Vortragender: Mark Hofstetter

PYTHON

Python 3 – Warum?

- Moderne gut strukturierte Sprache
- Klar, mit dem Ansatz, dass es nur EINEN sinnvollen weg für ein Problem geben sollte
- Interpretierte Scriptsprache, aber trotzdem schnell
- Passend für Programme jeder Größenordnung

Scriptsprachen Übersicht

Aug 2017	Aug 2016	Change	Programming Language	Ratings	Change
1	1		Java	12.961%	-6.05%
2	2		C	6.477%	-4.83%
3	3		C++	5.550%	-0.25%
4	4		C#	4.195%	-0.71%
5	5		Python	3.692%	-0.71%
6	8	▲	Visual Basic .NET	2.569%	+0.05%
7	6	▼	PHP	2.293%	-0.88%
8	7	▼	JavaScript	2.098%	-0.61%
9	9		Perl	1.995%	-0.52%
10	12	▲	Ruby	1.965%	-0.31%

https://www.tiobe.com/tiobe-index/ - 11.8.2017

Anwendungsgebiete

- Sehr gute Einsteigersprache
- Schnelles scripting:
  - Admin
  - OS Organisation
- Statistik/Mathematik/Visualisierung
  - Pandas - http://pandas.pydata.org/
  - NumPy - http://www.numpy.org/
  - https://matplotlib.org/

Anwendungsgebiete 2

- Programmatische Interfaces
  - Deep Learning - https://www.tensorflow.org/
  - …
- Rasberry PI
  - https://www.raspberrypi.org/documentation/usage/python/
- Serverseitige Web Entwicklung
  - Django - https://www.djangoproject.com/
  - Flask - http://flask.pocoo.org/

Was soll/muss ich am Ende des Kurses können

KURSZIELE

Lernergebnisse

- Die Studierenden beherrschen die Programmiersprache Python auf einem solidem mittleren Niveau, dass heißt sie können selbständig neue Module (wie zB Netzwerkverschlüsselung oder ähnliches) sicher benutzen, selbstständig kleinere Projekte erstellen und in größeren Projekten Änderungen am Code vornehmen.
- Theoretisches Verständnis der Kursinhalte über das Beherrschen der Programmiersprache hinaus, zB Begriffe aus der Objektorientierung etc
- Softwareentwicklung Hintergrundwissen:
  - Versionierung (mit git)
  - TDD Test Driven Developement
  - Pair Programming, Code Review

Vorkenntnisse

- Vorkenntnisse Python
- OO Vorkenntnisse
- Git/gitlab/github – wer hat schon einen account
- Test Driven Developement

Aufgaben

- Viele der Beispiele sind aus der Praxis
- Bitte um Vorschläge aus Ihrem Umfeld

PYTHON GRUNDLAGEN

Kompilierte vs Interpretierte Sprachen

- Kompilierte Sprachen erzeugen Maschinencode der nur auf einer bestimmten Plattform lauffähig ist, zB C, C++ etc
- Interpretierte Sprachen lesen den Sourcecode (Quelltext) jedes mal neu ein und interpretieren ihn dann, zB Perl, Python, Ruby, PHP
- Die Realität ist deutlich komplizierter mit allen möglichen Zwischenlösungen wie Just In Time Compiler, Bytecode, etc

Variablen

- Haben einen Typ
- Müssen vor der ersten Verwendung initalisiert werden (nicht deklariert)

```
a = 2
z = "2"
b = "Hallo Welt"
# a = "Zwei"
# print(type(a))

c = str(a)
print(a, b) ## einfache Ausgabe OHNE
Verkettung
# print(a + b) ## operand auf
unterschiedliche Typen funktioniert nicht
print(str(a) + b) # oder
print(c + b)

# t = "Falsch"
t = False
t1 = None
t2 = None
a = 2
```

```
a = 2
z = "2"
c = int(z)
print(hex(id(a)))
print(hex(id(c)))

c = 4
print(hex(id(c)))
```

- Variablen gleichen Werts sind nur als Referenzen abgelegt
- Spart Speicher
- Verhält sich Transparent

Indentation

- https://en.wikipedia.org/wiki/Indent_style
- https://www.youtube.com/watch?v=SsoOG6ZeyUI
- https://www.python.org/dev/peps/pep-0008/#tabs-or-spaces
  - Spaces are the preferred indentation method.
  - Tabs should be used solely to remain consistent with code that is already indented with tabs.
  - Python 3 disallows mixing the use of tabs and spaces for indentation.

- Einrückungen haben eine syntaktische Bedeutung in Python!!!

Kontrollstrukturen

```
zahl = int(input('bitte eine Zahl eingeben: '))

if zahl == 3:
    print("super drei!")
    print("gut das du 3 geschrieben hast")
elif zahl > 3 and zahl <= 6:
    print("mehr als 3")
    if zahl != 5:
        print("zahl muss 4 oder 6 sein")
elif zahl > 6:
    print("mehr als 6")
else:
    pass

print("Ende")
```

- If – elif – else
- Kein switch/case

Schleifen

```
for i in range(0,10,2):
    print(i)
print(i)

j = 0
while j < 10:
    print(j)
    # j = j + 1
    j += 1


k = 0
print("repeat until")
while True:
    k += 1
    if k == 4:
        continue
    print(k)
    if k > 10:
        break
```

- Es gibt die üblichen Schleifenstrukturen
- Die for-Schleife braucht immer ein Iterator Object (siehe später) oder einen Generator
- Es gibt keine Schleifenlabels wie in perl oder php
- break und continue gelten also immer für die „inner loop“

User Input – Rückgabetyp

- Built-in Functions: https://docs.python.org/3/library/functions.html
- zB https://docs.python.org/3/library/functions.html#input
- Funktionen haben definierte Rückgabewerte

```
user_input = input("Bitte Zahl eingeben: ")
user_input = int(user_input)
```

Fehlerbehandlung

```
try:
    user_input = input("Bitte Zahl eingeben: ")
    user_input = int(user_input)
except ValueError:
    print(user_input, " ist keine Zahl")

# Ausbauen mit Division by Zero
```

- https://docs.python.org/3/library/exceptions.html

String Manipulation

```
text = 'Python ist toll'

print(text + text)

print((text + ' ') * 4 )

print(text)
print('=' * len(text))

print(text.lower())
print(text.upper())


# siehe später
# for b in text:
#    print(b)

print(text.count('ll'))
print(text.replace('Python', 'perl'))
```

String Manipulation

```
'''
der benutzer soll eine suchstring eingeben
und es soll ueberprueft werden ob der in in text
enthalten ist
===
+ gross/kleinschreibung ignorieren

'''

searchstring = input("Suchtext eingeben bitte: ")
if searchstring.strip().lower() in text.lower():
    print(searchstring, "enthalten")
else:
    print(searchstring, "nicht gefunden")
```

Format String

```
a = 11
b = 7

print(a,"geteilt durch",b,"ergibt",str(a/b)+"!")

# pythonic
print("{0:d} geteilt durch {1:d} (nochmals {1:d}) ergibt
{2:0.3f}".format(a,b,a/b))

# C style - printf
print("%d geteilt durch %d ergibt %-12.3f=" % (a,b,a/b))
```

Listen

```
'''
list, array, field

liste, Feld
'''
#    0 1 2 3 4 5 6  7
f = [1,1,2,3,5,8,13,21]

print(f)
# der index einer liste beginnt bei 0
print(f[3])
# liefert das letzte Element
print(f[-1])
# list slices
print(f[3:6])
print(f[:3])
print(f[3:])
print(f[-2:])
```

Listen

```
print("zahl der Elemente", len(f))
print("letztes Element", f[len(f)-1])
f[7] = 23
f.append(45)
print(f)
r = f # kopiert nicht! sondern erstellt nur ein neue Variable auf
die bestehende referenz
r = list(f) # erstellt eine neue Liste mit den Werten der
Bestehenden
f.reverse() # oder sort etc
for element in r:
    print(element)
f.insert(0,'bla')
print(f)
f.insert(3, 23)
print(f)
i = f.remove(23) # geht auf Werte, bei nichtexistenz => fehler
print(i, f)
del(f[0])
l = [0] * 45
print(l)
```

Komplexe Listen

```
teilnehmer = [
    ['Martin',    1976, ['VS', 'Gym', 'Uni']],
    ['Harald',    1970],
    [Eva',        1973],
    ['Roland',    1998],
    ['Christoph', 1981],
    ['Roland',    1974],
]

teilnehmer.append([,Michala', 1975, ['VS', 'Gym', 'Uni']])
teilnehmer.sort()
```

Komplexe Listen

```
name = input("Name: ")
found = False
for tn in teilnehmer:
    # namen.append(tn[0])
    if tn[0] == name:
        # edu = 'n/a' if len(tn) < 3 else tn[2][-1]
        if len(tn) < 3:
            edu = 'n/a'
        else:
            edu = tn[2][-1]

        print("%s wurde %d geboren, hoechste Bildung %s" % (name, tn[1],
edu))
        found = True

if not found:
    print("Nichts gefunden")
```

Listenfunktionen

```
teilnehmer = [
    ['Eva', 1996],
    ['Sascha', 1995],
    ['Peter', 1980],
    ['Leo', 1966],
    ['Mark', 1975],
]

# list comprehension
namen = [ tn[0] for tn in teilnehmer ]

names, years = zip(*teilnehmer)
j = names.index(userinput)
print(years[j])

max_year = max(years)
i = years.index(max_year)
print(names[i])
```

Tuple

- Ähnlich wie Listen nur unveränderlich
- Werden deswegen gerne als Funktionsrückgabe/übergabewert verwendet

```
a = (1, 2, 3)

print(a)
print(a[:2])

red = (255,0,0)
red = list(red)
print(red)
red[0] = 249
red = tuple(red)
print(red)
```

Dictionary

```
import pprint

#     key           value
#     schlüssel     werte
teilnehmer = {
    'Martin':      {'year': 1976, 'height':
180, 'education': ['VS', 'Gym', 'Uni']},
    'Harald':      1970,
    'Gerald':      1973,
    'RolandW':      1998,
    'Christoph':   1981,
    'RolandS':      1974,
}

teilnehmer['Mark'] = 1975
# print(teilnehmer['Martin'])
pprint.pprint(teilnehmer)
```

- dict, dictionary
- hash, assoziativer array

Dictionary

```
import pprint

#     key           value
#     schlüssel     werte
teilnehmer = {
    'Martin':      {'year': 1976, 'height':
180, 'education': ['VS', 'Gym', 'Uni']},
    'Harald':      1970,
    'Gerald':      1973,
    'RolandW':      1998,
    'Christoph':   1981,
    'RolandS':      1974,
}

teilnehmer['Mark'] = 1975
# print(teilnehmer['Martin'])
pprint.pprint(teilnehmer)
```

- dict, dictionary
- hash, assoziativer array

Dictionary - Views

- zB Dictonary umkehren
- Nur auf keys/values zugreifen

```
eng2ger = {
    'dog': 'hund',
    'cat': 'katze',
    }
ger2eng = dict((v,k) for k,v in eng2ger.items())
ger2eng = dict( zip(eng2ger.values(), eng2ger.keys()) )
```

Komplexe Strukturen - APIs

```
import requests
import datetime
import pprint

url =
'http://samples.openweathermap.org/data/2.5/weather?q=London,uk&ap
pid=b1b15e88fa797225412429c1c50c122a1'


response = requests.get(url)
data = response.json()
pprint.pprint(data)

print("%.2f Grad C" % ( data['main']['temp']-273.15 ))

print(data['weather'][0]['main'])
```

Funktionen

- Werden mit „def“„forward declared“

```
b = 2

def addiere(a, b, c):
    summe = a + b + c
    b += 1
    print('aus fkt', b)
    return summe

print(addiere(1, b, 3))
print(b)
```

Funktionen

- Können beliebige Parameter übernehmen, benannt und unbenannt.
- Skalare Variablen werden „by value“ übernommen

```
def zinsen(kapital, laufzeit_jahre=10, zinsatz_prozent=3):
    return kapital * (1 + \
zinsatz_prozent/100)**laufzeit_jahre

print(zinsen(1000, laufzeit_jahre = 10,zinsatz_prozent= 10))
print(zinsen(1000))
```

Funktionen

- Es kann eine beliebige Zahl von Paramtern kann mit kwargs übergeben werden

```
def addiere_kw(**kwargs):
    sum = 0
    for a in kwargs.values():
        sum += a
    return sum

summe = addiere_kw(a=1, b=2, c=7)
print(summe)
```

Funktionen

- Lists & dicts werden „by reference“ übernommen

```
def sum_list(feld):
    feld.append(4)
    sum = 0
    for a in feld:
        sum += a
    return sum


feld_main = [1,2,3]
summe = sum_list(feld_main)
print(summe)
print(feld_main)
```

Generator

```
def my_range(low, high):
    current = low
    while current <= high:
        yield current
        current += 1

for c in my_range(2, 9):
    print(c)
```

- Die „yield“ liefert Wert zurück ohne zu „returnen“

Import

- Es gibt nur wenige python core Funktionen (built-in) (vergleiche php)
  - https://docs.python.org/3/library/functions.html
- Es gibt aber die „The Python Standard Library“
  - https://docs.python.org/3/library/index.html
- „import“ durchsucht
  - den Standard Pfad (zb C:\Users\mh\AppData\Local\Programs\Python\Python36-32\Lib)
  - Das Verzeichnis des ausgeführten scripts
  - Den/die Pfade der PYTHONPATH Umgebungsvariable

Import

```
import random
print(random.randint(1,9))

# oder

from random import randint
print(randint(1,9))

# oder

from random import randint as ri
print(ri(1,9))

# oder

from random import *
print(random())
```

- Verschiedene Möglichkeiten ein Modul bzw einzelne Funktionen daraus zu importieren

packages installieren - pip

Installing Packages

https://pypi.python.org/pypi

pip supports installing from PyPI, version control, local projects, and directly from distribution files.

The most common scenario is to install from PyPI using Requirement Specifiers

```
$ pip install SomePackage            # latest version
$ pip install SomePackage==1.0.4     # specific version
$ pip install 'SomePackage>=1.0.4'     # minimum version
```
For more information and examples, see the pip install reference.

Requirements Files
“Requirements files” are files containing a list of items to be installed using pip install like so:

```
pip install -r requirements.txt
```

Betriebssystempakete

root@debian:~# apt-cache search python3 | grep numpy
python3-numpy - Fast array facility to the Python 3 language

root@debian:~# apt-get install python3-numpy

Werden gepflegt und auf stabilem Stand gehalten, und ist bei größeren Paketen „machbarer“

zb:

http://caffe.berkeleyvision.org/installation.html

vs.

http://caffe.berkeleyvision.org/install_apt_debian.html

virtualenv

- https://www.dabapps.com/blog/introduction-to-pip-and-virtualenv-python/

virtualenv --python=python3 env
source env/bin/activate

plot

- pip install matplotlib

```
from matplotlib import pyplot as
plt

plt.plot([1,2,3,4])
plt.plot([2,3,4,5])

plt.show('foo.png')
```

configuration

```
import configparser
from pprint import pprint

config = configparser.ConfigParser()
config.read('woerterbuch.ini')
config.sections()

wb = config['english_to_german']
pprint(dict(wb))
```

```
[english_to_german]
dog=hund
cat=katze
mouse=maus
house=house
tree=baum
soup=suppe

[game]
number_of_tries=3
```

TEST DRIVEN DEVELOPMENT

pytest: helps you write better programs

- https://docs.pytest.org/
- Test Driven Developement
- Nur (!!!) mit Hilfe automatisierter Tests kann Softwareprojekte auf ein professionelles Niveau heben

pytest: helps you write better programs

- Aufgabe:

```
+ benutzer gibt „Wörter“ ein solange bis zur eingabe von 'ENDE'
+ wir führen mit wie oft welches Wort eingeben wurde
+ Ausgabe der „Statistik“
Beispieldurchlauf
 a b c b c a a a ENDE =>
a: 4
b: 2
c: 2
```

- testbar!!!

Lösung ohne Test

```
from pprint import pprint

word_count = dict()

while True:
    userinput = input('Bitte gib mir nur ein
Wort: ')
    if userinput.lower() == 'ende':
        break
    if userinput in word_count:
        word_count[userinput] += 1
    else:
        word_count[userinput] = 1

pprint(word_count)
```

- Mühsam
- Es besteht die Gefahr von „Regressions“

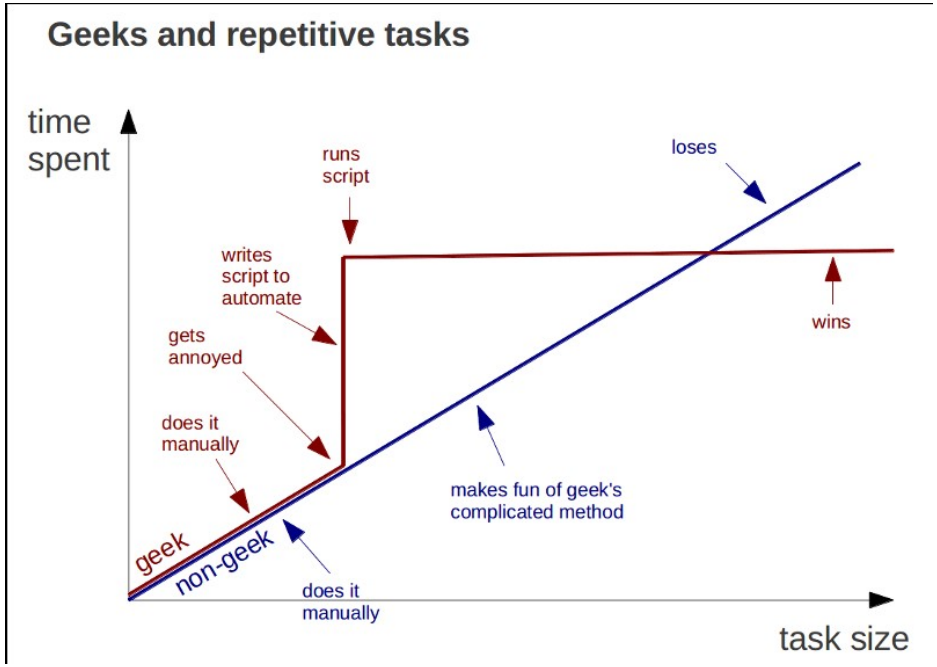
Test

```
import pytest
import word_count

def test_wc():
    assert word_count.count_words(['a', 'a', 'c',
'c', 'd']) == \
        {'a':2, 'c':2, 'd': 1}

    assert word_count.count_words([]) == \
        {}

def test_numbers():
    assert word_count.count_words([1,2,3,4,1,2,2])
== \
        {1:2,2:3,3:1,4:1}
```

Modul

```
def count_words(word_list):
    word_count = dict()
    for word in word_list:
        if word in word_count:
            word_count[word] += 1
        else:
            word_count[word] = 1
    return word_count

if __name__ == "__main__":
    word_list = list()
    while True:
        userinput = input('Bitte gib mir nur ein Wort: ')
        if userinput.lower() == 'ende':
            break
        word_list.append(userinput)

    print(count_words(word_list))
```

Warum



https://dev.to/s_anastasov/my-first-unit-test-c44

Regular Expressions

- Validieren
  - zB Ist ein Eingabestring eine Postleitzahl
    - A-123, A0123, A-2452 usw
- Extrahieren
  - zB Postleitzahl aus unstrukturierter Adresse
  - address = 'Schulgasse 234; 1090 gars am kamp, Stiege 27'
- Normalisieren
  - Vereinheitlichen von „multiplen Whitespaces“ zu jeweils einem Blank
  - „ Schulgasse 234; 1090 gars am kamp, Stiege 27“

Validieren

```
import re

# regex flag re.I

def check_plz(plz):
    matches = re.search(r'^A[ -]{0,1}(\d{4})$', plz)

    if matches:
        print("OK AT", matches[0], matches[1])
        return True
    else:
        print("nicht OK", plz)
        return False
```

Extrahieren

```
import re

address = 'Schulgasse 234; 1090 gars am kamp, Stiege
27'
data = re.search(r'(\d{4})', address)
print(data.group(1))
```

Normalisieren

```
address = ' Schulgasse    234;        1090    gars am
kamp, Stiege 27  '
address = address.strip()
address = re.sub(r'\s+', ' ', address)

print("[%s]" % (address))
```

Manueller Test

```
plz_to_check = {
    'A1234': True,
    'A-1239': True,
    'b1235': False,
    'A  5666': False,
    'A 666': True,
    }

def check_plz(plz):
    matches = re.search(r'^A[ -]{0,1}(\d{4})$', plz)
    if matches:
        return True
    else:
        return False

test_fail = False
for plz in plz_to_check:
    if check_plz(plz) == plz_to_check[plz]:
        pass
    else:
        test_fail = True
        print('Test NICHT OK !!!!')

if test_fail:
    print('es ist was schiefgegangen')
else:
    print('alles gut')
```

Datum & Zeit

```
import datetime
import time
from dateutil.relativedelta import relativedelta

print(datetime.datetime.now())

epoch = 1485789600

print(datetime.datetime.fromtimestamp(epoch))
#            jahr monat  tag stunde  minute sekunde
msek
birth_day = 1975,    2,  23,     4,       0,      0,
0,0,0

birth_day_dt = time.mktime(birth_day)
print(birth_day_dt)
```

Datum & Zeit

```
alter = time.time() - birth_day_dt
print(alter)
print(alter/(365.25 * 24 * 3600))

dt_birth_day = datetime.date(1975,2,23)
dt_today     = datetime.date.today()

diff = dt_today - dt_birth_day
print(diff.days)
rd_age = relativedelta(dt_today, dt_birth_day)
print(rd_age.years, rd_age.months, rd_age.days)

print(dt_today + relativedelta(years=+1))

print(datetime.date(1582,10,5) + relativedelta(days=+1))
```

DATEIHANDLING & BETRIEBSSYSTEMINTERAKTION

Command Line

```
import argparse

## komplexere Parameteruebergabe zB mit
## https://pypi.python.org/pypi/ConfigArgParse
parser = argparse.ArgumentParser()

parser.add_argument('-i', '--inputfile',
  action="store", dest="inputfile",
  help="inputfile to be parsed")
parser.add_argument('-d', '--debug',
  help="debug")

options = parser.parse_args()
print ('Input File', options.inputfile)
```

Dateien Lesen

```
error = False
output = []
try:
    with open(options.inputfile, "r") as file:
        #  lines = file.readlines()  # liest in eine liste
        i = 1
        for line in file:
            output_line = "%03d: %s" % (i, line.strip())
            print(output_line)
            output.append(output_line + "\n")
            i += 1
except FileNotFoundError:
    print("Datei nicht da")
    error = True
except Exception as error_msg:
    print(error_msg)
    error = True

if error:
    print("Exit on error")
    exit(1)
```

Dateien Schreiben

```
print("writing output")

with open('ln_' + options.inputfile, "w") as file:
    file.writelines(output)
```

csv -Dateien

```
import csv

# lesen
with open('daten.csv') as csvfile:
    data = list(csv.reader(csvfile, delimiter=';'))

# schreiben
with open('daten_clean.csv', 'w', newline='') as writecsv:
   writer = csv.writer(writecsv, delimiter=',')
```

os und shutil

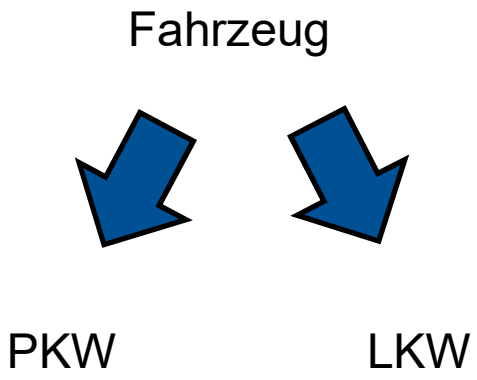
```
for filename in os.listdir(„dirname“):
  if not filename.endswith(".jpg"):
    continue

# oder rekursiv

for root, dirs, files in os.walk(paths['source']):


# etc
```

Möglichst wenig „händisch“ machen, dh keine „quoten“ und ähnliches, sondern IMMER zuerst nachsehen ob es entsprechenden Funktionen gibt

OBJEKTORIENTIERUNG I

Beispielklasse

- Abstrakte Parentclass



Nicht wirklich gutes Beispiel …

Objektorientierung – wann und warum

- Um vom allgemeinen ins Spezielle zu gehen
  - Datenbankbasisklasse – Eigenschaften die jede Tabelle hat (id, create_date, etc)
  - Einzelne Tabellen leiten sich davon ab
- Um Implementierungsdetails zu verbergen
  - Logging – stellt allgemeine Methoden zur Verfügung, dem Nutzer „kann egal sein“, wo/wie genau die Daten gespeichert werden.
  - Log->write() wird immer gleich aufgerufen egal ob in ein Logfile oder zb eine Datenbank geschrieben wird.

Send SMS

```
from Message import SMS

#message = Message()
#message.content = 'Hallo Welt'
#message.send()

# Objekt Instanzierung
sms = SMS(sender = 'Mark')
sms.content = 'Hallo alte SMS Welt'
sms.recipient = 'Alice'
print(sms.sender)
sms.send()
```

Message Class

```
from abc import ABC, abstractmethod

class Message(ABC):

    recipient = None
    sender = None
    __content = None

    @abstractmethod
    def __init__(self, sender = 'None'):
        print('init from parent')
        self.__content = None

    @property
    def content(self):
        return self.__content

    @content.setter
    def content(self, content):
        self.__content = content

    def send(self):
        print("logging from class:", self.__class__.__name__)
```

SMS Subclass

```
class SMS(Message):

    def __init__(self, sender):
        super(SMS, self).__init__()
        print('init')
        self.sender = sender

    @Message.content.setter
    def content(self, content):
        if (len(content) > 10):
            print("content zu gross")
        self.__content = content

    def send(self):
        # Message.send(self)
        super(SMS, self).send()
        print("sending sms:", self.recipient, self.__content)
```

DATENBANKEN

Verbinden

```
import sqlite3

conn = sqlite3.connect('students.db')

# mysql/MariaDB
import mysql.connector
cnx = mysql.connector.connect(user='scott',
database='employees')

# postgres
import psycopg2
try:
    conn = psycopg2.connect("dbname='template1'
user='dbuser' host='localhost' password='dbpass'")
except:
    print "I am unable to connect to the database"
```

Datenbank

- Minimales Beispielschema
- Kann mit sqlite erstellt werden

```
CREATE TABLE student (
        id INTEGER NOT NULL,
        name VARCHAR(50) NOT NULL,
        birth_year VARCHAR(50) NOT NULL,
        PRIMARY KEY (id)
);
```

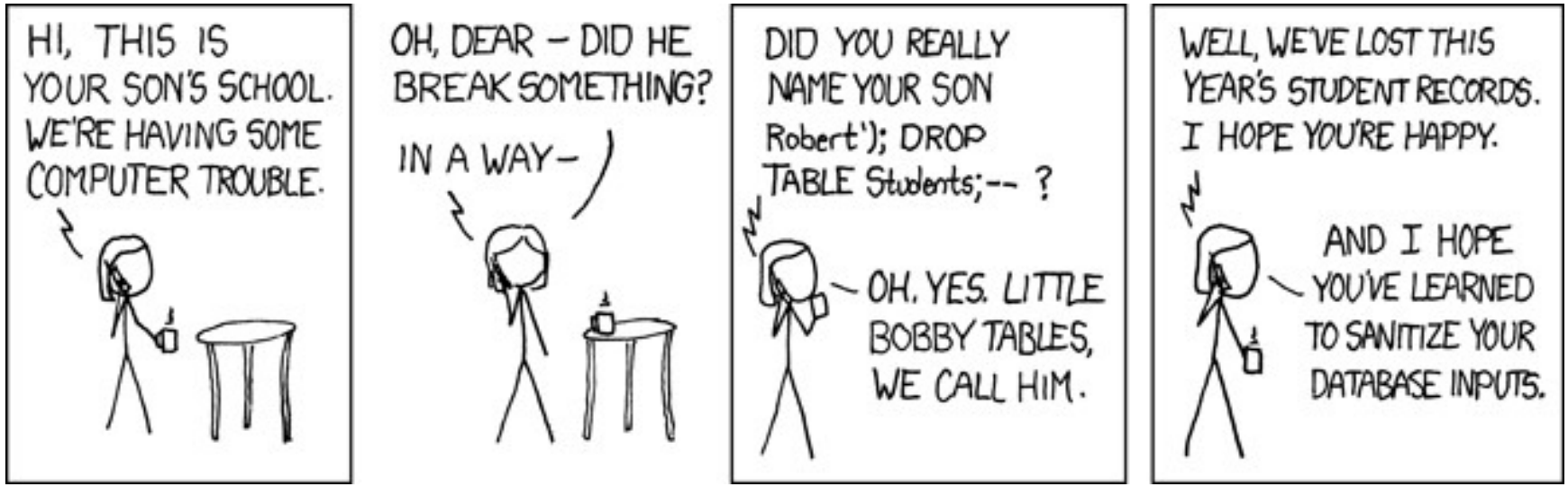
Query Strings

```
import sqlite3

conn = sqlite3.connect('students.db')

teilnehmer = {
    'Martin':     1976,
    'Harald':     1970,
    'Gerald':     1973,
    'RolandW':    1998,
    'Christoph':  1981,
    'RolandS':    1974,
}

for name, birth_year in teilnehmer.items():
    conn.execute("insert into student(name, birth_year) values (?,?)",
(name, birth_year))

conn.commit()
```

Abfragen

```
import sqlite3

conn = sqlite3.connect('students.db')

user_name = input('name: ')
birth_year =  input('birth year: ')

sql = 'SELECT * FROM student WHERE name ="' + user_name + '"
AND birth_year ="' + birth_year + '"'

cursor = conn.execute(sql)
for row in cursor:
    print(row)

conn.close()
```

xkcd



https://xkcd.com/327/
https://imgs.xkcd.com/comics/exploits_of_a_mom.png

Bind Parameter

```
import sqlite3

conn = sqlite3.connect('students.db')

user_name = input('name: ')
birth_year =  input('birth year: ')

cursor = conn.execute('SELECT * FROM student WHERE name = ?
AND birth_year = ?', (user_name, birth_year))
for row in cursor:
    print(row)

conn.close()
```

DECORATOR

Functional Decorators

```
import time

def timing_function(func):
    def wrapper():
        t1 = time.time()
        func()
        t2 = time.time()
        return "Time it took to run the function: " + str((t2 - t1)) +
"\n"
    return wrapper

@timing_function
def my_function():
    num_list = []
    time.sleep(1)
    for num in (range(0, 10000)):
        num_list.append(num)
    print("\nSum of all the numbers: " + str((sum(num_list))))

print(my_function())
```

Object-relational mapping

ORM

Warum

- Abstrahierung
- Um die ständige Wiederholung von CRUD Implementierungen zu vermeiden
- Um unabhängig(er) von der konkreten Datenbank zu sein
- „magic“

SQLalchemy - Schema

```
from sqlalchemy import Column, ForeignKey, Integer, String, Float
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
from sqlalchemy import create_engine

Base = declarative_base()

class Energy(Base):
    __tablename__ = 'energy'
    id = Column(Integer, primary_key=True)
    name = Column(String(250), nullable=False)

class Price(Base):
    __tablename__ = 'price'
    id = Column(Integer, primary_key=True)
    year = Column(Integer, nullable=False)
    price = Column(Float, nullable=False)
    energy_id = Column(Integer, ForeignKey('energy.id'))
    energy = relationship(Energy)

engine = create_engine('sqlite:///sqlalchemy_energy.db')

Base.metadata.create_all(engine)
```

SQLalchemy - Resultat

```
sqlite> .schema
CREATE TABLE energy (
        id INTEGER NOT NULL,
        name VARCHAR(250) NOT NULL,
        PRIMARY KEY (id)
);
CREATE TABLE price (
        id INTEGER NOT NULL,
        year INTEGER NOT NULL,
        price FLOAT NOT NULL,
        energy_id INTEGER,
        PRIMARY KEY (id),
        FOREIGN KEY(energy_id) REFERENCES energy (id)
);
```

SQLalchemy - Schreiben

```
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

from Model import Energy, Base, Price

engine = create_engine('sqlite:///sqlalchemy_energy.db')
Base.metadata.bind = engine

DBSession = sessionmaker(bind=engine)
session = DBSession()

new_energy = Energy(name='FluxCompensator')
session.add(new_energy)
session.commit()

new_price = Price(year=1997, price=12.98, energy=new_energy)
session.add(new_price)
session.commit()
```

SQLalchemy – Schreiben Resultat

```
sqlite> select * from energy;
id|name
1|FluxCompensator
sqlite> select * from price;
id|year|price|energy_id
1|1997|12.98|1
```

SQLalchemy - Lesen

```
from Model import *
from sqlalchemy import create_engine
engine = create_engine('sqlite:///sqlalchemy_energy.db')
Base.metadata.bind = engine
from sqlalchemy.orm import sessionmaker
DBSession = sessionmaker()
DBSession.bind = engine
session = DBSession()
session.query(Energy).all()

energy = session.query(Energy).first()

print(energy.name)
prices = session.query(Price).filter(Price.energy == energy).all()

for p in prices:
    print(p.year, p.price)

# session.query(Price).filter(Price.energy == energy).one()
# price = session.query(Price).filter(Price.energy == energy).one()
```

SQLalchemy – Lesen Resultat

```
sqlite> insert into price (year, price, energy_id) values (1998, 40.1, 1);

$ python read_data.py
FluxCompensator
1997 12.98
1998 40.1
```

WEBSCRAPING

Disclaimer - webscraping

- Vernüftig agieren, niemanden (D)DOSen
- Heruntergeladene Daten unterliegen dem Urheberrecht

- Net deppert sein!

„Old Style“

- request + regex

```
import re
import requests

url = 'https://www.bruttonettorechner.at/einkommensteuer'

gehalt = 40000
values = {'gehalt' : gehalt,'jahr' : '2017', 'familie' : '0'}
r = requests.post(url, data = values)
match = re.findall(r'Ihre Einkommensteuer</div> <div
class="h1">(.*?)€</div>',r.text,re.DOTALL)
match = re.sub(r'[.]','',match[0])
match = re.sub(r'[,]','.',match)

print('webserver: ',float(match))

# output
# webserver:  10080.0
```

Welches Werkzeug wofür

- Xpath/CSS Selector
  - Für einfache „punktuelle“ Extraktionen
- BeautifulSoup - https://www.crummy.com/software/BeautifulSoup/
  - HighLevel HTML Parser, der nicht crawled oder ähnliches macht
- Scrapy - https://scrapy.org/
  - Ein Webscraper/crawling Framework
  - Parst nicht nur HTML, sondern kann auch automatisch links folgen etc
- Braucht man den Browserkontext (dh zB Javascript), kommen zB selenium, PhantomJS, etc in Frage

CSS selector - xpath

```
import requests
from bs4 import BeautifulSoup
#  pip3 install BeautifulSoup4
from pprint import pprint
import json
from lxml import html

url = 'https://www.bruttonettorechner.at/einkommensteuer'
gehalt = 45623
user_agent = 'Mozilla/5.0 (compatible; Chrome/22.0.1229.94; Windows NT)'

user_agent = 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/35.0.1916.47 Safari/537.36'

data = {'gehalt': gehalt, 'jahr' : '2017', 'familie' : '0'}
headers = {'User-Agent': user_agent}

response = requests.post(url, data = data, headers = headers)
```

CSS selector - xpath

```
# forts.
# Methode 1
soup = BeautifulSoup(response.text, "html.parser")
for cell in soup.select('html body div#at div.frm div.well.well-
success.text-center div.h1'):
    print('beautiful soup css selector:', cell.text)

# Methode 2
tree = html.fromstring(response.text)
tax = tree.xpath('//html/body/div[1]/div/div/div[2]')[0]
print('xpath:', tax.text)

# output
$ python scrape_eks.py
beautiful soup css selector: 12.441,66
xpath: 12.441,66
```

Beautiful Soup

```
import requests
from bs4 import BeautifulSoup
#  pip3 install BeautifulSoup4
from pprint import pprint

url = "https://www.google.at/search?q=site:.at+a"
user_agent = 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_3)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.47 Safari/537.36'

headers = {'User-Agent': user_agent}

response = requests.get(url, headers = headers)
soup = BeautifulSoup(response.text, "html.parser")

for link in soup.find_all('cite'):
    print(link.text)
```

Scrapy

- scrapy startproject mygoogle
- cd mygoogle/
- scrapy genspider google google.at
- In mygoogle/settings.py eintragen
  - ROBOTSTXT_OBEY = False
  - USER_AGENT = "Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/27.0.1453.93 Safari/537.36„
- Den crawler in mygoogle/spiders/google.py programmieren

Scrapy

```
import scrapy
class GoogleSpider(scrapy.Spider):
    name = 'google'
    allowed_domains = ['google.at']

    def start_requests(self):
        urls = [
            "https://www.google.at/search?q=site:.at+a",
            ]
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)


    def parse(self, response):
        for i in response.css('cite::text').extract():
                link_file.write(i+"\n")
                yield {'link': i}
```

Scrapy ausführen

- scrapy crawl google -o at-sites.json
- cat at-sites.json

[
{"link": "https://www.d-systeme.at/"},
{"link": "Kurier.at"},
{"link": "https://d-und-s.at/"},
{"link": "www.oekopharm.at/index.php/oekomed-vitamin-d-complex.html"},
{"link": "www.vundd.at/ueber_vd.php"},
…..
]

Programm/Projekt Design

- MVC - Model - View - Controller
- die Daten des Model kommen meistens aus einer DB, können aber zB auch REST Calls sein (die dann in eine DB führen :-), etc
- Der Controller so möglichst schlank sein, möglichst keine/wenig Logik enthalten, sondern nur Model und View verknüpfen
- Views (=Input/Ouput Layer), was (vorallem wie) bekommt der User Daten präsentiert, bzw was ist das API Interface nach aussen
- Testbarkeit bedarf eines guten Designs, weil nur an sauberen Interfaces/Schnittstellen können Tests ansetzen
- Ein gutes Design zeichnet sich durch leichte Austauschbarkeit der Komponenten aus, zb wenn eine Filestorage leicht durch eine Datenbankstorage ersetzt werden kann
- Möglichst wenige Komponenten deren Verhalten konfigurierbar sein soll
- Eine „Funktion“ soll genau „eine“ Sache machen
- Möglichst viel code reuse, ständiges refactorn – das ist nur möglich wenn man Tests hat

Flask

WEB PROGRAMMIERUNG

Flask Demo App

- Http://flask.pocoo.org/docs/0.12/tutorial/
- Vorsicht, falscher source tree verlinkt!
- git: https://github.com/pallets/flask/tree/0.12-maintenance/examples/flaskr
- virtualenv --python=python3 flaskr-env
- source flaskr-env/bin/activate
- pip install --editable .
- export FLASK_APP=flaskr
- flask initdb
- flask run --host 0.0.0.0

Flask Demo REST

- Erweiterung um ein Interface: REST

```
from flask_restful import Resource, Api

api = Api(app)
class FlaskrApi(Resource):
    def get(self):
        db = get_db()
        cur = db.execute('select title, text from
entries order by id desc')
        entries = cur.fetchall()
        list_entriesj = [list(row) for row in entries]
        return list_entries

api.add_resource(FlaskrApi, '/api')
```

Interfaces

- Web (REST + JS?)
- REST
  - https://en.wikipedia.org/wiki/Representational_state_transfer
- CLI/Scripting
- Database

NETZWERK

Portscan – „primitiv“

```
import socket

TargetIP  = '127.0.0.1'

for port in range(1024,2225):
    sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    result = sock.connect_ex((TargetIP, port))
    if result == 0:
        print("Port {}: Open".format(port))
    else:
        print("Port {}: Close".format(port))
    sock.close()
```

Portscan – „parallel“

- Threads
  - Unix
  - einfacher
- Asyncio
  - Geht auch unter Windows
  - Kommunikation zwischen den Workern
  - komplexer

Portscan – „parallel“

https://stackoverflow.com/questions/26174743/python-making-a-fast-port-scanner

```
import socket, threading

def TCP_connect(ip, port_number, timeout, ports):
    TCPsock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
    TCPsock.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)
    TCPsock.settimeout(timeout)
    try:
        TCPsock.connect((ip, port_number))
        ports[port_number] = True
    except:
        ports[port_number] = False
```

Portscan – „parallel“

```
def scan_ports(host_ip, timeout):
    threads = []
    ports = {}
    for i in range(10000):
        t = threading.Thread(target=TCP_connect, args=(host_ip, i, timeout,
ports))
        threads.append(t)
    for i in range(10000):
        threads[i].start()

    # Wartet bis alle threads zuende sind
    for i in range(10000):
        threads[i].join()

    for i in range(10000):
        if ports[i]:
            print("Port [%d] Listening" % (i))

if __name__ == "__main__":
    host_ip = input("Target IP: ")
    timeout = 1
    scan_ports(host_ip, timeout)
```

OBJEKTORIENTIERUNG II

Singletons

```
class Singleton(object):
    __metaclass__ = ABCMeta
    __instance = None

    def __new__(new_singleton, *arguments, **keyword_arguments):
if new_singleton.__instance is None:
            new_singleton.__instance = object.__new__(new_singleton)
            new_singleton.__instance.__init__(*arguments,
**keyword_arguments)
        return new_singleton.__instance


class DemoBase(Singleton):

    engine = None
    session = None
    demo = None
```

Singletons

```
import pytest
from demobase import DemoBase
from Model import *
# test model
import os

def test_singleton():
    demobase = DemoBase(d_configfile = 'demo-test.ini')
    demobase.demo = 42
    demobase2 = DemoBase(d_configfile = 'demo-test.ini')
    print(demobase.demo)
    print(demobase2.demo)
    assert demobase.demo == demobase2.demo
```

Dependency Injection

- Schon alleine aus Gründen der Testbarkeit ist es nicht sinnvoll alles von einer „God Class“ abzuleiten
- zB statt direkt in einer Klasse die Datenbank abzufragen, ist es sinnvoller entweder direkt nur die Daten zu übergeben, oder eine Funktion die die Daten retourniert

Ohne Dependency Injection

```
import scrapy
import Model as datasource
import re

class GetGeneratorSpider(scrapy.Spider):
    name = 'get_generator'
    allowed_domains = ['.at']

    def start_requests(self):
        urls = datasource.get_urls()
        for url in urls:
            url = 'http://'+url
            yield scrapy.Request(url=url, callback=self.parse)


    def parse(self, response):
        fn = re.sub(r'https?://', '', response.url)
        fn = fn.replace('/', '')

        with open(fn+'.html', 'w') as file:
            file.write(str(response.text))
```

Dependency Injection

- Die Herkunft der Daten ist „fix verdrahtet“
- Nachteile beim
  - Entwickeln
  - Testen
  - Modifizieren
- Besser wäre es die Datenquelle dynamisch zu übergeben

```
#Model.py

def get_urls():
    urls = ['orf.at',
            'ripperl.at',
           ]
    return urls
```

Dependency Injection

- An scrapy spiders kann man Paramter übergeben
- Siehe https://doc.scrapy.org/en/latest/topics/spiders.html

```
scrapy crawl myspider -a category=electronics
```

- Idee die datasource als Parameter übergeben:

```
scrapy crawl get_generator -a datasource=Model
```

- Und dann den Inhalt dynamisch mit importlib einbinden

importlib

```
import importlib

my_module = importlib.import_module('os')

print(my_module.sep)
```

Dependency Injection

```
import scrapy
import importlib
import re

class GetGeneratorSpider(scrapy.Spider):
    name = 'get_generator'
    allowed_domains = ['.at']
    start_requests_func = None

    def start_requests(self):
        datasource = importlib.import_module(self.datasource)
        urls = datasource.get_urls()
        for url in urls:
            url = 'http://'+url
            yield scrapy.Request(url=url, callback=self.parse)

    def parse(self, response):
        fn = re.sub(r'https?://', '', response.url)
        fn = fn.replace('/', '')

        with open(fn+'.html', 'w') as file:
            file.write(str(response.text))
```

Traits

- http://code.enthought.com/projects/traits/docs/html/traits_user_manual/defining.html

Command line debugger

- https://docs.python.org/3/library/pdb.html
- Kann auch im kontext von pytest aufgerufen werden
  - python -m pytest

IDE - PYCHARM

Debugging

PYTHON DEBUGGER - PDB

AUFGABEN

Aufgaben

- Lotto Zahlen ermitteln (6 aus 45)
- Einkommenssteuer
- Conways Game of Life
- Wahlumfragesimulation (oo?)
- SecurePassword
  - Checken
  - Generieren
  - Keines der letzten x Passwörter, DI vs Singleton
- Webcrawler
- Webservice REST
- Port Scanner
- Inline C