Würgeschlange 3 - Lesson 1

Tobias Maschek, Viktor Reusch

mit Materialien von Felix Döring, Felix Wittwer https://github.com/fsr/python-lessons

29. Oktober 2019

Python-Kurs

Gliederung

1. Python - Auch nur eine Sprache
2. Python Programmieren
3. Python Interactive Shell
4. Types

Python - Auch nur eine Sprache

Python-Interpreter

Wir arbeiten mit *Python 3.7* oder höher.
Nicht mit Python 2.7!

Windows Download über `https://www.python.org/downloads/`

Debian-Derivate `apt-get install python3.7`

Scriptcharakter

- Python Code wird nicht kompiliert, sondern beim importieren in Python Bytecode übersetzt
- Bytecode wird auf einer VM ausgeführt
- Kein Memory Management nötig, alles sind Referenzen
- Syntaxerror wird beim Importieren geworfen
- Andere Fehler findet man erst, wenn die betreffende Zeile ausgeführt wird.

Programmierpradigmen

- Python ist vor allem eine imperative und objektorientierte Sprache
- reine Funktionen und Variablen können auf oberster Ebene definiert werden
- Variablen, Klassen und Funktionen sind ab der Ebene sichtbar, in der sie eingeführt werden

Python Programmieren

Python-Code schreiben

Editor simpel und schnell

- Atom
- Notepad++

IDEs empfohlen (benutzen wir im Kurs)

- PyCharm (free + professional für Studenten)
- Eclipse mit PyDev

Ausführen von Python-Skripten

1. Schreiben seines Python-Codes in seinem Lieblings-Editor
2. Speichern des Codes als eine *.py*-Datei (z. B.: `my.py`)
3. Ausführen seiner Python-Datei im terminal

   Windows In der *CMD* mit `python my.py`

   Debian-Derivate In seiner Lieblings-Shell mit `python3.7 my.py`

Dokumentation

Erklärungen zu Tyen, Klassen und Funktionen der Standard Library:

offizielle Python-Dokumentation: `https://docs.python.org/3/`

print*(*objects, sep=' ', end='', file=sys.stdout, flush=False)*

Print objects to the text stream file, separated by sep and followed by end. sep, end, file and flush, if present, must be given as keyword arguments.

...

Python Interactive Shell

Starten

Windows In der *CMD* mit `python` bzw. über die Windows-Suche

Debian-Derivate In der Shell `python3.7`

- Evaluiert eingegebene Python-Code-Zeilen
- Erlaubt zeilenweises Abarbeiten von Python-Skripten
- Das Ergebnis der evaluierten Zeile ausgegeben
- Vorheriges Ergebnis kann über `_` referenziert werden

Starten

Windows In der *CMD* mit `python` bzw. über die Windows-Suche

Debian-Derivate In der Shell `python3.7`

- Evaluiert eingegebene Python-Code-Zeilen
- Erlaubt zeilenweises Abarbeiten von Python-Skripten
- Das Ergebnis der evaluierten Zeile ausgegeben
- Vorheriges Ergebnis kann über `_` referenziert werden

```
>>> name = "Alice"
>>> name
'Alice'
>>> print(_ + "&Bob")
Alice&Bob
```

How to exit Python Interactive Shell

Alles Eingegebene wird verworfen (**keine Speichern!**)

```
>>> abhaun
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'abhaun' is not defined
>>> exit
Use exit() or Ctrl-Z plus Return to exit
>>> quit
Use quit() or Ctrl-Z plus Return to exit
>>> exit()
```

„Hello World“

```
>>> print("Hello World")
Hello World
>>> print('Hello World')
Hello World
```

Variablen

- Besitzen einen festen Namen
- Besitzen eine feste Lebensspanne (*scope*)
- Ihnen kann ein Objekt zugewiesen werden
- Referenzieren dann ein Objekt im Speicher z.,B.: eine Zeichenkette, eine Zahl etc.
- Variablen der Interactive Shell sind global d. h. werden nur ungültig, wenn gelöscht oder die Shell geschlossen wird

```
>>> age = 20
>>> age
20
>>> age = age * 2 + 1
>>> age
41
>>> temp = age
>>> temp
41
>>> del age
>>> age
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'age' is not defined
>>> temp
41
```

Simple Built-in Types

Name	Funktion
`int`	Ganzzahl "beliebiger" Größe (`100`)
`float`	Kommazahl "beliebiger" Größe (`-0.432`)
`bool`	Wahrheitswert (`True`, `False`)
`str`	Zeichenketten (`"Hallo Welt"`, `'also valid'`)

Operatoren

Name	Funktion
`+, -, *, /`	Addition, Subtraktion, Multiplikation, Division
`**, //, %`	Potentieren, ganzzahlige Division, Rest
`not, or, and`	Boolsches Nicht, Boolsches Oder, Boolsches Und
`<, <=, >, >=, !=, ==`	Vergleichsoperatioren (`!=` bedeutet $\neq$)

```
>>> 8 / 3
2.6666666666666665
>>> 8 // 3
2
>>> not (True or False)
False
>>> 55 <= 76.3
True
```

Weitere Funktionen

Name	Funktion
`print()`	Ausgebe einer Zeichenkette
`type()`	Bestimmeung des Types einer Variable, Funktion oder Klasse
`abs()`	Absolut-Wert einer Zahl

```
>>> print("Hello World")
Hello World
>>> type("Hello World")
<class 'str'>
>>> abs(-1.5)
1.5
>>> type(abs)
<class 'builtin_function_or_method'>
```

Strings

```
>>> str1 = "This is "
>>> str1
'This is '
>>> str2 = 'a String.'
>>> str2
'a String.'
>>> str1 + str2
'This is a String.'
>>> str3 = ''
>>> str3
''
```

Strings - Jetzt mit Esc⊠ping

```
>>> no_esc = 'New \n line'
>>> no_esc
'New \n line'
>>> print(no_esc)
New
 line
```

Strings - ganz\n roh

```
>>> raw_str = r'No new \n line'
>>> raw_str
'No new \\n line'
>>> print(raw_str)
No new \n line
```

Strings - Schöner den je

```
>>> print('{} | {} | {}'.format('string', 5, -1.8))
string | 5 | -1.8
>>> print('{2} | {1} | {0}'.format('string', 5, -1.8))
-1.8 | 5 | string
>>> age = 20
>>> print(f'Alice is {age} years old.')
Alice is 20 years old.
```

Nützlich: `https://pyformat.info/`

Types

Types - None

```
>>> var = None
>>> type(var)
<class 'NoneType'>
>>> var2 = ''
>>> type(var2)
<class 'str'>
>>> var3 = 0
>>> type(var3)
<class 'int'>


>>> var is None
True
>>> var2 is None or var3 is None
False
```

Types - (Tuple, Tuple)

```
>>> atuple = (1, 'alice', 'bob')
>>> atuple[1]
'alice'

>>> ntuple = ('charlie', 'dave', 'eve', 'oscar', 'victor')
>>> ntuple[-1]
'victor'
>>> ntuple[2:5]
('eve', 'oscar', 'victor')
>>> ntuple[0] = 'ted'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment

>>> tuple_comb = atuple + ntuple
>>> tuple_comb
(1, 'alice', 'bob', 'charlie', 'dave', 'eve', 'oscar', 'victor')
```

Types - [List, List, List, ...]

```
>>> list = [0, 2, 3, 'string']
>>> list [2]
3
>>> len(list)
4
>>> list.append(4)
>>> list
[0, 2, 3, 'string', 4]
>>> list.insert(1, '1')
>>> list
[0, '1', 2, 3, 'string', 4]
>>> list.remove(0)
>>> list
['1', 2, 3, 'string', 4]
>>> list.remove('string')
>>> list
['1', 2, 3, 4]
```