Kleines Python-Handbuch zum STUTS-Workshop 2017

Fachverein Computerlinguistik UZH

May 15, 2017

1 Die Grundlagen

Python ist eine sogenannte 'objektorientierte' Programmiersprache. Das bedeutet, dass alles, was in unserem Programm erstellt wird, als virtuelles Objekt gespeichert wird. Verrechnen wir zum Beispiel zwei Zahlen miteinander, erstellen wir das Resultat als ein neues Objekt.
Python liest ausserdem den Code immer von oben nach unten. Es wird eine Zeile nach der anderen ausgeführt. Wird zum Beispiel auf Zeile 3 eine Variabel aufgerufen, die vorher noch nie vorkam (noch nicht initialisiert wurde), dann wird ein Fehler ausgegeben.
Ausserdem berücksichtigt Python beim Lesen des Codes die Einschübe (durch Tabulator oder vier Leerschläge). Wozu diese Einschübe benötigt werden, wird später erklärt. Das Hashtag (#) wird zum Deklarieren von Kommentaren verwendet.

Dieses Handout kann natürlich nur einen kleinen Einblick in die Masse von Möglichkeiten bieten, welche in Python zu finden sind. Mehr Informationen findet ihr durch Google (im Besonderen als Fragen bei Stackoverflow) oder in der Python-Dokumentation.

1.1 Variablen zuweisen und einfache Operationen

Wie schon erwähnt, müssen Variablen immer initialisiert werden, d.h. wir weisen der Variabel einen Wert zu. Python ist hierbei so nett, und erkennt selbst, welchen Datentyp wir zuweisen möchten.
Mit folgender Codezeile weisen wir der Variable `eine_zahl` die Zahl 5 und `andere_zahl` die Zahl 4 zu:

```
1 eine_zahl = 5
2 andere_zahl = 4
```

Eine einfache Rechenoperation wäre nun, diese beiden Zahlen zu addieren, dazu addieren wir die beiden Variablen und speichern das Resultat unter `resultat` ab:

```
1 eine_zahl = 5
2 andere_zahl = 4
3 resultat = eine_zahl + andere zahl # 9
```

Dasselbe können wir auch mit Zeichenketten (Strings) machen:

```
1 ein_Wort = "Hallo"
2 anderes_Wort = "Welt"
3 resultat = ein_Wort + anderes_Wort # "HalloWelt"
4 besseres_Resultat = ein_Wort + " " + anderes_Wort # "Hallo Welt"
```

Im besseren Resultat wird ausserdem noch ein Leerschlag hinzugefügt. Variablen können auch einfach überschrieben werden, wenn sie später im Code nochmal belegt werden:

```
1 ein_Wort = "Hallo"
2 anderes_Wort = "Welt"
3 anderes_Wort = "Freund" # Die alte Zuweisung wird überschrieben
4 resultat = ein_Wort + anderes_Wort # "HalloFreund"
5 besseres_Resultat = ein_Wort + " " + anderes_Wort # "Hallo Freund"
```

1.2 If/Else-Statements

If/Else-Statements helfen uns, bestimmten Code nur dann ausführen zu lassen, wenn eine Bedingung erfüllt ist oder nicht:

```
1 eine_zahl = 5
2 if eine_zahl > 3:
3     ein_wort = "Die Zahl ist kleiner oder gleich 3!"
4 else:
5     ein_wort = "Die Zahl ist grösser als 3!"
```

Nun wird auch die Bedeutung von Einschüben klar, alles was nach einem If- bzw. Else-Statement eingeschoben ist, wird beim Eintreffen der Bedingung ausgeführt. In unserem Code-Beispiel wäre die Zahl grösser als drei, daher würde der Code auf Zeile 3 ausgeführt werden.

1.3 Schleifen

Python kennt zwei verschiedene Arten von Schleifen. Eine ist die While-Schleife, welche läuft, bis die angegebene Bedingung erfüllt ist:

```
1 eine_zahl = 0
2 while eine_zahl < 10:
3     print(eine_zahl) # Gibt den Inhalt der Variabel aus
4     eine_zahl += 1
```

Dieses Codebeispiel würde uns die Zahlen von 0 bis 9 eine nach der anderen auf der Kommandozeile ausgeben. Auch hier gilt wieder, dass alles eingeschobene ausgeführt wird, solange die Bedingung der Schleife erfüllt ist.
Der andere Typ von Schleife ist die For-Schleife. Diese besagt, dass für jedes Element in einem Iterator (wie eine Liste, dazu kommen wir später), etwas ausgeführt wird.

```
1 eine_liste = [0,1,2,3,4,5,6,7,8,9]
2 for zahl in eine_liste:
3     print(zahl)
```

Dieses Codebeispiel gibt genau dasselbe aus, wie unser vorheriges! Aber ihr könnt hier die Unterschiede der Schleifen erkennen. Hier noch ein kombiniertes Beispiel des bisher gelernten:

```
1 eine_liste = [0,1,2,3,4,5,6,7,8,9]
2 for zahl in eine_liste:
3     if zahl % 2 == 0: # Die Modulo-Operation gibt Divisionsreste zurück
4         print("Gerade Zahl")
5 I   else:
6         print("Ungerade Zahl")
```

2 Datentypen

In diesem Kapitel werden kurz die wichtigsten Datentypen in Python vorgestellt. Leider ist es nicht möglich im Detail auf alle einzugehen, konsultiere dazu am Besten die Python-Dokumentation:

2.1 String

Ein String ist ein Objekt, das jeweils eine Zeichenkette beinhaltet. Über einen String kann auch mit einer For-Schleife iteriert werden! Dann stellt jedes Zeichen ein Element dar.

```
ein_string = "Hallo Welt"
for zeichen in ein_string:
    print(zeichen)
 Ausgabe:
 H
 a
 l
 usw.
```

2.2 Integer und Floats

Integer sind Objekte, die ganze Zahlen enthalten, während Floats auch Zahlen mit Kommastellen enthalten können. Wir verwenden Integer wann immer möglich, da sie weniger Speicherplatz als Floats benötigen.

```
ein_integer = 5
ein_float = 3.7
```

2.3 Boolean

Booleans sind Objekte, die entweder den Wert 'True' oder 'False' enthalten. Sie können als Bedingungen z.B. bei While-Schleifen und If/Else-Statements zum Einsatz kommen.

```
ein_wahres_statement = True
ein_unwahres_statement = False
```

2.4 Listen, Tuple und Sets

Listen, Tuple und Sets sind Objekte, die eine Menge von anderen Objekten beinhalten. Listen sind geordnet. Sets sind nicht geordnet und können im Gegensatz zu Listen nur einzigartige Elemente enthalten. Tuple sind geordnet und im Gegensatz zu Listen 'unveränderlich'. Manchmal ist die Veränderlichkeit von Listen nicht gewollt, dann wandeln wir sie in Tuple um (Zum Beispiel, wenn wir die Liste als einen Schlüssel für ein Dictionary verwenden wollen).

```python
eine_liste = ["Hallo", 3, "Hallo"]
ein_set = set(eine_liste) # Beinhaltet nun "Hallo" und 3 je einmal
ein_tuple = tuple(eine_liste) # ("Hallo", 3, "Hallo")
anderes_tuple = ("Hallo", "Welt")
```

Im Folgenden werden einige Listen-Operationen vorgestellt:

```python
eine_liste = ["Hallo", "Welt"]
eine_liste.append("!") # ["Hallo", "Welt", "!"]
ein_element = eine_liste[1] # "Welt", der Index startet bei 0
ein_index = eine_liste.index("!") # 2
eine_liste.sort() # ["!", "Hallo", "Welt"]
```

2.5 Dictionary

Dictionaries sind Objekte, in denen Schlüssel-Wert-Paare von Objekten gespeichert werden. Der Schlüssel muss dabei unveränderlich sein (Listen können zB. keine Schlüssel sein). Dictionaries sind unsortiert. Wollen wir Dictionaries sortiert ausgeben, müssen wir sie zuerst in Listen umwandeln. Im Beispiel wird ein Dictionary mit Wortpaaren Deutsch-Englisch gefüllt.

```python
ein_dict = {}
ein_dict["Hallo"] = "Hello"
ein_dict["Welt"] = "World"
print(ein_dict) # {"Hallo":"Hello", "Welt":"World"}
eine_liste = ein_dict.items() # Umwandlung zur Liste
print(eine_liste) # [("Hallo", "Hello"), ("Welt", "World")]
```

Über ein Dictionary zu iterieren ist etwas speziell:

```python
ein_dict = {"Hallo":"Hello", "Welt":"World"}
for schluessel, wert in ein_dict.items():
    print("Deutsch: " + schluessel)
    print("Englisch: " + wert)
# Ausgabe:
# Deutsch: Hallo
# Englisch: Hello
# Deutsch: Welt
# Englisch: World
```

Wenn wir die Funktion .items() weglassen, iterieren wir nur über die Schlüssel, nicht über die ganzen Paare. In den Beispielen des Workshops wird ausserdem ein Counter()-Objekt verwendet. Dabei handelt sich um eine Unterklasse eines Dictionary, das aber nur Zahlen als Werte erlaubt und mehr Rechenoperationen ermöglicht. Es ist dadurch perfekt geeignet um z.B. die Anzahl Vorkommen jedes Worts in einem Dokument zu speichern.

3 Wichtige Operationen zur Textverarbeitung

3.1 Slicing

Durch Slicing erhalten wir nur bestimmte Teile eines Strings oder einer Liste. Wichtig ist zu beachten, dass Indizes bei Python immer bei 0 beginnen. Das erste Zeichen ist also beim Index 0, das zweite bei 1 und so weiter.

```
ein_string = "Hallo Welt"
print(ein_string[4]) # "o"
print(ein_string[:5]) # "Hello" (von Anfang bis Index 5)
print(ein_string[-1]) # "t"
eine_liste = ["Hallo", "Welt", "!"]
print(eine_liste[1]) # "Welt"
```

3.2 Strings trennen und zusammensetzen

Verschiedene Funktionen ermöglichen es uns, Strings an bestimmten Stellen zu trennen und als Listen zu speichern und sie später wieder zusammenzusetzen. Die beiden einfachsten sehen wir hier:

```
ein_string = "Hallo Welt! Das ist ein Satz!"
eine_liste = ein_string.split() # Trennt den Satz an Leerschlägen
print(eine_liste)
# ["Hallo", "Welt!", "Das", "ist", "ein", "Satz!"]
ein_neuer_string = " ".join(eine_liste) # zusammensetzen
print(ein_neuer_string)
# "Hallo Welt! Das ist ein Satz!"
```

3.3 Benutzereingabe

Wir können den Benutzer auch nach einer Eingabe fragen, die wir dann als String in einer Variabel speichern. Falls wir wiederholt Eingaben möchten, dann lohnt es sich, den Code in eine while-Schleife zu setzen.

```python
weiter = True
while weiter:
    ein_string = input("Gib einen Satz ein oder Q zum Verlassen: ")
    if ein_string == "Q":
        weiter = False # Beim nächsten Prüfen wird die Schleife gestoppt
    else:
        print(ein_string.split()) # Gibt die Eingabe als Liste zurück
```

3.4 Einlesen und Schreiben von Dateien

Eine Datei zum Lesen öffnen:

```python
infile = open("Demodatei.txt", mode="r")
inhalt = infile.read() # Speichert den Inhalt als String
for zeile in infile: # oder liest direkt Zeile für Zeile
    print(zeile) # Gibt alle Zeilen aus
infile.close()
```

Eine Datei zum Schreiben öffnen:

```python
outfile = open("Demodatei.txt", mode="w")
outfile.write("Hallo Welt!") # Schreibt eine Zeile
outfile.close()
```

4 Natural Language Toolkit (NLTK)

Das Natural Language Toolkit, kurz NLTK, bietet verschiedene Werkzeuge, die einem die Arbeit mit natürlicher Sprache mit Python vereinfachen. Darunter finden sich Tokenisierer, Tagger oder auch Klassifikatoren für maschinelles Lernen.
Links:
NLTK installieren
Das NLTK-Buch mit guten Tutorials und Übungen