Einführung in die Programmierung

Prof. Dr. Peter Thiemann
Marius Weidner
Simon Dorer, Moritz Steffl

Universität Freiburg
Institut für Informatik
Wintersemester 2026

Übungsblatt 3

Abgabe: Dienstag, 10.11.2026, 10:00 Uhr

Bitte lesen Sie die folgenden Hinweise durch, bevor Sie mit den Aufgaben beginnen:

Importieren von eigenen Modulen

In der Vorlesung wurde gezeigt, wie Sie Definitionen aus dem `math`-Modul von Pythons 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[1]:

```
from foo import some_function
```

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

Abgaberegel: Top-Level-Statements 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.

Angenommen die Dateien `foo.py` und `bar.py` haben folgenden Inhalt:

`foo.py`:

```
1 def some_function(x: int) -> None:
2     print(x)
3
4 print("Hi! My name is foo.py!")
```

`bar.py`:

```
1 from foo import some_function
2
3 some_function(42)
```

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

Terminal

```
$ python3.14 bar.py
Hi! My name is foo.py!
42
```

[1] Führen Sie das Programm aus dem Ordner aus, in dem beide Dateien liegen. Wechseln Sie dazu im Terminal gegebenenfalls zuerst mit dem Befehl `cd` in diesen Ordner.

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

`foo.py`:

```python
def some_function(x: int) -> None:
    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 Namen des Moduls.
- Wird `foo.py` aber als Programm ausgeführt, z.B. mit `python3.14 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 weder Definitionen noch Importe sind, nur dann ausgeführt werden, wenn die Python-Datei auch als Programm ausgeführt wird. Dies ist eine verbreitete Konvention und erlaubt es unseren Tutorinnen Ihre Abgaben einfacher zu testen.

Abgaberegel: Typannotationen

In der Vorlesung haben Sie Funktionen mit Typannotationen für Argumente und Rückgabewerte kennengelernt. Diese sind für *alle* von Ihnen implementierten Funktionen notwendig.

Eine vollständige Übersicht aller gültigen Abgaberegeln finden Sie hier auf unserer Website.

Aufgabe 3.1 (Kreisgrößen; 3 Punkte; Dateien: `conversion.py`, `converter.py`)

In dieser Aufgabe sollen Sie ein Programm schreiben, das aus einer gegebenen Kreisgröße eine andere Größe desselben Kreises berechnet. Zur Auswahl stehen der Radius (r), der Umfang (U) und die Fläche (A). Radius und Umfang werden in Zentimetern, die Fläche in Quadratzentimetern angegeben.

Für einen Kreis gelten die Formeln

$$U = 2\pi r, \qquad r = \frac{U}{2\pi}, \qquad A = \pi r^2, \qquad r = \sqrt{\frac{A}{\pi}}.$$

Importieren Sie die Konstante `pi` und die Quadratwurzelfunktion `sqrt` für die Berechnung der Formeln aus dem `math`-Modul.

Ruft man das Programm auf, um aus einem Radius von 2 Zentimetern die Fläche zu berechnen, soll dabei *exakt* die folgende Ein- und Ausgabe erscheinen (Eingaben in Türkis hervorgehoben):

```
Gegeben [r / U / A]: r
Wert: 2.0
Gesucht [r / U / A]: A

r = 2.0 entspricht A = 12.57
```

Gehen Sie dabei wie folgt vor:

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

- `radius_to_circumference`
- `circumference_to_radius`
- `radius_to_area`
- `area_to_radius`

in der Datei `conversion.py`. Jede Funktion soll den Wert der gegebenen Kreisgröße als Argument entgegennehmen und den Wert der gesuchten Kreisgröße zurückgeben. Die Funktionen sollen keine Ein- oder Ausgaben durchführen und ihre Ergebnisse nicht runden.

Zum Beispiel:

```
>>> circumference_to_radius(2 * pi)
1.0
>>> area_to_radius(4 * pi)
2.0
>>> radius_to_circumference(1.0)
6.283185307179586
>>> radius_to_area(2.0)
12.566370614359172
```

(b) (1 Punkt) Definieren Sie die Funktionen

- `circumference_to_area`
- `area_to_circumference`

ebenfalls in der Datei `conversion.py`. Berechnen Sie jeweils zuerst den Radius und daraus die gesuchte Größe. Rufen Sie hierzu jeweils zwei Funktionen aus dem vorherigen Aufgabenteil auf, anstatt die Formeln direkt zu verwenden. Runden Sie auch hier keine Zwischenergebnisse oder Rückgabewerte.

Zum Beispiel:

```
>>> circumference_to_area(2 * pi)
3.141592653589793
>>> area_to_circumference(pi)
6.283185307179586
```

(c) (1 Punkt) Schreiben Sie ein Skript `converter.py`, das die Funktionen aus `conversion.py` importiert. Verwenden Sie `input`, `print` und `if`-Verzweigungen,

um das gewünschte Verhalten zu erzeugen. Das Programm soll alle sechs Umrechnungsrichtungen unterstützen. Sind gegebene und gesuchte Größe gleich, bleibt der Wert unverändert. Runden Sie den berechneten Wert erst für die Ausgabe auf zwei Nachkommastellen. Dies können Sie mit der `round`-Funktion erreichen:

```
>>> round(3.14159, 2)
3.14
>>> round(4.2, 2)
4.2
```

Sie dürfen annehmen, dass alle eingegebenen Zahlen nichtnegativ sind und nur die angegebenen Kürzel verwendet werden. Da diese Datei als Programm ausgeführt wird, müssen Sie *in dieser Datei* die Top-Level-Statements nicht hinter eine `if __name__ == "__main__"` Verzweigung schreiben.

Aufgabe 3.2 (Datumsrechner; 3 Punkte; Datei: `dates.py`)

In dieser Aufgabe implementieren Sie die Datumslogik für einen kleinen Kalender. Verwenden Sie `if`-Verzweigungen, um die verschiedenen Fälle zu unterscheiden.

(a) (1 Punkt) Schreiben Sie eine Funktion `is_leap_year`, die eine Jahreszahl `year` entgegennimmt und `True` zurückgibt, wenn `year` ein Schaltjahr ist, andernfalls `False`. Ein Jahr ist ein Schaltjahr, wenn es durch 4 teilbar ist, es sei denn, es ist durch 100 teilbar. Ist es durch 400 teilbar, ist es trotzdem ein Schaltjahr.

```
>>> is_leap_year(2024)
True
>>> is_leap_year(2023)
False
>>> is_leap_year(1900)
False
>>> is_leap_year(2000)
True
```

(b) (2 Punkte) Schreiben Sie eine Funktion `days_in_month`, die eine Monatszahl (z.B. 5 für Mai) `month` und eine Jahreszahl `year` entgegennimmt und die Anzahl der Tage dieses Monats zurückgibt. Beachten Sie dabei Schaltjahre und verwenden Sie die Funktion `is_leap_year` aus Aufgabenteil (a). Sie können davon ausgehen, dass die Eingaben einer korrekten Monats- oder Jahreszahl entsprechen.

```
>>> days_in_month(2, 2024)
29
>>> days_in_month(2, 2023)
28
>>> days_in_month(1, 2026)
31
>>> days_in_month(4, 2026)
30
```

```
>>> days_in_month(12, 2026)
31
```

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

In dieser Aufgabe sollen Sie als Team ein eigenes textbasiertes Abenteuerspiel (Text-Adventure[2]) 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 sind hier keine Grenzen gesetzt.

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 Funktionen `input` und `print` zur Interaktion mit der Spielerin sowie `if`-Verzweigungen zum Implementieren der Spiellogik.

Hier ist ein Beispiel (Eingaben in Türkis hervorgehoben):

```
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 Erkundung 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] https://de.wikipedia.org/wiki/Adventure#Textadventures

```
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 derselben Tutorin zugewiesen sein. Zur Gruppenfindung empfehlen wir Ihnen in die Tutorate zu kommen oder im Forum nachzufragen.
- Ihr Spiel muss mindestens $2n^2 - n + 5$ verschiedene Handlungsoptionen anbieten, wobei $n \in \{1, 2, 3\}$ die Anzahl der Gruppenmitglieder ist.
- 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.**

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 Skript zum Abgabesystem gezeigt wurde.

Vergessen Sie nicht, Ihre Gruppenmitglieder der Aufgabe 3.3 anzugeben!