IWGS1_WS1920_Zettel4

July 7, 2020

1 Informatische Werkzeuge in den Geistes- und Sozialwissenschaften I

1.1 Hausaufgabe 4 (Datenstrukturen in Python, 100 Punkte)

Erschienen: 09.11.2019
Abgabe bis: 17.11.2019

Bitte laden Sie Ihre Notebooks bis 23:59 Uhr am Abgabetag in Ihrer Übungsgruppe bei StudOn hoch.

Wenn Ihnen einige der hier verwendeten Konzepte unbekannt sind oder Sie nicht wissen, wie Sie fortfahren sollen, können Sie die Vorlesungsunterlagen zu Rate ziehen oder jederzeit Fragen im Forum stellen, im Tutorium nachfragen, oder Ihrem Tutor eine Mail schreiben.

1.1.1 Aufgabe 4.1 (Listen, 20 Punkte)

Die Aufgabe der meisten Programme ist die Arbeit mit oder die Transformation von Daten (welcher Art auch immer: Wetterdaten, Texte, Buchtitel, Bilder, ...). Deshalb ist Vertrautheit mit den wichtigsten *Datenstrukturen* sehr wichtig, um selbst leicht komplexere Programme schreiben zu können. In dieser Aufgabe werden Sie näher mit *Listen* arbeiten, einer der elementarsten und häufigsten Datenstrukturen.
Details zur Syntax von Listen finden Sie in der Vorlesung, dem Skript oder alternativ auf der Website w3schools.

Aufgabe: * Schreiben Sie in der nächsten Zelle eine Python-Liste in eine Variable und ein Python-Programm, das in zwei separaten Zeilen mit etwas erklärendem Text sowohl das *erste* als auch das *letzte* Element der Liste printet. * Erläutern Sie in der übernächsten Zelle... - ... ob das auch möglich ist, ohne mit einer Schleife über alle Elemente der Liste zu laufen. Wie / Warum nicht? - ... was passiert, wenn Sie dieser Funktion eine Liste mit nur einem Element übergeben. - ... was passiert, wenn Sie dieser Funktion eine leere Liste übergeben.

[1]:
```
# Ihr Code hier! (Doppelklick zum Editieren, Shift + Enter um die Zelle␣
↪auszuführen)
```

Ihre Beschreibung und Erläuterungen hier! (Doppelklick zum Editieren, Shift + Enter um die Zelle auszuführen)

1.1.2 Aufgabe 4.2 (Dictionaries, 20 Punkte)

Eine weitere wichtige Datenstruktur sind *Dictionaries*. Diese bestehen aus so genannten "Key Value Pairs" (also Paaren aus "Schlüssel" und "Wert". Sowohl der Schlüssel als auch der Wert können von beliebigem Typ sein, also Zahlen, Strings, ...) und geben im Gegensatz zu Listen nicht notwendigerweise eine Reihenfolge vor. Es werden lediglich eine Menge von Schlüsseln jeweils mit einem Wert assoziiert. In dieser Aufgabe werden Sie sich etwas näher mit *Dictionaries* auseinander setzen.
Details zur Syntax von Dictionaries finden Sie in der Vorlesung, dem Skript oder alternativ auf der Website w3schools.

Aufgabe: * Schreiben Sie in der nächsten Zelle ein Python-Dictionary, das Namen berühmter Personen mit deren Geburtsjahren verknüpft. In diesem Dictionary sind also Strings die Schlüssel und Zahlen die Werte, z.B. `"Jean-Luc Picard" : 2305`. Die Personen können echt oder fiktiv sein, sie müssen lediglich ein eindeutiges Geburtsjahr haben. Ihr Dictionary sollte mindestens vier Personen enthalten. * Schreiben Sie (in der gleichen Zelle oder in einer weiteren) ein Python-Programm, das die älteste Person (= niedrigstes Geburtsjahr) in Ihrem Dictionary ermittelt. (Wie) können Sie mit einer Schleife über alle Elemente eines Dictionaries laufen? Zum Schluss sollte Ihr Programm printen, in welchem Jahr die älteste Person geboren wurde (es muss nicht sagen, wer diese Person ist).

[2]:
```
# Ihr Code hier! (Doppelklick zum Editieren, Shift + Enter um die Zelle␣
↪auszuführen)
```

1.1.3 Aufgabe 4.3 (Input von Menschen II, 25 Punkte)

Wir haben auf dem letzten Übungszettel bereits diskutiert, wie wichtig in der Informatik Input von den Menschen vor dem Computer ist. In dieser Aufgabe bauen wir darauf auf. Oft ist es nämlich so, dass wir keinen *beliebigen* Input brauchen, sondern genau eine von endlich vielen "akzeptablen" Antworten oder Antwortmustern. Wenn wir zum Beispiel nach einer Schulnote fragen, können wir vielleicht sowohl Eingaben wie `"1+"` oder `"Eins Plus"` verarbeiten, aber wenn wir auch Fälle wie `"Gerade noch so bestanden"` abfangen müssten, damit unser Programm nicht versehentlich abstürzt, wird es komplizierter.

Aufgabe: Schreiben Sie ein Programm, dass den Benutzer oder die Benutzerin nach der liebsten Todsünde fragt. Wenn der Input keine der klassischen Todsünden repräsentiert durch die Strings `"Eitelkeit"`, `"Habgier"`, `"Wollust"`, `"Rachsucht"`, `"Maßlosigkeit"`, `"Eifersucht"` und `"Faulheit"` ist, soll es nochmal und nochmal fragen, bis eine korrekte Antwort kommt. Wurde endlich eine akzeptable Antwort gegeben, soll das Programm den Nutzer / die Nutzerin für die gewählte Sünde entweder loben oder tadeln (Ihre Wahl!).

Hinweis: Sie können diesen Effekt leicht mit einer *Schleife* erreichen. Eine "Sündenliste" kann ebenfalls hilfreich sein, ist aber nicht notwendig.

[3]:
```
# Ihr Code hier! (Doppelklick zum Editieren, Shift + Enter um die Zelle␣
↪auszuführen)
```

1.1.4 Aufgabe 4.4 (Verständnisaufgabe 3, 35 Punkte)

In dieser Aufgabe verwenden wir erstmalig "Sequence Slicing", eine oft verwendete Syntax von Python, die es erlaubt, einfacher mit Listen jeder Art (z.B. sowohl Listen von Zahlen als auch Strings, also Listen von Zeichen) umzugehen. Um an das `n`-te Element einer Liste `exampleList` zu kommen benutzen wir in Python die Syntax `exampleList[n]`, wobei das erste Element den Index (= Position in der Liste) 0 hat, das zweite 1, und so weiter. In der Informatik wird oft bei 0 angefangen zu zählen. Außerdem können wird auf Elemente vom Ende der Liste mit negativen Zahlen zugreifen (so ist z.B. `exampleList[-2]` das vorletzte Element).

Sequence Slicing funktioniert ähnlich und basiert auf den gleichen Prinzipien. Der Unterschied ist, dass wir nicht an ein einzelnes Element wollen, sondern an eine Teilliste. Dafür müssen wir prinzipiell zwei Werte angeben (für Start und Ende), getrennt durch ein `:`. Ungefähr so: `exampleList[n:m]`. Das gibt uns eine Liste, die bei Index `n` startet und alles von dort bis vor Index `m` enthält. Das Element mit Index `m` selbst wird nicht Teil der Liste sein.
Wenn einer der oder beide Werte weggelassen werden, nimmt Python den Anfang bzw. das Ende der Ursprungsliste für den entsprechenden Wert. So können wir zum Beispiel einfach Teillisten wie "Alles ab dem dritten Element" (`[2:]`) oder ähnliches abfragen.

Hier ein paar Beispiele:

[4]:
```
# Eine Liste zum Testen von Beispielen.
testList = ["Michael", "Philipp", "Jonas", "Jacqueline", "Beatrice"]

# Alles _ab_ Element 0 und _vor_ Element 1
# => also eine Liste mit nur dem ersten Element.
```

```
profs = testList[0:1]
print(profs)

# Alles _ab_ Element 1 und _vor_ Element 3
tutoren = testList[1:3]
print(tutoren)

# Alles _ab_ Element 3 bis zum Ende der Liste
frauen = testList[3:]
print(frauen)

# Alles _vor_ dem letzten Element
real = testList[:-1]
print(real)

# Die komplette Liste!
alles = testList[:]
print(alles)
```

```
['Michael']
['Philipp', 'Jonas']
['Jacqueline', 'Beatrice']
['Michael', 'Philipp', 'Jonas', 'Jacqueline']
['Michael', 'Philipp', 'Jonas', 'Jacqueline', 'Beatrice']
```

Nun zur Aufgabe: Nachdem Sie ihr schon mit ihrem letzten Problem so gut weiterhelfen konnten, tritt Ihre gute Freundin Beatrice Beispiel abermals an sie heran und fragt Sie um Rat mit einem Stück Python, das leider nicht genau tut, was es soll. Sie schickt Ihnen eine Kopie einer Datei namens `accounts.txt`, in der sie die Namen von Accounts (einer Online-Community, die sie aufbauen will) speichert. Wie Sie selbst leicht überprüfen können sind aktuell vier Accounts angemeldet. Sie hat nun eine Funktion geschrieben, die herausfindet, ob ein bestimmter Name schon vergeben ist, um Fehler bei der Anmeldung vorzubeugen. Sie erzählt Ihnen, dass die Funktion eigentlich nichts tut, außer die besagte Datei zeilenweise in eine Liste einzulesen, die `\n` newline-Zeichen am Zeilenende, die von Python mit als Teil der Zeile gelesen werden, zu entfernen und dann zu überprüfen, ob der gegebene Name bereits ein Element der Liste ist.

Sie zeigt Ihnen ebenfalls, dass die Funktion für manche eingetragenen Accounts (und ebenfalls für ausgedachte Accounts die nicht in der Datei stehen) funktioniert wie gedacht, aber nicht für alle.

Aufgabe: Machen Sie sich mit der Funktion `usernameTaken` in der nächsten Zelle und mit der Datei `accounts.txt` vertraut. Finden Sie den Fehler, den Beatrice gemacht hat, als sie diese Funktion geschrieben hat. Schreiben Sie Ihre Erkenntnisse in Ihren eigenen Worten in der übernächsten Zelle auf. Sie müssen Beatrice nur erklären, wo der Fehler liegt und was falsch gelaufen ist; es wird nicht verlangt, dass Sie den Code auch korrigieren.

Hinweis: Menschen (und sowohl Sie als auch Beatrice sind Menschen) machen häufig Annahmen (manchmal sogar ohne es zu merken!) die gültig scheinen, es aber tatsächlich nicht in *allen* Fällen stimmen. Es kann sich bei solchen Aufgaben deshalb als hilfreich erweisen, den Inhalt von kritischen Variablen zu printen (eventuell sogar mehrmals), um besser zu verstehen, welche Werte diese

Variablen wann annehmen. So merkt man auch leichter, wenn eine der eigenen Annahmen dann doch plötzlich nicht stimmt. Dieses Vorgehen wird in der Informatik als ”printline-debugging” bezeichnet und kann vielleicht auch hier weiter helfen.

[5]:
```
# Eine Funktion, die einen String erwartet und True/False zurück gibt,
# je nachdem ob der String schon in der accounts.txt-Datei steht.
def usernameTaken(username):
    # Datei im Lesemodus öffnen!
    nameFile = open("accounts.txt", "r+")

    # Alle Zeilen in eine Liste
    nameList = nameFile.readlines()

    # Hier laufe ich über alle Elemente der Liste nameList drüber.
    # len(nameList) gibt mir die Länge der Liste, genau passend für range().
    for i in range(len(nameList)):

        # Der alte Name ist die gesamte Zeile aus der Datei, inklusive dem \n␣
 ↪am Ende.
        oldName = nameList[i]

        # Der neue Name ist der alte, aber ohne das letzte Zeichen.
        # [:-1] ist "sequence slicing" Syntax für den ganzen String, ohne das␣
 ↪letzte Zeichen
        newName = oldName[:-1]

        # Setze den neuen Namen in der Liste an die Stelle des alten.
        nameList[i] = newName

    # Datei schließen nicht vergessen!
    nameFile.close()

    # Ein Name ist bereits vergeben, falls er in der Liste steht.
    vergeben = username in nameList
    return vergeben

print(usernameTaken("jonas.betzendahl"))   # Dieser Aufruf gibt "True" zurück,␣
 ↪wie gedacht, weil der String in der Datei steht.
print(usernameTaken("nicht.in.der.datei")) # Dieser Aufruf gibt "False" zurück,␣
 ↪wie gedacht, weil der String nicht in der Datei steht.
print(usernameTaken("beatrice.beispiel"))  # Dieser Aufruf jedoch gibt "False"␣
 ↪zurück, obwohl der String in der Datei steht. Warum?!?!
```

```
True
False
False
```

Ihre Beschreibung und Erläuterungen hier! (Doppelklick zum Editieren, Shift + Enter um die Zelle auszuführen)