mehrere Funktionen aus dem vorherigen Aufgabenteil auf, anstatt die mathematischen Formeln zur Konvertierung direkt zu verwenden.

(c) (2 Punkte) Verwenden Sie die Funktionen aus den vorherigen Aufgabenteilen zusammen mit `input`, `print` und `if`-Verzweigungen, um das gewünschte Verhalten zu erzeugen (wie im Einleitungstext der Aufgabe beschrieben).

Verwenden Sie die Technik mit der `__name__`-Variable, um sicherzustellen, dass dieser Code nur ausgeführt wird, wenn die Datei auch als Programm ausgeführt wird (und nicht importiert wird).

Aufgabe 3.3 (Gruppenarbeit: Text Adventure; Datei: `game.py`; Punkte: 8)

Schreiben Sie ein kleines Text-Adventure.

Verwenden Sie mindestens die `input`- und `print`-Funktionen und mehrere `if-else`-Verzweigungen.

Wenn der gleiche Handlungspfad auf mehrere Weisen ausgelöst werden können soll, empfiehlt es sich diesen in eine Funktion zu schreiben und diese Funktion dann mehrmals aufzurufen, anstatt Kopien des gesamten Hanldungspfads zu erstellen.

Variablenzuweisungen können hilfreich sein, wenn Sie sich Eingaben merken möchten, die dann zu einem späteren Zeitpunkt erst relevant werden sollen.

Gestalten Sie Ihr Spiel so, dass man es spielen kann ohne den Pythoncode kennen zu müssen. Das Spiel sollte also dem Benutzer per `print` mitteilen welche gültigen Eingaben es gibt (mal abgesehen von easter eggs).

Die kreativsten Text-Adventures werden anschließend in einer Hall-of-Fame auf unsere Webseite veröffentlicht[1], sodass auch ihre Kommilitonen und die Nachwelt sich daran erfreuen können.

Wir verstehen, dass es bei dieser Aufgabe verlockend ist auch fortgeschrittenere Features von Python zu verwenden, die wir bisher noch nicht in der Vorlesung behandelt haben (z.B. Schleifen). Für die Hall-of-Fame können Sie daher eine Extra-Abgabe in eine Extra-Version`game-hof.py` hochladen, die wir nicht bewerten sondern nur spielen werden. In diesem Fall müssen Sie trotzdem eine kurze `game.py`-Datei zur Bewertung hochladen, die keine fortgeschrittenen Features verwendet und wenigstens formal die Anforderungen dieser Aufgabe erfüllt.

Hier ein Beispiel für ein simples Text-Adventure (Usereingaben in blau):

```
Du stehst im Wald an einer Weggabelung.
Was möchtest du tun? [ links / rechts / umdrehen ]
> links

Du läufst nach links und nach ein paar Minuten stehst du vor einem alten Haus.
Was möchtest du tun? [ klopfen / türe öffnen ]
> türe öffnen

Du öffnest die Türe und läufst durch den Flur in den nächsten Raum.
Der Raum hat eine auffällig moderne Einrichtung.
Auf einem Tisch steht ein Laptop.
```

[1] Natürlich nur falls Sie damit einverstanden sind und optional auch anonym oder unter Pseudonym.

```
Die einzige Datei auf dem Laptop heißt 'sheet01.pdf'.
Was möchtest du tun? [ eine Kurzbiographie über Ada Lovelace schreiben ]
> nach einem Strick umsehen!!1

Ungültige Eingabe. Wie durch eine magische Kraft bewegt, setzt du dich an den
Laptop und schreibst eine Kurzbiographie über Ada Lovelace.
```

Beachten Sie unbedingt die notwendigen Angaben zu Gruppenaufgaben in Aufgabe 3.4! Abgaben ohne diese Angaben werden nicht bewertet!

Aufgabe 3.4 (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 `3.5 h` steht dabei für 3 Stunden 30 Minuten.

Bei Gruppenaufgaben müssen alle Gruppenmitglieder die RZ-Accounts aller Gruppenmitglieder angeben und die Abgabe hochladen, 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.