



# Scripting in Python

## OO & Jupyter Notebook

O\_ASC1V: Scripting Fortgeschritten (DSE VZ)

Susanne Schaller, MMSc

University of Applied Sciences Upper Austria, Hagenberg Campus

Bioinformatics Research Group

WS 2019/2020

[susanne.schaller@fh-hagenberg.at](mailto:susanne.schaller@fh-hagenberg.at)

# OO Programmierung I

- Prozedurale Programmierung: Funktionen bzw. Blöcke von Statements
- Objektorientierte Programmierung: Klassen, Objekte, Methoden
- *„A class creates a new type where objects are instances of the class“*
- Beispiel: Variable vom Typ **int** → Variablen, welche Integer speichern, sind Instanzen (Objekte) der Klasse **int**

# self

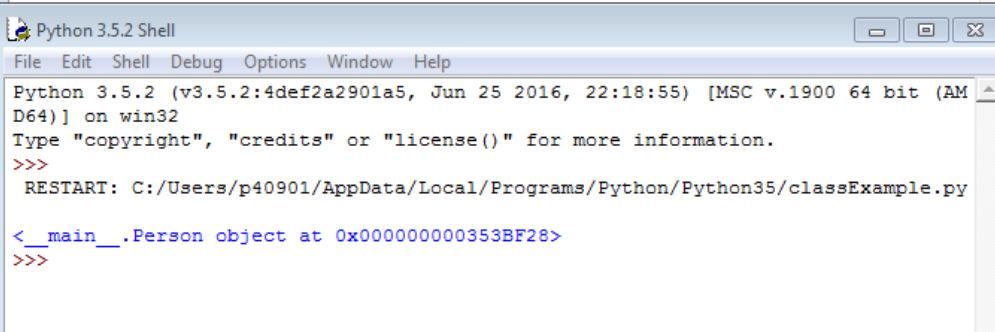
- Klassen unterscheiden sich im wesentlichen von Funktionen durch ein Merkmal:
  - Es muss ein zusätzlicher Parameter am Beginn der Parameterliste übergeben werden; allerdings benötigt dieser keinen Wert - Warum?
    - Beispiel: `MyClass` – Instanz dieser Klasse ist `myobject`
    - `myobject.method(arg1, arg2)` wird in Python zu:
    - `MyClass.method(myobject, arg1, arg2) → self = myobject`
- *„This particular variable refers to the object itself, and it is given the name `self`“*

# Beispiel – Klasse anlegen

- Klassen können mit dem Schlüsselwort **class** angelegt werden
- **pass** kennzeichnet einen leeren Block
- Objekt/Instanz der Klasse wird angelegt mit Klassenname und () Klammern

```
class Person:
    pass # empty block

p = Person()
print (p)
```



The screenshot shows a Python 3.5.2 Shell window with a menu bar (File, Edit, Shell, Debug, Options, Window, Help) and a status bar. The command prompt shows the Python version and architecture: Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AMD64)] on win32. It also displays the file path: C:/Users/p40901/AppData/Local/Programs/Python/Python35/classExample.py. The output of the code is: <\_\_main\_\_.Person object at 0x000000000353BF28>

# Beispiel – Methoden anlegen

- Methoden sehen von der Syntax genauso aus wie Funktionen
- Es gibt vordefinierte Methoden `__init__` (entspricht dem Konstruktor)
  - Diese Methode wird sofort aufgerufen sobald ein Objekt einer Klasse instanziiert wird
  - Initialisierung eines Objektes

```
import sys
import os
import re

class Person:

    def __init__(self, name):
        self.name = name

    def say_something(self):
        print("Hello!", self.name, "- I am currently learning object oriented programming!")

p = Person('Susi')

p.say_something()

Person('Susi').say_something()
```

# Klassen und Objektvariablen

- 2 Typen von Fields:
  - Klassenvariablen:
    - Kann von allen Instanzen einer Klasse zugegriffen werden
    - Es gibt nur eine einzige Kopie von der Klassenvariable – wird eine Änderung durchgeführt bei einem Objekt dann werden diese Veränderungen auch bei allen anderen Instanzen durchgezogen
  - Objektvariablen:
    - Gehören einem Objekt/Instanz einer Klasse an
    - Jedes Objekt hat seine eigenen Kopie
- Sichtbarkeit der Attribute:
  - schwach (weak) und starke (strong) Privatheit
    - `__starkprivate`: nur sichtbar innerhalb der Klassendefinition
    - `_private`: man kann von Außen zugreifen, vorausgesetzt man kennt den Namen

- \* You cannot enforce privacy
- \* You can only indicate/suggest privacy

To paraphrase a famous line about privacy from the book, the philosophy is that you should stay of the living room because you weren't invited, not because it is defended with a shotgun.

# Beispiel Life of Robot



# Klassen und Objektvariablen

```
class Robot:
    """Represents a robot, with a name."""

    # A class variable, counting the number of robots
    population = 0

    def __init__(self, name):
        """Initializes the data."""
        self.name = name
        print("(Initializing {})".format(self.name))

        # When this person is created, the robot
        # adds to the population
        Robot.population += 1

    def die(self):
        """I am dying."""
        print("{} is being destroyed!".format(self.name))

        Robot.population -= 1

        if Robot.population == 0:
            print("{} was the last one.".format(self.name))
        else:
            print("There are still {:d} robots working.".format(
                Robot.population))

    def say_hi(self):
        """Greeting by the robot.

        Yeah, they can do that."""
        print("Greetings, my masters call me {}.".format(self.name))

    @classmethod
    def how_many(cls):
        """Prints the current population."""
        print("We have {:d} robots.".format(cls.population))
```

Population gehört zur Klasse Robot und ist eine Klassenvariable

Die Variable name ist eine Objektvariable

Die methode how\_many ist eine Klassenmethode → Verwendung des decorator @classmethod

# Was ist noch möglich in Python

- Polymorphismus – Überladen von Operatoren
- Operatoren und Standardfunktionen
- Beispiele für reservierte Methodennamen:
  - `__add__(self, other)`: Überladen des Plusoperators
  - `__cmp__(self, other)`: Überladen der Vergleichsoperatoren
  - `__contains__(self, other)`: Überladen des in Operators
  - `__float__(self)`: Überladen der Standardfunktion `float()`
  - `__del__(self)`: Die Methode wird aufgerufen, wenn ein Objekt gelöscht werden soll
- Standardbeispiel: Überladung des + Operator
- Sources: <https://python.swaroopch.com/oop.html>

# Beispiel: Intelligente Pinnwand

| Pinboard_1                          |
|-------------------------------------|
| +__sheet                            |
| +__str__()<br>+delete()<br>+pinOn() |

- Die Methode delete() liefert eine Notiz mit der höchsten Priorität und entfernt sie aus der internen Liste

Entwickeln Sie ein objektorientiertes Programm, das eine intelligente Pinnwand mit Ordnungssinn modelliert. An die Pinnwand können Zettel mit Notizen geheftet werden. Auf Wunsch gibt die Pinnwand die Notizen geordnet nach Priorität wieder aus, die wichtigsten Notizen werden zuerst genannt.

Die Priorität einer Notiz wird durch eine Analyse ihres Textes ermittelt, in denen viele Ausrufungszeichen oder Bemerkungen wie „wichtig!“ oder „dringend!“ vorkommen, erhalten eine hohe Priorität.

# Beispiel: Intelligente Pinnwand

Klasse Pinboard

3 Methoden

`pinOn(self, note)` – Anheften eines Notizzettels an die Pinnwand – Argument ist ein String-Objekt

Durch eine Textanalyse wird eine ganze Zahl als Priorität ermittelt; Methode: `calculatePriority(self, note)`

Paar aus Text und Priorität wird der internen Liste `__sheet` hinzugefügt

Methode `__str__()` überlädt die Standardfunktion `str()`  
→ Diese Methode liefert einen String mit einer kompletten Auflistung der Notizzettel – geordnet nach Priorität

## Pinnwand

<N>eue Notiz anheften                      <A>lle Notizen auflisten  
<W>ichtigste Notiz entfernen              <E>nde

Ihre Wahl: A  
Notiz

<N>eue Notiz anheften                      <A>lle Notizen auflisten  
<W>ichtigste Notiz entfernen              <E>nde

Ihre Wahl: N  
Notiz: Dringend Python implementieren  
Notiz: Lebensmittel besorgen?  
Notiz: Proske anrufen WICHTIG!  
Notiz: Assignment fertig dokumentieren?  
Notiz:

<N>eue Notiz anheften                      <A>lle Notizen auflisten  
<W>ichtigste Notiz entfernen              <E>nde

Ihre Wahl: A  
Notiz  
Proske anrufen WICHTIG!<Prioritaet: 2>  
Lebensmittel besorgen!<Prioritaet: 1>  
Dringend Python implementieren<Prioritaet: 1>  
Assignment fertig dokumentieren!<Prioritaet: 1>

<N>eue Notiz anheften                      <A>lle Notizen auflisten  
<W>ichtigste Notiz entfernen              <E>nde

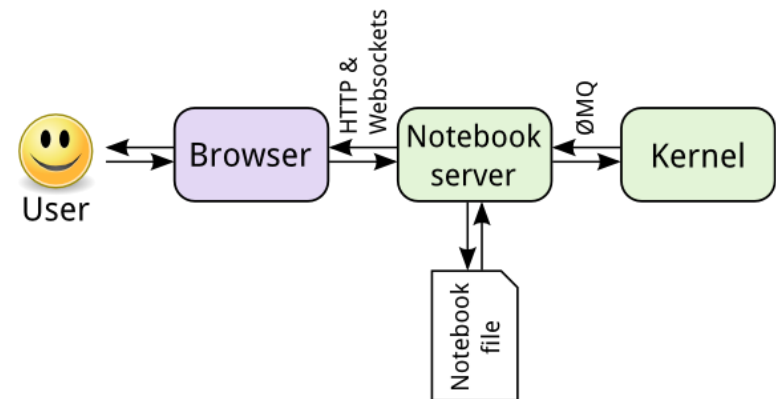
Ihre Wahl: W  
Proske anrufen WICHTIG!

<N>eue Notiz anheften                      <A>lle Notizen auflisten  
<W>ichtigste Notiz entfernen              <E>nde

# Jupyter



- Begann 2001 als IPython (Interactive Python)
- Entwickelt von Fernando Pérez
- 2014 gab Pérez bekannt, ein Spin-Off Projekt zu starten, er nannte es Project Jupyter
- Name Jupyter stammt von Sprachen Julia, Python und R
- IPython wurde zu einem Kernel für Jupyter



Bildquelle:

[https://jupyter.readthedocs.io/en/latest/architecture/how\\_jupyter\\_ipython\\_work.html](https://jupyter.readthedocs.io/en/latest/architecture/how_jupyter_ipython_work.html)

# Jupyter Notebooks



- Jupyter Notebooks sind webbasierte interaktive Programmierumgebungen
- Werden als JSON Dokumente gespeichert
- Enthalten eine geordnete Liste an Input/Output Zellen
- Jede Zelle kann Code, Text (via Markdown), mathematische Formeln, Diagramme, oder Mediendateien enthalten

# Jupyter Kernel



- Jupyter Kernel übernehmen die Ausführung der verschiedenen Sprachen
- Standardmäßig wird Jupyter mit dem IPython Kernel ausgeliefert
- Es existieren jedoch weit über 100 weitere Kernel  
z.B. ICSsharp, IRKernel
- Mit deren Hilfe kann Jupyter um nahezu jede Sprache erweitert werden



```
Administrator: Command Prompt - jupyter notebook
Microsoft Windows [Version 10.0.18362.387]
(c) 2019 Microsoft Corporation. All rights reserved.

U:\>jupyter notebook
[I 15:53:40.318 NotebookApp] Serving notebooks from local directory: U:\
[I 15:53:40.318 NotebookApp] The Jupyter Notebook is running at:
[I 15:53:40.318 NotebookApp] http://localhost:8888/?token=f52e31d1691d1d7086ee8c56bf097ec77c3171684ff6de69
[I 15:53:40.318 NotebookApp] or http://127.0.0.1:8888/?token=f52e31d1691d1d7086ee8c56bf097ec77c3171684ff6de69
[I 15:53:40.318 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 15:53:40.365 NotebookApp]

To access the notebook, open this file in a browser:
    file:///C:/Users/p27588/AppData/Roaming/jupyter/runtime/nbserver-35032-open.html
Or copy and paste one of these URLs:
    http://localhost:8888/?token=f52e31d1691d1d7086ee8c56bf097ec77c3171684ff6de69
    or http://127.0.0.1:8888/?token=f52e31d1691d1d7086ee8c56bf097ec77c3171684ff6de69
```

```
In [1]: %matplotlib notebook
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from math import pi
import scipy
import math
```

```
In [2]: df1 = pd.read_csv("data_preparation.csv", sep=';')
```

## Überschrift in Markdown

Hierüber ist eine horizontale Linie

*Kursiv* **Fett** **Fett** *Kursiv*

CodeBlock  
über Einrückung

- Ein Aufzählungspunkt
  - in der nächsten Ebene
- Wieder vorherige Ebene

```
In [3]: df1
```

Out[3]:

|   | Zeitstempel            | Geschlecht | Alter      | Höchste<br>abgeschlossene<br>Ausbildung | Aufgeschlossenheit | Perfektionismus | Geselligkeit | Kooperationsbereitschaft | Verletzlichkeit | Fahren Sie<br>regelmäßig<br>Auto? |
|---|------------------------|------------|------------|---|--------------------|-----------------|--------------|--------------------------|-----------------|-----------------------------------|
| 0 | 12.22.2017<br>16:41:19 | weiblich   | 30 -<br>39 | Bachelorstudium                         | 4                  | 4               | 2            | 4                        | 3               | Ja                                |
| 1 | 12.22.2017<br>19:24:39 | weiblich   | 30 -<br>39 | Bachelorstudium                         | 2                  | 4               | 4            | 3                        | 4               | Ja                                |
|   | 12.22.2017             |            | 30 -       |   |                    |                 |              |                          |                 |                                   |

# PYTHON PHILOSOPHY

“Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.  
Special cases aren't special enough to break the rules.  
Although practicality beats purity.  
Errors should never pass silently.  
Unless explicitly silenced.  
In the face of ambiguity, refuse the temptation to guess.  
There should be one-- and preferably only one --obvious way to do it.  
Although that way may not be obvious at first unless you're Dutch.  
Now is better than never.  
Although never is often better than \*right\* now.  
If the implementation is hard to explain, it's a bad idea.  
If the implementation is easy to explain, it may be a good idea.  
Namespaces are one honking great idea -- let's do more of those!”