FH JOANNEUM
University of Applied Sciences

Python in a Day

Programmieren leicht gemacht

Maierbrugger, Raab

Kursplan

Vormittag: 09:00 – 12:30

Pause nach Bedarf

Inhalte:

- Setup
- Programmier Basics
- Selected Pythonic code
- Type hinting, type checking
- Modules

Nachmittag: 13:30 – 17:00

Pause nach Bedarf

Inhalte:

- Datenmanagement
- Machine Learning
- Datenvisualisierung
- Python Utilities

SETUP

Voraussetzungen für Python Programmierung

- Python Installation - kompatibel mit den Packages die man verwenden will
- Virtuelle Umgebung – isolierte Umgebung für jedes Projekt
- Python Packages (Libraries)
- Dependency management – Abhängigkeiten der Packages untereinander verwalten
- Publishing – Code soll möglichst einfach weitergegeben werden können
- Eine Entwicklungsumgebung (IDE)

Visual Studio Code

- Code editor für Windows, Linux, MacOS
- https://code.visualstudio.com/

VS Code installieren

Visual Studio Code

- Settings
  - Theme
  - Auto save
  - Default python path
- Extensions installieren
  - Python
  - Jupyter
  - Ruff

UV – Python Package Manager

- Kann alle der genannten Voraussetzungen (außer IDE) managen
- Einfach zu benutzen
- Schneller als andere Package Manager (pip, anaconda, poetry)
- Hat eine sehr gute Dokumentation:
  - https://docs.astral.sh/uv/

UV installieren

UV – Wichtige Commands

- uv --help
- uv python list
- uv python install 3.8 / uv python uninstall 3.8
- uv python pin <python-version>
- uv init … erstellt ein environment im momentanen Folder
- uv init <project_name> … neuer Folder project_name
- uv add <package> <package> <package>
- uv sync

UV – Konfigurations-Files

- Konfigurationsfiles beinhalten die dependencies eines environments
- Mithilfe dieser Files kann eine Umgebung verlässlich wiederhergestellt werden
  - `pyproject.toml`
    definiert die breiten requirements und enthält Python Projekt Metadaten
  - `uv.lock`
    Enthält die exakten, aufgelösten dependency versionen
- Diese Files (vor allem `uv.lock`) sollten nicht manuell verändert werden!

RUFF

- Ruff:
  - Code Formatter und Linter
  - Konfigurieren mittels (*ruff.toml* file) – oder default rules
  - Rules: https://docs.astral.sh/ruff/rules/
  - Ausführliches Beispiel für ruff.toml: https://github.com/CoreyMSchafer/dotfiles/blob/master/settings/ruff.toml
  - RUFF als Formatter und Linter aktivieren:
    - Strg + Shift + p
    - „User Settings JSON“

RUFF

Formatter und Linter testen

test_ruff.py > ...

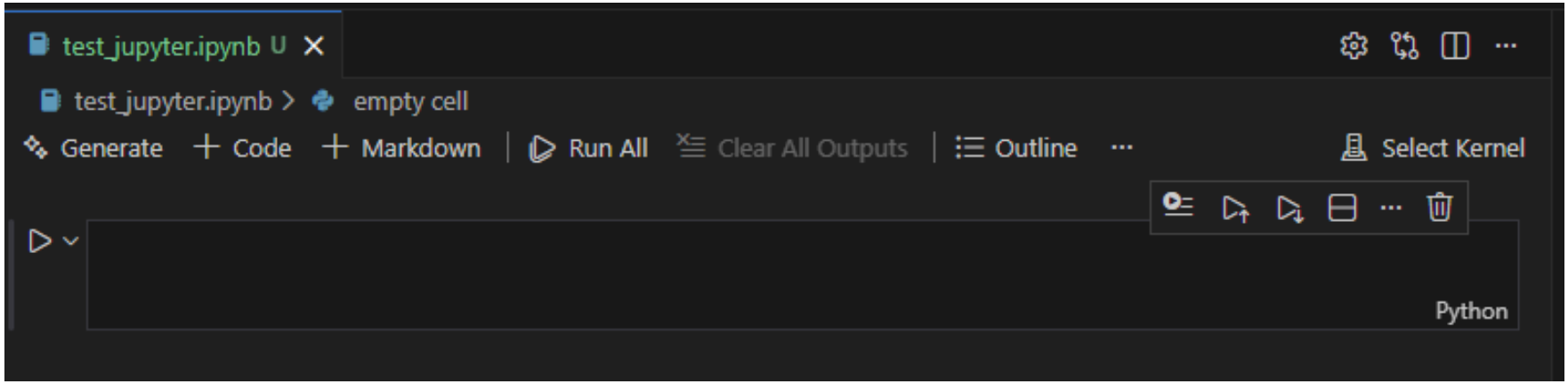
```python
import os
import pandas as pd
import numpy as np


p = {1: 'this is a test for my formatter',2: 'it will surely manage to clean this mess',3: 'I would hatefor my code to be so ugly',
}
def function1(a, b):
    return np.sum(a + b)

def function(c):
    return pd.DataFrame(c)
g = 'another variable'
```

Jupyter - Notebooks

- Erlauben die Kombination von Markdown text und ausführbarem Python Code
- Verwenden die virtual environment als kernel
- Man kann einzelne Zellen ausführen (oder mehrere auf einmal)



BASICS

Elementare Datentypen

- Integers (int)
- Dezimalzahlen (float)
- Strings (str)
- Boolean values (bool)
- Prüfen eines Datentyps mit Funktion type()

```
age = 10
share = 0.25
name = 'Michael'
is_valid = True

type(name)
```

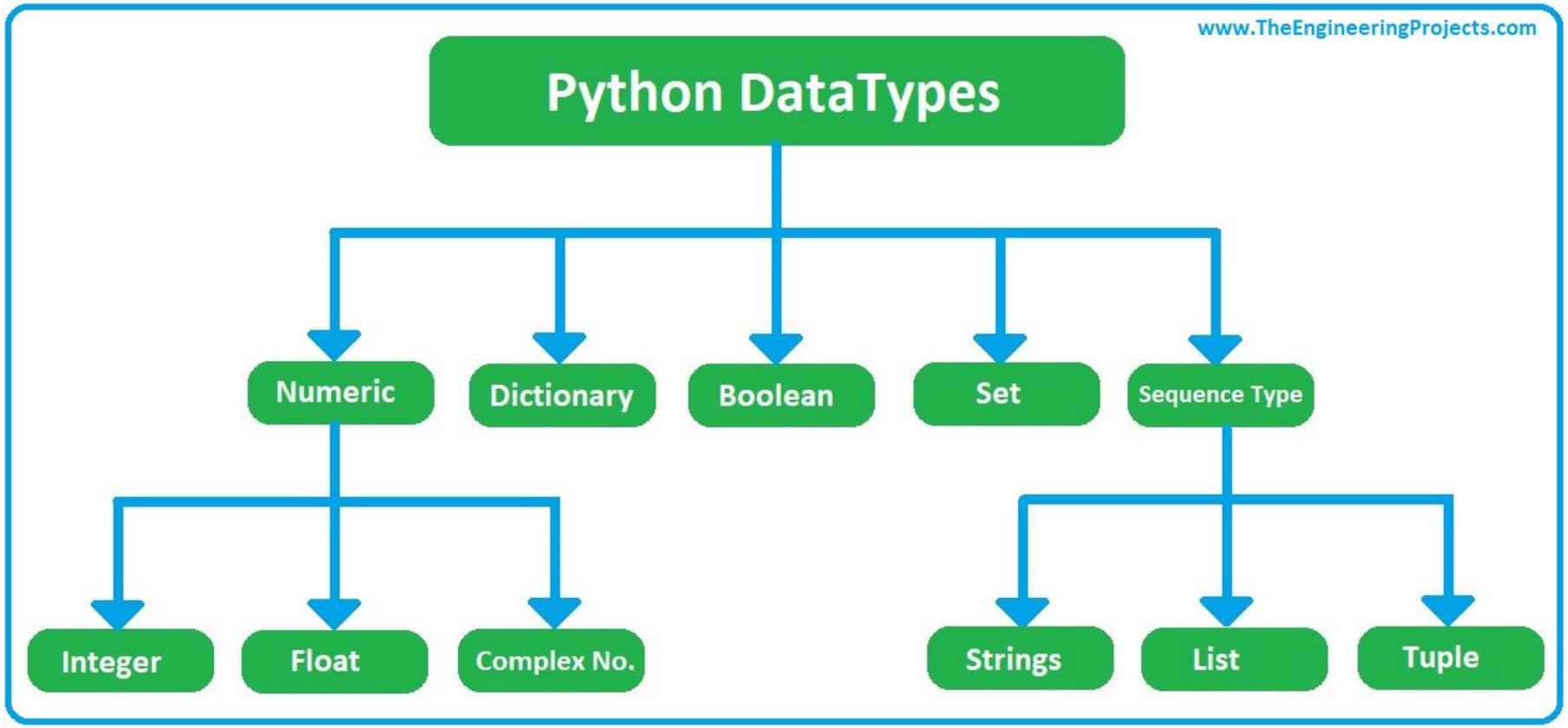
```
<class 'str'>
```

Nicht-elementare Datentypen

- Listen (list): geordnet, mutable, eckige Klammern
- Tupel (tuple): geordnet, immutable, runde Klammern
- Menge (set): ungeordnet, mutable, geschwungene Klammern, keine Duplikate
- Dictionaries (dict): key-value pairs, mutable, geschwungene Klammern

```
days = ['Monday', 'Tuesday', 'Wednesday']
coordinates = (10.5, 20.3)
teammembers = {'Paul', 'Jim', 'Sally'}
person = {'name': 'Alice', 'age': 30}
```

Datentypen Übersicht



Operators - Arithmetic

Operator	Meaning	Example	Result
+	Addition	5 + 3	8
-	Subtraction	10 - 4	6
*	Multiplication	6 * 2	12
/	Division	8 / 2	4.0
//	Floor Division	9 // 2	4
%	Modulus (Remainder)	10 % 3	1
**	Exponentiation (Power)	2 ** 3	8

Operators - Relational

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True

Operators - Assignment

Operator	Example	Equivalent Expression (m=15)	Result
=	y = a+b	y = 10 + 20	30
+=	m +=10	m = m+10	25
-=	m -=10	m = m-10	5
*=	m *=10	m = m*10	150
/=	m /=10	m = m/10	1.5
%=	m %=10	m = m%10	5
**=	m**=2	m = m**2 or $m = m^2$	225
//=	m//=10	m = m//10	1

Control Flow – Conditional Statements

- Python benutzt Einrückungen und einen Doppelpunkt, um Code-Blöcke zu separieren
- Syntax für Conditional Statements:

```
age = 9

if age >= 18:
    print("adult")
elif age >=12:
    print("teenager")
else:
    print("child")
```

✓ 0.0s

```
child
```

Control Flow – Loops

- **For loops** können verwendet werden, um über ein iterable object (list, tuple, str, …) zu iterieren
- Man benötigt keinen Index!

```
coordinates = (10.5, 20.3)

for coordinate in coordinates:
    print(coordinate)
```

✓ 0.0s

```
10.5
20.3
```

Control Flow – Loops

- **While loops** führen einen Code aus, solange eine Bedingung erfüllt ist
- Achtung! Gefahr von Endless-Loops

```
number = 1

while number > 0.1:
    print(number)
    number *= 0.5
```

✓ 0.0s

```
1
0.5
0.25
0.125
```

Functions

- User-definierte Funktionen in Python werden mit dem def keyword definiert

```
def addition(a, b):
    return a + b

addition(13,3)
```

✓ 0.0s

```
16
```

PYTHONIC CODE

Zen of Python

- 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.

Tim Peters (Peters 2004)