Einführung in Python

Version 2.7

Version 2.7

- Warum nicht Python 3?
- Stand Juni 2016:
  - Red Hat Enterprise Linux 7 --> Python 2.7
  - Ubuntu 16.04 --> Python 2.7
  - SuSE Enterprise Linux 12 --> Python ???
  - MacOSX 10.11.5 (El Capitan) --> Python 2.7

Python ...

- ... ist eine interpretierte Sprache (Source -> Bytecode -> Interpreter)
- ... ist objektorientiert
- ... benutzt dynamische Typisierung
- ... erzwingt lesbar(er)en Quellcode

Documentation

- Hervorragende Dokumentation:
  https://docs.python.org/2/

- Eingebaute Funktionen:
  https://docs.python.org/2/library/functions.html

Pydoc

- "man" für Python-Module

```
pydoc <modulname>
```

Pythonscript erstellen

- Shebang. Erste Zeile des Scripts enthält den Verweis auf den zu verwendenden Interpreter:

```
#!/usr/bin/env python
```

- In der zweiten Zeile das Encoding festlegen:

```
# -*- coding: utf-8 -*-
```

- Script muss nachher ausführbar gemacht werden:

```
chmod 755 meinscript.py
```

Pythonscript erstellen

- Eine Anweisung pro Zeile
- Einzeiliger Kommentar

```
# Dies ist ein Kommentar
```

- Mehrzeilige Kommentare

```
'''
   Dies ist
   ein Kommentar
'''
```

Pythonscript erstellen

- Anweisungsblöcke müssen eingerückt werden

```
if a == b:
    print "ist gleich b"
    if a == c:
        print "ist auch gleich c"
print "ende"
```

- Innerhalb eines Blocks muss die Einrückung gleich sein.

- Im Styleguide werden 4 Spaces empfohlen.

Style Guide

- Die Problemlösung steht an erster Stelle
- Sourcecode darf aber auch schön und wartbar sein.

- PEP 8 -- Style Guide for Python Code: https://www.python.org/dev/peps/pep-0008/
- Überprüfung mit `pylint` https://www.pylint.org/

Variablen

- Namen dürfen nicht reservierten Wörtern entsprechen
- Namen dürfen nur aus Buchstaben, Zahlen und _ bestehen. Umlaute und Sonderzeichen sind nicht erlaubt.
- Groß- und Kleinschreibung wird unterschieden
- Namen müssen immer mit einem Buchstaben beginnen
- Es gibt keine Typendeklaration

Variablen

- Beispiele:
  - `foo = 42`
  - `foo123bar = "Hallo Welt"`
  - `FOOBAR=23`
  - ~~`1foo = 1`~~
  - ~~`lösung = 7`~~
  - ~~`else = 73`~~

Reservierte Wörter

- `and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, yield`

Basisdatentypen
Numerische Typen

- Ganzzahlen:

```
foobar = 1
```

- Fließkommazahlen:

```
foobar = 1.0
```

- Komplexe Zahlen:

```
foobar = 1j
```

Basisdatentypen
Sequenzielle Typen

- Zeichenketten:

```
foobar = 'Hallo Welt'
```

- Listen:

```
foobar = [ "foo", "bar", 23 ]
```

- Tupel:

```
foobar = ("Manfred", "Muster", 4242, False)
```

Basisdatentypen
Assoziative Typen

- Wörterbuch (dictionary):

```
foobar = { "key1": "wert1", "key3":
( "foo", "bar", 73)}
```

Basisdatentypen
Mengen

- Set:

```
foobar = set(["apfel", "birne", "orange",
"birne", "ananas"])
```

Kontrollstrukturen
Fallunterscheidung

```
if a == b:
    print "a gleich b"
elif a == c:
    print "a gleich c"
else:
    print "a ungleich b, a ungleich c"
```

Kontrollstrukturen
While-Schleife

```
a = 0
while a < 10:
    a = a + 1
    if a == 3:
        continue
    if a == 5:
        break
    print a
else:
    print "a war nie kleiner als 3"
```

Kontrollstrukturen
For-Schleife

```
a = [ 1, 2, 3, "foo", True ]
for element in a:
    if element == "foo":
        continue
    print element
    if element == False:
        break
for i in range(1, 10):
    print i
```

Exceptions

```
try:
    print 5 / 0
except Exception as e
    print e
```

Funktionen

```
def foo(parameter1, parameter2):
    pass

def bar(parameter1, parameter2="Blah"):
    print parameter1
    print parameter2

foo(47,11)

bar(47, "baz")

bar(47)
```

Klassen

```
class FooBar():
    def __init__(self):
        # Konstruktor
        self.wert = 8
    def __del__(self):
        # Destruktor
        pass
    def meine_funktion(self, param1):
        print param1
    def wert(self):
        return self.wert

FOO = FooBar()

FOO.meine_funktion('blahblah')

print FOO.wert()
```

Modules

- Module stellen Klassen und/oder Funktionen bereit (Library)
- Module sind einfache Pythonscripte
- Sie werden über den Befehl `import` im Script verfügbar gemacht:
  - `import datetime`
  - `from datetime import date`
- Werden über Suchpfad gefunden (`PYTHONPATH`, `sys.path`)
- dir() zeigt an, welche Namen in einem Modul definiert sind

Packages

- Komplexere Bibliotheken mit Python Funktionen und Klassen kann man in Packages zusammenfassen

- Realisiert über eine spezielle Verzeichnisstruktur

```
meinpackage/
    __init__.py
    foo/
        __init__.py
        tolleklasse.py
    bar/
        __init__.py
        geilefunktionen.py
```

- Ebenfalls über den Suchpfad aufgelöst

Dokumentation erstellen

- Sourcecode mit docstrings versehen (Kommentare unter `class ...` oder def ...)

```
pydoc ./meinescript.py
```

Testdriven development pyunit

- http://pyunit.sourceforge.net/
- https://wiki.python.org/moin/PyUnit

Beispiel: Dateien schreiben/ lesen

```
import os

# Dateien schreiben
meinedatei = open('foobar.txt', 'w')

for i in range(1, 10):
    meinedatei.write('Zeile %d\n' % i)

meinedatei.close()

# Dateien zeilenweise lesen
meinedatei = open('foobar.txt', 'r')
for zeile in meinedatei:
    print zeile,
meinedatei.close()

# Datei löschen
os.remove('foobar.txt')
```

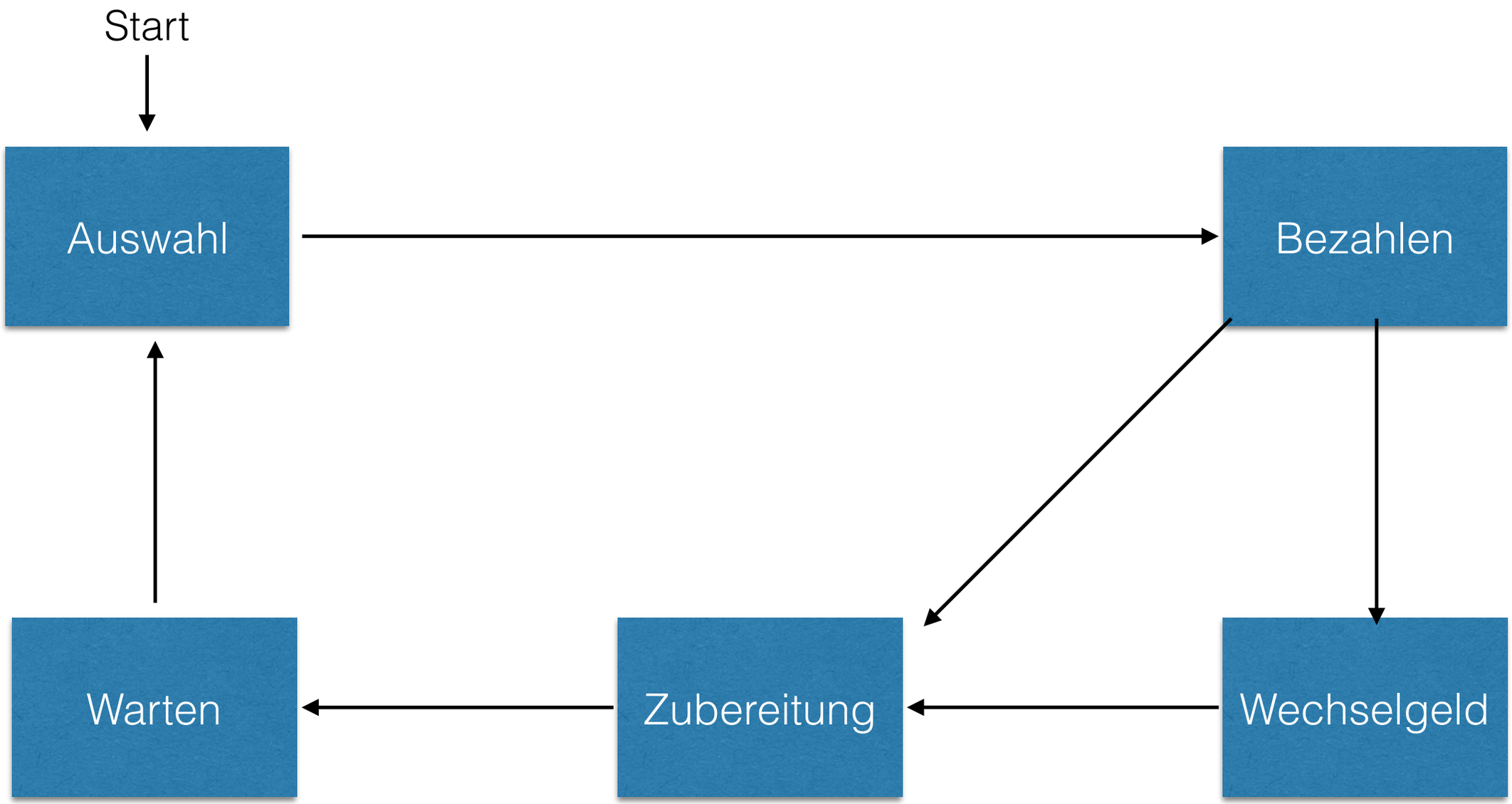
Beispiel:
Datum und Zeit

```
import datetime

# Aktuelles Datum und Zeit setzen
jetzt = datetime.datetime.now()

# Ausgabe in Standarddarstellung
print jetzt.strftime('%c')

# Ausgabe im ISO 8601 Format
print jetzt.isoformat()

# Ausgabe in eigenem Format
print jetzt.strftime('%Y%m%d-%H%M')

# Datum vorgeben
irgendwann = datetime.date(2017,1,1)
print irgendwann.strftime('%c')
```

Programm Kaffeeautomat

- Simulation eines einfachen Kaffeeautomaten

Statemachine Kaffeeautomat



Links

- http://python.haas.homelinux.net/
- http://python.swaroopch.com/control_flow.html