Bisherige Themen

Letzte Woche haben wir folgende Themen behandelt:

- Auf bestimmte Positionen oder Bereiche von sortierten Sequenzen (Strings, Listen) zuzugreifen
- Zwischen veränderlichen und unveränderlichen Datentypen (mutable/immutable) zu unterscheiden
- Verschiedene Operationen auf Listen anzuwenden
- Dictionarys zu erstellen
- Die Werte aus Dictionarys abzufragen oder neu hinzuzufügen
- Die Wertpaare in einem Dictionary nach den Keys oder Values sortiert zu verarbeiten
- Zufallszahlen zu erzeugen

Falls Sie Fragen zu diesen Themen oder zu den Übungsaufgaben haben, sprechen Sie uns bitte an!

Und so geht es weiter:

Editor-Trick des Tages: Mehrere Dateien gleichzeitig anzeigen, zu letztem aktivem Tab wechseln

Oft beschäftigen wir uns mit mehr als nur einer Code-Datei auf einmal. Dabei kann es hilfreich sein, die Dateien nebeneinander zu betrachten. Dazu können Sie die Command Palette öffnen und `>layout` eingeben. Wählen Sie dann den Befehl `View: Two Columns Editor Layout` aus.

Jetzt können Sie in der neu erschienenen Spalte eine Datei öffnen, oder Sie verschieben den Tab einer bereits offenen Datei in den rechten Bereich.

Um zwischen den Tabs in einer Spalte zu wechseln, können Sie wie im Browser `Ctrl + Tab` verwenden. Mit `Ctrl + 1` springen Sie in die linke Spalte, mit `Ctrl + 2` springen Sie in die rechte Spalte - auch wenn die rechte Spalte noch nicht offen war. Sie können auf diese Weise auch noch mehr Spalten öffnen.

Besonders komfortabel daran ist, dass auch der Cursor in den Tabs immer genau da platziert wird, wo er sich zuletzt befunden hat.

Besprechung zu Übungsaufgabe 03-03

Heute besprechen wir die Funktionalität von Dictionarys noch einmal im Detail, anhand der Übungsaufgabe zu Würfelhäufigkeiten. Falls Sie diese Aufgabe bearbeitet haben und sie noch nicht eingereicht haben, können Sie ***jetzt*** noch eine Mail an uns schicken mit Ihrer Lösung im Anhang. Einreichungen nach dieser Besprechung können nicht mehr auf Ihren BN angerechnet werden!

Die Aufgabenstellung lautete so:

```
Aufgabe 03-03: Dictionarys und Zufallszahlen
============================================

Schreiben Sie ein Programm, das einen normalen, fairen, 6-seitigen Würfel
simuliert. Würfeln Sie 10, 100, 10000 mal und speichern Sie in Dictionaries
die Häufigkeit der erwürfelten Augenzahlen.
Lassen Sie sich die Ergebnisse der Versuche auf geeignete Weise ausgeben.

Was ist Ihre Erwartung bzgl. der Werte der Dictionaries? Wird diese erfüllt?
Welche Beobachtung machen Sie, wenn Sie die Versuche wiederholen?
```

Beginnen wir mit einem Brainstorming. Welche der bisher bekannten Konzepte sind für diese Aufgabe sinnvoll?

- Strings?
- Integer-Zahlen?
- Float-Zahlen?
- Listen?
- Dictionarys?
- for-Schleifen?
- if-Blöcke mit optionalem elif und else?

In der Aufgabenstellung wird explizit erwähnt, dass die Ergebnisse in einem Dictionary gespeichert werden sollen. Hier müssen Sie sich überlegen, welche Struktur das Dictionary haben soll: Welche Informationen sind geeignete Schlüssel, und welche Informationen sind die dazugehörigen Werte? Welchen Typ sollten die Schlüssel und die Werte jeweils haben?

Definieren Sie hier ein Beispieldictionary, das ausgedachte Daten enthält und einer sinnvollen Struktur für diese Aufgabe folgt.

In [ ]:

```
# Verändern Sie dieses Dictionary, indem Sie ausgedachte
# Beispieldaten in einer von Ihnen gewählten Struktur einfügen.
ergebnisse = {}
```

In der Aufgabenstellung steht, dass Sie 10 (bzw 100, 10000) mal würfeln sollen. Welches Konstrukt, das wir bereits gelernt haben, ist nützlich, um so eine Wiederholung von Programmteilen umzusetzen?

Schreiben Sie hier die Codezeilen, die dafür zuständig sind, 10 mal zu würfeln. Sie müssen das Ergebnis jedes Wurfs noch nicht weiterverarbeiten, das ist erst im nächsten Schritt dran.

In [ ]:

```
# Erzeugen Sie hier mit bekannten Python-Konstruktionen
# 10 Zufallszahlen. Wie erzeugen wir eine Zufallszahl?
# Wie wiederholen wir einen Code-Abschnitt genau 10 mal?
```

Wir beginnen jetzt, die Teile zusammenzufügen, die wir eben einzeln geschrieben haben. Dazu können wir erst einmal überlegen, wie unser Dictionary initialisiert werden kann. Wir können zum Beispiel alle Schlüssel, die wir erwarten, schon anlegen und jedem Schlüssel zu Beginn den Wert 0 zuweisen.

Schließlich können wir den Code aus dem vorigen Code-Kästchen verwenden und dafür sorgen, dass bei jedem unserer 10 Würfe der passende Wert im Dictionary aktualisiert wird.

In [ ]:

```
# Hier Ihr Dictionary initialisieren:

# Hier Ihren Code für 10 Würfe einfügen:

# Nicht vergessen, die Werte im Dictionary anzupassen!
```

Teilen und Herrschen

Der Prozess, den wir anwenden, um Programmieraufgaben nach und nach in kleinen Einzelteilen zu lösen, wird manchmal *Teilen und Herrschen* genannt. Die Aufgabe als ganzes ist zunächst etwas unübersichtlich. Um das Problem handhabbar zu machen, zerlegen wir es in Teilprobleme. Jedes einzelne Teilproblem ist dann gar nicht so schwer zu lösen.

Eine gute Beschreibung dieser Vorgehensweise finden Sie [hier (https://www.cs.kent.ac.uk/people/staff/djb/ProblemSolving/doc1.n-S-3.html)](https://www.cs.kent.ac.uk/people/staff/djb/ProblemSolving/doc1.n-S-3.html). In diesem Artikel wird das Lösen von Programmieraufgaben mit Aufgaben aus der echten Welt verglichen: Was gibt es heute zum Abendessen?

```
Tonight's dinner:
    Find a nice recipe.
    Make a list of ingredients to buy.
    Go shopping for the ingredients that are needed.
    Cook the dinner.
```

> *Each of these new tasks is a slightly smaller problem to solve than the original task but, together, they constitute a solution to the overall problem. We shall often see this kind of approach in solving computer problems.*
>
> *Having broken the initial task down one level, we will often find that we need to break these second level tasks down still further:*

```
Tonight's dinner:
    Find a nice recipe.
    Make a list of ingredients to buy.
    Go shopping for the ingredients that are needed.
    Cook the dinner.
Find a nice recipe:
    Look in a few cookery books for a short list.
    Check with everyone which recipe they would prefer.
Make a list of ingredients to buy:
    Make a list of ingredients in the recipe.
    Check the cupboards for which ingredients are missing.
Go shopping for the ingredients that are needed:
    Get the car out.
    Drive to Sainsburys.
    Buy the ingredients.
    Drive back home.
Cook the dinner:
    Prepare the ingredients.
    Put the oven on to the right temperature.
    Follow the recipe.
```

> *This process might continue through several stages until the problem has reached the sort of level that we feel we can make a start on.*

Ein etwas ausführlicheres Tutorial ist [hier (https://happycoding.io/tutorials/how-to/program)](https://happycoding.io/tutorials/how-to/program) zu finden.

Versuchen Sie ab jetzt, Übungsaufgaben zuerst in Teilprobleme zu zerlegen, damit Sie diese dann lösen können. Das hilft Ihnen, sich einen Überblick zu verschaffen. Sie können dann sich selbst und den Dozierenden konkretere Fragen stellen.

User-Input während der Laufzeit

Im folgenden Code-Beispiel wird ein Dictionary erstellt, indem Sie während der Laufzeit des Programms die Antworten auf einzelne Fragen eingeben. Eingaben während der Laufzeit werden mit dem Befehl `input()` abgefragt, das Ergebnis kann in einer Variable gespeichert werden. Die Eingabe wird mit `Enter` bestätigt.

Führen Sie das Programm aus und beobachten Sie, wie der Inhalt des Dictionarys sich in jedem Schleifendurchlauf ändert.

In [ ]:

```
datentypen = ["Strings", "Listen", "Zahlen", "Dictionarys"]
mutability = {}    # leeres Dictionary wird hier erstellt und
                   # unten nach und nach gefüllt

for typ in datentypen:
    print("Sind " + typ + " veränderlich?")

    antwort = input()  # !!!

    mutability[typ] = antwort
    print("Ihre bisherigen Antworten: " + str(mutability))
```

Die `input()` -Funktion kann auch mit einem Parameter in Form eines Strings ausgeführt werden. Dann erscheint beim Ausführen des Programms ein Prompt, der dem übergebenen String entspricht.

In [ ]:

```
# ohne Prompt
print("Ihre Nachricht (ohne Prompt): ")
message = input()     # !!!
print(message)

# mit Prompt
message = input("Ihre Nachricht (mit Prompt): ") # !!!
print(message)
```

Dateien lesen und schreiben

Der Inhalt von Variablen ist immer bis zum Ende der Laufzeit verfügbar. Wenn wir Informationen **langfristig speichern** wollen, machen wir das in Dateien, die im Dateisystem des Computers, auf dem das Programm läuft, abgelegt werden.

Der folgende Code erzeugt eine Datei und schreibt alles, was Sie während der Laufzeit eingeben, in diese Datei:

In [ ]:

```
filename = "secrets.txt"

with open(filename, "w") as outfile:
    content = input("Hier können Sie Ihre Geheimnisse eingeben:\n")

    # Dieser print-Befehl gibt den Inhalt in die Datei outfile aus.
    print(content, file=outfile)

    # Dieser print-Befehl zeigt die Ausgabe direkt an.
    print("Daten wurden in der Datei " + filename + " gespeichert!")
```

Den Dateinamen können Sie frei wählen. Wir verwenden meist die Dateiendung `.txt` für Textdateien. Wenn wir nur den Namen angeben, wird die Datei im gleichen Verzeichnis erzeugt, in dem unser Pythoncode ausgeführt wird, hier also in dem Verzeichnis, in dem Sie auch das Jupyter Notebook abgelegt haben. Um einen anderen Speicherort zu wählen, geben wir einfach absolute Pfade an, z.B. `filename = "D:/Files/secrets.txt"` .

Das Öffnen der Datei ähnelt optisch einer Schleife oder einem `if` -Block: Auch hier sind einige Zeilen nach rechts **eingerückt**. Der Block wird mit der Zeile eröffnet, die die Datei aus dem Dateisystem öffnet:

```
with open(<filename>, <read-or-write mode>) as <variablename>:
    <...>
```

Für alle eingerückten Zeilen ist die geöffnete Datei verfügbar. Sobald wieder Code ohne Einrückung folgt, wird die Datei automatisch vom Interpreter **geschlossen**.

Um Inhalte in die Datei zu schreiben, können wir `print` mit dem zusätzlichen Paramenter `file` verwenden. Dabei muss der Wert für `file` der Variablenname sein, den wir beim Öffnen der Datei erzeugt haben.

Wir haben oben den Modus **`"w" wie write`** (schreiben) gewählt. Bei jeder Ausführung des Codes wird die Datei komplett überschrieben. Wenn wir stattdessen **`"a" wie append`** (anhängen) verwenden, wird neuer Inhalt immer nach dem bisherigen Inhalt platziert, sodass nichts verlorengeht. Probieren Sie es oben aus!

Um eine existierende Datei in unserem Pythoncode zu öffnen, verwenden wir den Modus **`"r" wie read`** (lesen). Dann können wir z.B. mit einer `for` -Schleife durch den Inhalt der Datei iterieren. Dabei wird jede Zeile als String aus der Datei gelesen und kann mit unseren verfügbaren Stringmethoden weiterverarbeitet werden.

Konventionell verwenden wir für Dateien, die geschrieben werden, den Variablennamen `outfile` , und für Dateien, die gelesen werden, den Variablennamen `infile` .

In [ ]:

```
filename = "secrets.txt"
with open(filename, "r") as infile:
    for line in infile:
        print(line)
```

Wenn wir die Datei nicht zeilenweise verarbeiten möchten, sondern alles auf einmal lesen wollen, verwenden wir `read()`. Achtung: Bei großen Dateien kann das schnell zu Problemen führen, da der gesamte Inhalt der Datei in den Arbeitsspeicher passen muss.

Bei vielen Dateiformaten hier im Kurs ist das Lesen einzelner Zeilen angebracht.

So wird `read()` verwendet:

In [ ]:

```
filename = "secrets.txt"
with open(filename, "r") as infile:
    content = infile.read()
    print(content)
```

Zusammenfassung

Sie haben heute gelernt,

- Komplexe Aufgabenstellungen in Teilprobleme zu zerlegen, die sich leichter bearbeiten lassen
- Nutzereingaben während des Programms abzufragen
- Dateien zu lesen (Modus "r")
- Dateien anzulegen oder zu überschreiben (Modus "w")
- Dateien zu ergänzen (Modus "a")

Und morgen...

In der Übung am 30.10.2019 werden Sie üben, Nutzereingaben abzufragen und Dateien zu schreiben und zu lesen.

Wiederholungsfragen

In [ ]:

```
# Wie können Sie in Python prüfen, welchen Datentyp
# der Wert einer Variable hat?
```

In [ ]:

```
# Wie können Sie in einem String alle Vorkommen der Zeichen
# "ä", "ö" und "ü" durch die Alternativen "ae", "oe" und "ue"
# ersetzen?

# Wie können Sie sicherstellen, dass auch großgeschriebene
# Umlaute gefunden und korrekt ersetzt werden?
```

In [ ]:

```
# Wie können Sie in Python einer Variable, die eine Zahl enthält,
# einen um genau 1 größeren Wert zuweisen?
```

In [ ]:

```
# Wie können Sie den Wert, der einem Dictionary-Schlüssel
# zugeordnet ist, um genau 1 erhöhen?
```

In [ ]:

```
# Wie können Sie in einem String Anführungszeichen ("")
# so einfügen, dass der String mit den Anführungszeichen
# korrekt ausgegeben werden kann?
```

In [ ]:

```
# Wie können Sie eine Liste sortiert ausgeben (ein Element
# pro Ausgabezeile)?
```

In [ ]:

```
# Wie können Sie auf den Teilstring "ben" im String "Nabenschaltung"
# zugreifen?
```

In [ ]:

```
# Wie können Sie unter Angabe negativer Indizes den Teilstring
# "ben" aus dem String "Nabenschaltung" extrahieren?
```

In [ ]:

```
# Wie können Sie das letzte Element aus einer Liste löschen
# und es gleichzeitig ausgeben?

mylist = [1, 2, 3]

# Hier Ihren Code einfügen...
print("...")    # gewünschte Ausgabe: 3

print(mylist)   # gewünschte Ausgabe: [1, 2]
```

In [ ]:

```
# Wie können Sie ermitteln, ob der String, der in einer
# Variable gespeichert ist, länger ist als der String, der
# in einer anderen Variable gespeichert ist?
# Geben Sie zum Schluss aus, ob der erste String länger
# oder kürzer ist als der zweite.

erster_string = "Dieser String hat eine gewisse Länge"
zweiter_string = "Dieser String ist eventuell kürzer"

# Hier Ihren Code einfügen...

print("...")
```