Python Einführungskurs

Einführung in die Numerik
Sommersemester 2022

27. April 2022

1. Übung

Einführung und Basics

Was ist Python?

Eine der beliebtesten Programmiersprachen weltweit

- leicht zu erlernen
- *high-level*
- *interpreted language*, also direkt ausführbar (muss nicht erst kompiliert werden)
- Multiparadigmensprache
  - objektorientiert und strukturiert
  - funktional und aspektorientiert
- benannt nach der BBC show "Monty Python's Flying Circus"

Installation

- Linux/UNIX
  - normalerweise vorinstalliert (z.B. auf Ubuntu)
  - sonst über die Paketverwaltung
- Windows
  - unter `https://www.python.org/` → Downloads
  - Im Abschnitt Files die passende (32/64 Bit) Version herunterladen
  - Installationsdatei *python-3.7.3-webinstall.exe* ausführen
- zusätzlich wird ein Editor benötigt (kate, VS Code, ...)
  - Liste vieler Editoren: `https://wiki.python.org/moin/PythonEditors`

Linux-Terminal

1. Lasst euch alle Dateien und Verzeichnisse anzeigen: `ls`
2. Legt ein Verzeichnis an: `mkdir TestOrdner`
3. Benutzt abermals `ls`. Was hat sich geändert?
4. In das Verzeichnis wechseln: `cd TestOrdner`
5. Wieder `ls` verwenden. Beobachtung?
6. In das übergeordnete Verzeichnis zurück wechseln: `cd ..`
7. `ls`. Und jetzt?
8. Löscht das Verzeichnis: `rm -r TestOrdner`
9. `ls`. Der Ordner sollte verschwunden sein!
10. Erstellt einen Ordner der `PythonIntro` heißt und wechselt in diesen Ordner!

Passwort ändern

Passwort ändern und **merken**

1. Terminal öffnen
2. In die Kommandozeile

   `passwd`

   eintippen und <ENTER> drücken
3. Altes Passwort (xxxxxxxxxx) eintippen
4. Neues Passwort (yyyyyyyyyyyy) eingeben
5. Neues Passwort (yyyyyyyyyyyy) bestätigen

Terminal Cheatsheet

Befehle nachschlagen in cheatsheets, z. B.

- `https://cheatography.com/davechild/cheat-sheets/linux-command-line/`
- `https://www.guru99.com/linux-commands-cheat-sheet.html`

Python starten und schließen

- Starten von Python im Terminal:

  `python`

- Starten einer bestimmten Version von Python (z.B. Version 3):

  `python3`

- Beenden von Python:

  `quit()` oder `exit()` oder `Ctrl-D`

Python Interpreter

Interaktiver Modus

1. Python öffnen
2. in die Konsole tippen:

```
print("Hello world!")
```

Python Skripte

1. Editor öffnen
2. Datei `hello.py` erstellen
3. in hello.py schreiben:

```
print("Hello world!")
```

4. speichern
5. im Terminal eintippen: `python3 hello.py`
6. Ausgabe: Hello world!

Hilfe zu Python

Hilfe-Funktion

- Hilfe-Funktion in Python aufrufen:

  ```
  help()
  ```

  Schließen der Hilfe-Funktion mit: `quit` oder `Ctrl-D`

- Hilfe zu einem bestimmten Befehl/Funktion/Ausdruck (z.B. zum Befehl `print`):

  ```
  help('print')
  ```

  Schließen der Hilfe hier mit: `q`

Online-Hilfe

Online-Hilfe

- Tutorial:
  `https://docs.python.org/3/tutorial/`
- Tutorial:
  `https://www.tutorialspoint.com/python/`
- Bei Programmier-Fragen jeglicher Art sehr hilfreich: `https://stackexchange.com/`

Zahlenformate

- Logik (`bool`):
  - Beispiele: `True`, `False`
- Ganze Zahlen (`int`):
  - Beispiele: `0`, `1`, `12`, `-7`
  - beliebig große Zahlen darstellbar in reinem Python
- Reelle Zahlen (`float`):
  - Beispiele: `0.0`, `1.4`, `2.`, `-16.9875`, `1e5`, `1.e5`, `2.345E-2`
  - beliebige Genauigkeit (bis auf Maschinengenauigkeit)
- Komplexe Zahlen (`complex`):
  - Beispiele: `2+4j`, `1j`, `1.j`, `-7.5+13.45j`

Abfrage eines Objekt-Formats mit `type()` :

- `type(1)` , `type(1.0)` , `type(True)`

Rechenoperationen

- Grundoperationen: `+ - * /`
- Gruppierung: `()`
- Potenz: `**`
- Floor division: `//` (geteilt und gerundet gegen $-\infty$)
- Modulo: `%`

```
>>> 5/3
>>> 5//3
```

Variablen und Zuweisungen

Einer Variable einen Wert zuweisen:

```
>>> n = 1
```

Der Variable `n` wird der `int` Wert `1` zugewiesen

```
>>> width = 13.6
```

Der Variable `width` wird der `float` Wert `13.6` zugewiesen

Variablen löschen mit `del`

```
>>> s = 1
>>> del(s)
>>> s
```

gibt `NameError`

Multiple Assignment

Mehreren Variablen gleichzeitig Werte zuweisen (multiple assignement):

```
>>> n, width, length = 5, 13.6, 4.2
```

ist gleichbedeutend zu

```
>>> n = 5
>>> width = 13.6
>>> length = 4.2
```

Kombinierte Rechenoperationen

Rechenoperationen, die Variablen direkt verändern ("in-place")

```
x += 2
x -= 2
x *= 2
x /= 2
x //= 2
x **= 2
```

```
x = x + 2
x -= x-2
x = x*2
x = x/2
x = x//2
x = x**2
```

Keine Allokation von neuem Speicherplatz!

Kommentare

Kommentare beginnen mit einem `\#` und werden vom Python-Interpreter nicht ausgeführt:

```
# Erstmal ein Kommentar zu Anfang
x = 1 # Warum tun wir das?
""" Hier schreiben
  wir einen Kommentar
  über mehrere Zeilen
"""
```

ist gleichbedeutend zu

```
x = 1
```

Strings 1

- Texte werden in Anführungszeichen geschrieben: `""` oder `''`

```
string = "text"
```

```
string = 'text'
```

- Alles innerhalb des ersten und letzten Anführungszeichens wird als Text interpretiert. Beispiel:

```
>>> print("# Kein Kommentar'!")
```

Strings 2

- Strings zusammenführen mit `+`:

```
>>> s1, s2 = 'text', 'message'
>>> print(s1+s2)
```

- Viele weitere Operationen, siehe z. B.
  - `https://www.codecademy.com/learn/learn-python-3/modules/learn-python3-strings/cheatsheet`

Listen 1

- Eine Liste erstellen mit `[]`

```
squares = [1, 4, 9, 16, 25]
```

- Auf Elemente zugreifen: Nummerierung von 0 bis n-1, oder rückwärts von -1 bis -n

```
>>> squares[0] # = 0
>>> squares[4] # = 25
>>> squares[-1] # = 25
>>> squares[-5] # 16
```

Slicing

Auf Listenabschnitte zugreifen

```
>>> squares[2:4] # [9,16]
>>> squares[3:4] # [16]
>>> squares[2:] # [9,16,25]
>>> squares[:4] # [1,4,9,16]
```

Listen 2

Werte zuweisen

```
>>> squares[1] = 100
>>> squares
[1, 100, 9, 16, 25]
```

```
>>> squares[1] = []
>>> squares
[1, [], 9, 16, 25]
```

Listenabschnitte zuweisen

```
>>> squares[0:3] = [100, 200, 300] # [100, 200
```

```
>>> squares[0:3] = []
>>> squares # [16, 25]
```

Liste von Listen

Liste von Listen

```
>>> A = [[1, 2, 3], [5, 7]]
```

```
>>> A[0] # [1, 2, 3]
>>> A[0][2] # 3
```

Listen 3

- Liste erweitern mit `+` und `+=`:

```
>>> liste = [0, 1, 2] + [42, 43, 44]
>>> liste += [42]
```

oder mit der Methode `append()`:

```
>>> liste = [0, 1, 2]
>>> liste.append(42)
>>> liste # [0,1,2,42]
```

Tuple

Erstellen eines `tuple` mit `()`

```
>>> t1 = (1,2,3,4)
>>> t1 # (1, 2, 3, 4)
>>> t1[1] # 2
>>> t1[0:3] # (1,2,3)
>>> t1[1] = 3 # TypeError
```

Tuples sind nicht veränderbar!

Tuple vs. List

Tuple

- sind nicht veränderbar, nachdem sie erstellt wurden
- nur das Auslesen ist erlaubt
- benötigen weniger Speicherplatz als List

List

- sind veränderbar
- können ausgelesen, modifiziert, erweitert, verkleinert werden
- benötigen mehr Speicherplatz als Tuple

dictionaries

key-value-Paare:

```
>>> d = {'name':'Max','Alter':35,1:[2,4,3]}
```

Zugriff Mittels key:

```
>>> d['name']
```

Dictionaries sind veränderbar:

```
>>> d['Alter'] = 36
```

Vergleichs-Operationen

Wertevergleich

```
==    # ist gleich
!=    # ist nicht gleich
<     # kleiner als
<=    # kleiner oder gleich
>     # groesser als
>=    # groesser oder gleich
```

Logische Operationen

- Logische Operatoren

```
>>> True and False
>>> (5>3) or (-1 == 0)
>>> True or False
>>> not True
```

- Mengenzugehörigkeit

```
>>> liste = [0,1,2]
>>> 0 in liste
```

Blöcke

Einrückung ist nicht egal

```
for x in range(100):
*TAB*if x == 2:
*TAB**TAB*print("foo")
*TAB**TAB*break
*TAB*print(x)
```

- 4 *Leerzeichen*
- keine Tabs benutzen
- vor allem nicht Tabs und Leerzeichen mischen!

if-else-Anweisung

```
if condition:
*TAB*statement
elif condition:
*TAB*statement
else:
*TAB*statement
```

if-else Beispiel

```
>>> x = 0.5
>>> if x > 0:
...     print(x, '> 0')
... elif x < -1:
...     print(x, '< -1')
... else:
...     print(x, 'in [-1,1]')
```

while Schleife

```
while condition:
*TAB*statement
```

Beispiel:

```
>>> a = 0
>>> while a < 3:
...     a += 1
...     print(a)
```

for Schleife

```
for target_list in expression_list:
*TAB*statement
```

Beispiel:

```
>>> # Laenge von strings messen:
... words = ['cat', 'window', 'foo']
>>> for w in words:
...     print(w, len(w))
```

List Comprehension

expressionlist kann sein:

- liste
- tuple
- dictionary
- `range(n)`

Auch möglich:

```
>>> y = [x**2 for x in range(4)]
```

break Statement

- `break`: Unterbricht `for`- oder `while`-Schleifen

```
>>> for x in range(100):
...         if x == 2:
...             break
...         print(x)
```

continue statement

- `continue`: Fortfahren mit nächster Iteration einer `for`- oder `while`-Schleife

```
>>> for x in range(4):
...     if x == 2:
...         continue
...     print(x)
```

- `pass`: Macht nichts

```
>>> x = 2
>>> if x>0:
...     pass
```