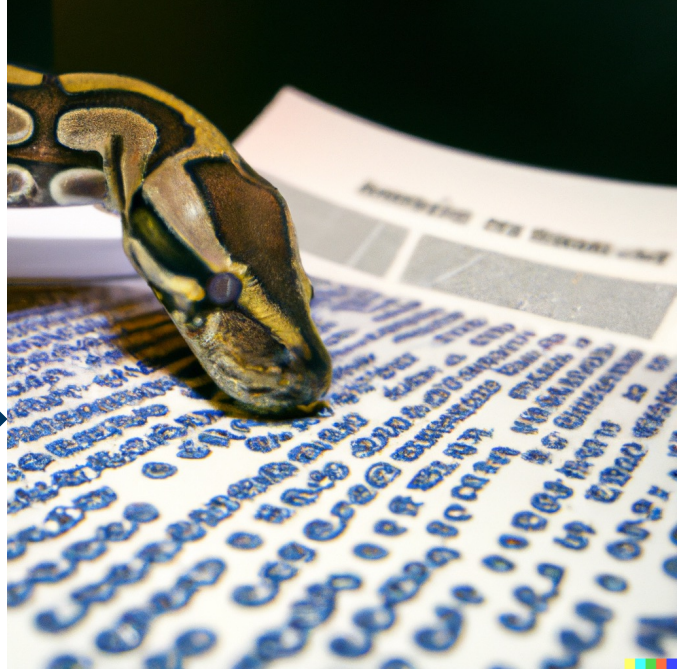
AI generated



RUB

CUBiMed.RUB
CORE UNIT BIOINFORMATICS
MEDIZINISCHE FAKULTÄT
DER RUHR-UNIVERSITÄT BOCHUM

RUHR-UNIVERSITÄT BOCHUM

DATA EXPLORATION WITH PYTHON

The young researcher's toolkit

Why Python?

- **Easy to use**
- **Common in scientific setting**
- **Lot of 3rd party support**
- **Works on Linux, MacOS and Windows**
- **General purpose**
  - Automating (Ansible)
  - Web application (streamlit)
  - Machine and Deep Learning (Keras & PyTorch)
  - Data analysis & visualization (Pandas, Plotly)
  - Game development (Pygame)

Similar languages

- **R**
  - Not a general purpose language
  - Focused more on scientific applications like statistics, algebra, machine learning
- **Ruby**
  - Not much support for scientific libraries

What you will learn today

- **Basic idea how to use Python**
  - Installation
  - Syntax & semantic
  - Structure & concepts
- **Installation and management of (scientific) dependencies**
- **First exploration of a dataset**

Installation: VSCode

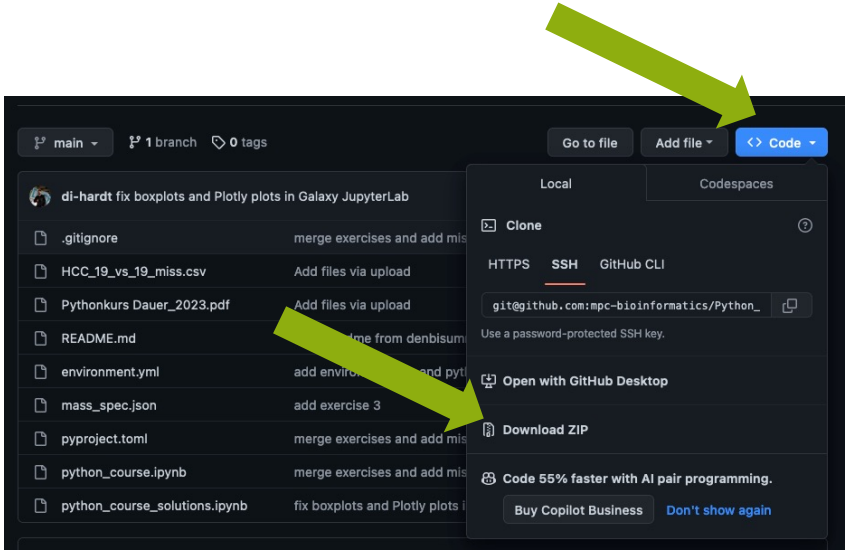
- **Integrated development environment (IDE)**
- **Visual Studio Code: https://code.visualstudio.com/Download**
  - Extensions
    - https://marketplace.visualstudio.com/items?itemName=ms-python.python
    - https://marketplace.visualstudio.com/items?itemName=ms-python.vscode-pylance
    - https://marketplace.visualstudio.com/items?itemName=ms-toolsai.jupyter




Installation: Micromamba

- **Fully compatible alternative to (Ana-)conda**
  - But much faster
  - Makes tool installation, easy and reproducible
  - Keeps different environments separated
    - Makes installation of one tool in different versions possible
- **https://mamba.readthedocs.io/en/latest/installation/micromamba-installation.html**

Download the prepared repository



https://l.rub.de/3a11cda2

Installation: Create the environment for today

- **Python: Open a Terminal (Linux/Mac) or CMD (Windows) and type**

```
micromamba -c defaults env create -f environment.yml
micromamba activate 20231117_python_course_med
```

- `environment.yml` **- Looks software up in the default channels of Conda**
  - Unlike Anaconda micromamba only uses the supplied chanels
  - Usfull channels
    - `bioconda` - Biology related software (also Proteomics related)
    - `conda-forge` - Various software like compilers, libraries etc

Installation: Create the environment for today

`environment.yml`

```yaml
name: 20231117_python_course_mpc
channels:
  - defaults
  - bioconda
  - conda-forge
dependencies:
  - python=3.11
  - setuptools
  # here we install the software `pip` the Python package manager
  - pip
  # this calls pip to install the necessary Python packages
  # it basically searches in the current directory for one of the supported requriements files
  # we use the some what new pyproject.toml file
  - pip:
    - -e ./
```

Source file

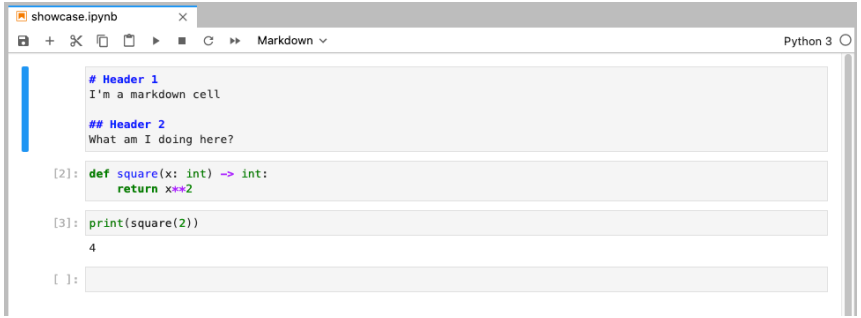
- **Normally your code lives in Python files ending with `.py`**
- **Which are executed using the command:**
  `python do_what_i_say.py`
- **Python starts to execute with the first line and goes through the file line by line**

```python
# std imports
import argparse
import importlib
from pathlib import Path
from types import ModuleType

# internal imports
import sdrf_convert.config_generators as config_generators
import sdrf_convert.sdrf_generators as sdrf_generators

def import_submodules(sub_module: ModuleType, python_file_wildcard: str):
    """Import the submodules of a module dynamically
    """
    for converter_path in Path(sub_module.__file__).parent.glob(python_file_wildcard):
        importlib.import_module(f"{sub_module.__name__}.{converter_path.stem}")

# Import all subclassed generators automatically so they are available in Abstract*Generator.__subclasses__()
# and can be automatically added to the CLI
import_submodules(config_generators, '*_config_generator.py')
import_submodules(sdrf_generators, '*_sdrf_generator.py')


You, yesterday | 2 authors (You and others)
class CommandLineInterface:
    """
```

Jupyter Notebooks

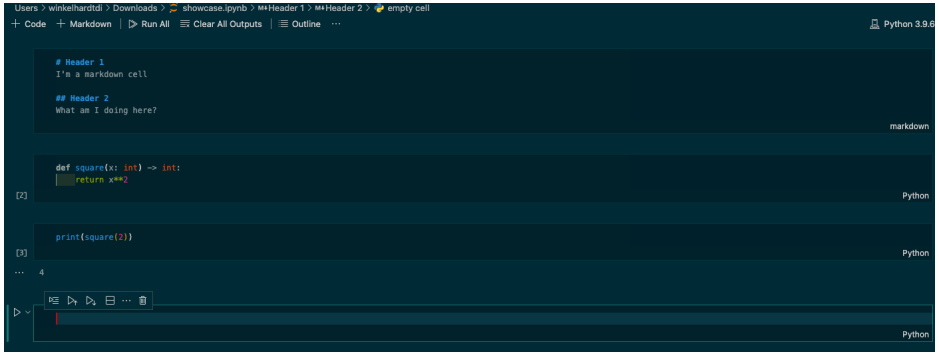
- **Jupyer Notebooks are a frontend for combining code and text blocks**
  - They can display results immediatly (plots, tables, text)
  - Export to PDF, HTML, TeX, ...
  - Runs
    - Visual Studio Code (local)
    - Galaxy (Browser)
      - Which basically starts Jupyter Lab (self hosted / browser)
    - Google Colab

Jupyter Notebooks



Jupyter Lab (browser)

Visual Studio Code (local)



Primitive data, standard operators and comments

Primitive datatypes

Type	Description	Examples
Integer		... -2, -1, -0, -1, -2 ...
Float	Numbers of various **precision** (rounding errors included)	..., -0.2, 0.0001, 0.0, 0.3, 0.333335, ...
Boolean	Value about **truthiness**	True & False

Primitive data, standard operators and comments

Standard operator

Operator	Usage	Examples
Assignment	=	i = 1
Equals	==	1 == 1
Not equals	!=	1 != 2

Primitive data, standard operators and comments

Numeric standard operator

Operator	Usage	Examples
Addition	+	1 + 1 == 2
Substraction	-	1 - 1 == 0
Multiplication	*	2 * 3 == 6
Division	/	6 / 3 == 2
Modulo	%	5 % 2 == 1
Exponential	**	5 ** 3 == 125

Primitive data, standard operators and comments

Boolean standard operator

Operator	Usage	Examples
and	and	True and True == True True and False == False False and True == False False and False == False
or	or	True or True == True True or False == True False or True == True False or False == False
negation	not	not True == False not False == True not not True == True not not not not not True == False

Syntax

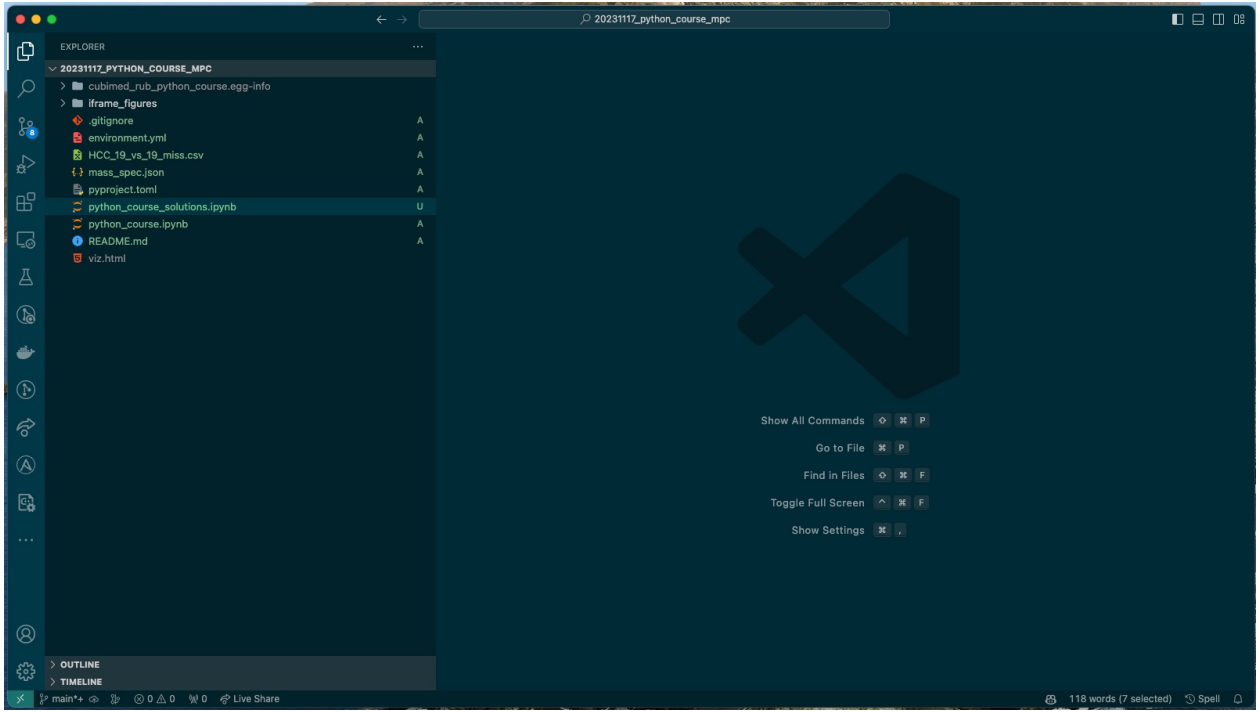
```
# This is a comment, it‘s start with ‚#‘ and is not evaluated by Python
“““
This is also a comment but spanning over
multiple lines
“““
# Your future you will appreciate to see some of them in your code ... Believe me


x = 1 # This is an assignment/declaration
y = 3 # This is also an assignment/ declaration
x + y # This is an operation, adds y to x
z = x + y # operation and assignment/declaration
print(“z is:”, z)
=> z is: 4
```

Exercise 1

- **Open the downloaded folder in Visual Studio Code**
- **Now we do a few steps together. After each step raise your hand to show you're ready!**

https://l.rub.de/3a11cda2

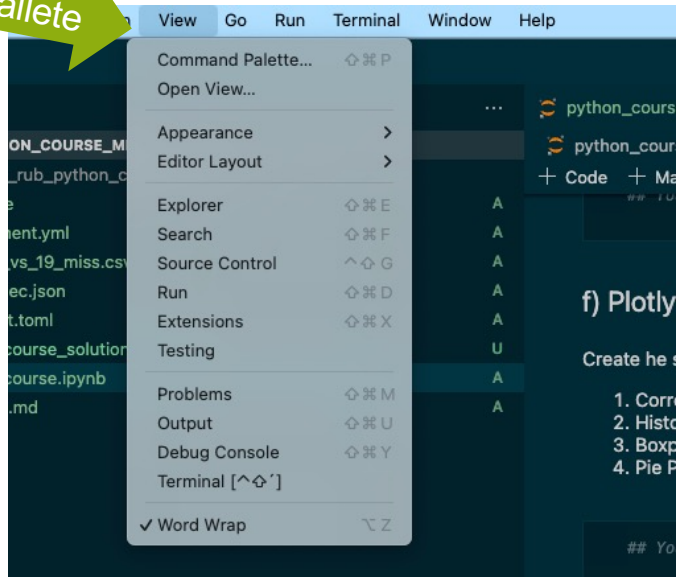
Exercise 1



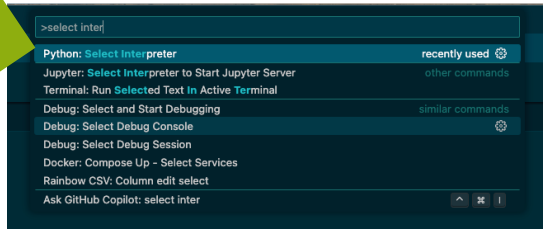
Exercise 1

https://l.rub.de/3a11cda2

- **Open the command pallete**



- **Type in: `select inter` and select `Python: Select interpreter`**

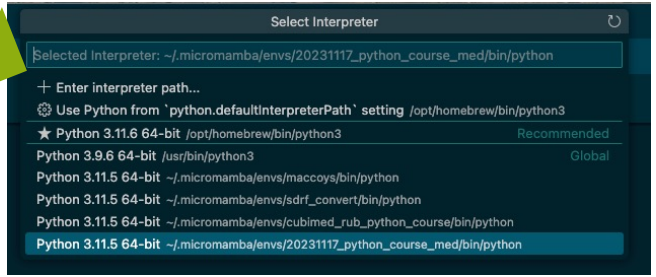


CUBiMed.RUB
CORE UNIT BIOINFORMATICS
MEDIZINISCHE FAKULTÄT
DER RUHR-UNIVERSITÄT BOCHUM
RUB

Exercise 1

https://l.rub.de/3a11cda2

- **Select `Enter interpreter path …`**



- **Type in your path to micomamba + `/envs/20231117_python_course_med/bin/python`**
  - In my case: `~/.micromamba/envs/20231117_python_course_med/bin/python`

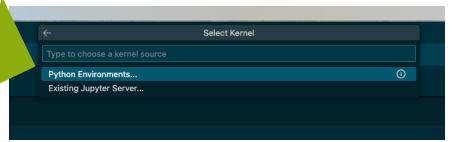
- Open `python_course.ipynb`

Exercise 1

https://l.rub.de/3a11cda2

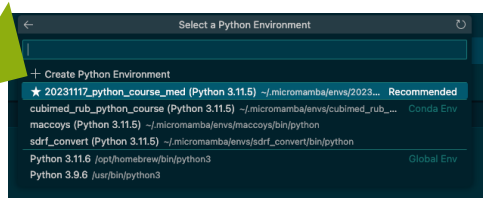
- **Select your kernel**

- **Python environment**



Python env

- **Your selected python environment should be listed**

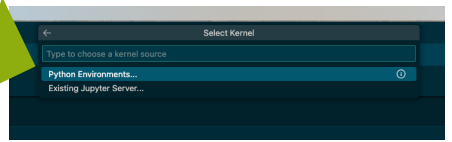


Exercise 1

https://l.rub.de/3a11cda2

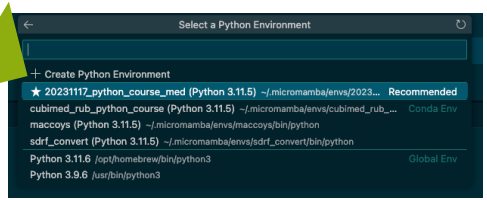
- **Select your kernel**

- **Python environment**



Python env

- **Your selected python environment should be listed**



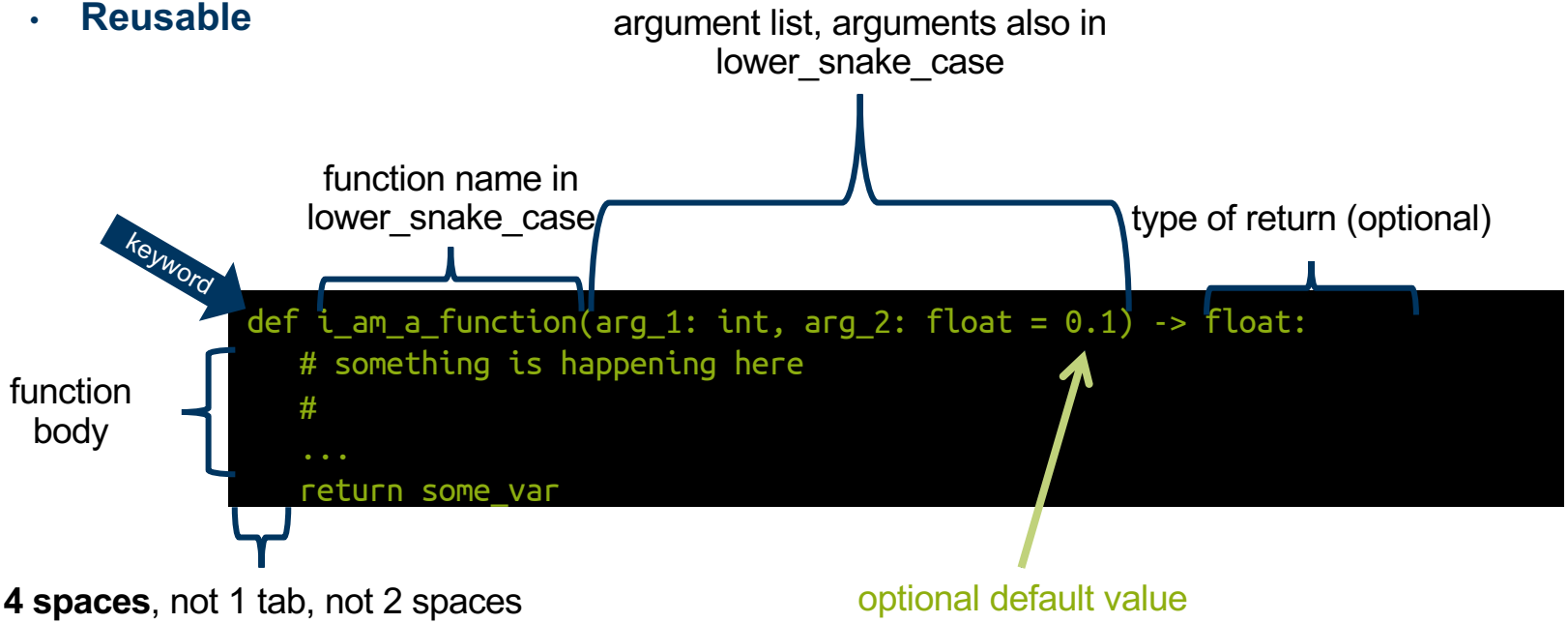
Exercise 1

https://l.rub.de/3a11cda2

Happy coding!!!

Functions

- **Represents a calculation / method / process / user interaction ...**
- **Reusable**



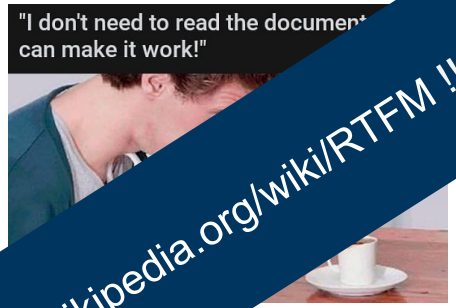
Functions

- **Built-in functions**
  - `pow(base, exp, mod=None)` - Returns *base* to the power of *exp*
  - `round(number, ndigits)` - Rounds *number* to *ndigits*, if *ndigits* is *None* (default) it rounds to nearest integer
  - `print(*objects, sep=' ', end='\n', file=None, flush=False)` - Prints the given *objects*, using the provided separator and end character. *file* & *flush* is for advanced purpose.

  - More: https://docs.python.org/3/library/functions.html

RTFM – Read the ******* manual (docs)

You can find the Python documentation here: https://docs.p…n.org/3/



!!! https://en.wikipedia.org/wiki/RTFM !!!

Function usage

```
print(“I am“, “Groot“)
=> “I am Groot“
print(“I am“, “Groot“, sep=“__“)
=> “I am__Groot“
round(3.14159265359)
=> 3
round(3.14159265359, 2)
=> 3.14
pow(2, 3)
=> 8

# save the result of a function
res = pow(2,3)
```

Imports

- **Python contain more functionality then the built-in function and additional constants like the number Pi**
- **Make them accessible via *imports***

```
import math
import random

print(math.pi)
=> 3.141592653589793
math.ceil(math.pi)
=> 4
random.randint(1, 10)
=> ???
```

Complex data types & data collections

Type	Description	Examples
List	A list of **elements**	[1,2,3,4,5], [„I“, „am“, „Groot“], [True, False, True, True]
String	List of character or simply: a **text**.	„“I am Groot“
Tuple	**Immutable** list of elements	(1,2,3,4), („I“, „am“, „Groot“), („only one element requires a comma at the end“,)
Set	A list of **unique** elements	{„Every“, „value“, „only“, „once“}
Dictionary	A list of key-value pairs. Each key is **unique**.	{„DS9“: „Deep Space 9“, „Greece“: „Athen“, 1: 4, „a list“: [1,2,3,4,5]}

https://docs.python.org/3/tutorial/datastructures.html

Complex data types & data collections

```
a_list = [1, 2, 3, “4”]
# zero based indexing
print(a_list[0])
=> 1

# we can append new elements
a_list.append(5)
print(a_list)
=> [1, 2, 3, “4”, 5]

# removing the first element
a_list.pop()
```

https://docs.python.org/3/tutorial/datastructures.html#more-on-lists

Complex data types & data collections

```
a_dict = {“Groot” : “I am Groot”, ”This is”: ”Sparta”, 7: “of 9?”}
# Accessing by key
print(a_dict[“Groot”])
=> I am Groot
print(a_dict[7])
=> of 9?

# Adding a new element
a_dict[“Yoda”] = ”Do or do not, there is nor try”

# Removing an element
a_dict.remove(“Yoda”)
```

https://docs.python.org/3/tutorial/datastructures.html#dictionaries

Exercise 2

https://l.rub.de/3a11cda2

Control flow

- **if-else-statement**

```
surname = input(“Enter surname:“)
if surname[0].lower() < “m”:
    print(“Your case worker is Person X”)
else:
    print(“Your case worker is Person Y”
```

- **if-elif-else-statement**

```
surname = input(“Enter surname:“)
if surname[0].lower() < “j”:
    print(“Your case worker is person X”)
elif surname[0].lower() >= “j” and surname[0].lower() < “q”:
    print(“Your case worker is person Y”)
elif ... :
else:
    print(“No idea who your case worker is: ¯\_(ツ)_/¯ ”
```

Control flow

- **`match`-statement**
  - Introduced in Python 3.10

```
def get_mass_of_amino_acid(amino_acid: str) -> float:
    match amino_acid.lower():
        case “a”:
            return 71.037113805
        case “c”:
            return 103.009184505
        ...
        case “w”:
            return 186.079312980
        case _:
            # Default case
            raise Error(f”’{amino_acid}’ not found”)
```

Control flow

- **`try-catch-statement`**

```
amino_acid = input("Amino acid: ")

try:
    get_mass_of_amino_acid(amino_acid)
except Error as err:
    print("Oops! Something went wrong. You might want to have a look:", err)
```

Control flow

- **Loops**

```
# Simple for loop counting from 0 to 9
for i in range(10):
    print(i)
```

```
# A while loop repeats until a given condition is false
i = 0
while i < 10:
    print(i)
    i += 1
```

- **You can skip a step in the loop by calling `continue`**
- **Leaving a loop is possible with `break`**

Control flow

- **Loops**

```
numbers = [1,2,3,4,5,6,7,8,9,110]

for i in numbers :
    print(i)
```

- **Types which can be iterated called *iterables*!**

Control flow

- **Loops**

```
dictionary = {“I am”: “Groot”, “This is”: “Sparta”}
for i in dictionary:
    print(i)
```

```
dictionary = {“I am”: “Groot”, “This is”: “Sparta”}
for k, v in dictionary.items():
    print(k, v)
```

Exercise 3

https://l.rub.de/3a11cda2

Unleash the force: The light side

- Next to the built-in functions and modules you can use 3rd party imports
- You need to install them first

```
> pip install <PACKAGE_NAME>
```

- You can find them
  - Python Package Index https://pypi.org/
  - GitHub (you can install directly from a repository)
- Be aware that this packages can also contain harmful code
  - Never install a package from a ZIP archive or similar
  - Look up the developers on GitHub
  - Check the stars on GitHub

Unleash the force: The dark side

```
import argparse
from datetime import datetime, timedelta
from io import IOBase
from typing import ClassVar, Dict, Iterator, List, Tuple, Type, Union
import pandas as pd
from pathlib import Path
from pyteomics import mass
import re
from sdrf_convert.abstract_converter import AbstractConverter
import xml.etree.ElementTree as ET
```

What is the problem here?

Unleash the force: The dark side

```
# std imports
import argparse
from datetime import datetime, timedelta
from io import IOBase
from typing import ClassVar, Dict, Iterator, List, Tuple, Type, Union
from pathlib import Path
import re

# 3rd party imports
import pandas as pd
from pyteomics import mass
import xml.etree.ElementTree as ET

# local import
from sdrf_convert.abstract_converter import AbstractConverter
```

Unleash the force: Tame the dark side

```
astroid==3.0.1
dill==0.3.7
greenlet==3.0.1
isort==5.12.0
lxml==4.9.3
mccabe==0.7.0
numpy==1.26.1
pandas==2.0.3
platformdirs==3.11.0
pylint==3.0.2
pyteomics==4.6.3
python-dateutil==2.8.2
pytz==2023.3.post1
-e git+ssh://git@github.com/di-
hardt/sdrf_convert.git@d2c570754dbf8939cd1c35de3a55628697a0e48f#egg=sdrf_convert
six==1.16.0
SQLAlchemy==2.0.23
tomlkit==0.12.2
typing_extensions==4.8.0
tzdata==2023.3
```

- **Old school: `requirements.txt` contains direct and indirect dependencies (or sub dependencies)**
- **Why could this be a problem?**

Unleash the force: Tame the dark side

```toml
[build-system]
requires = ["setuptools", "wheel"]

[project]
name = ”<SOME_NAME>"
version = "0.0.1"
requires-python = ">=3.11"

dependencies = [
    "pandas >= 2, < 2.1.0",
    "pyteomics >= 4.6, < 5",
    "lxml >= 4.9.3, < 5",
    "sqlalchemy >= 2, < 3",
]


[project.optional-dependencies]
dev = [
    "pylint"
]
```

- **The newest way: `pyproject.toml` contains direct and indirect dependencies (or sub dependencies)**
- **Don’t put them in the conda’s `environment.toml`**

Unleash the force: Tame the dark side

```
# Install a requirements.txt
pip install -r requirements.txt


# Install a pyproject.toml
pip install <FOLDER_WHERE_PYPROJECT_TOML_IS_LOCATED>
```

Object oriented programming

How would YOU represent a mass spectrum with m/z and intensities?

Nope.

Object oriented programming

- **Let‘s abstract something by creating a template, a so called `class`**

```
class MassSpec:
    # constructor
    def __init__(self, mz: list, intensities: list):
        self.mz = mz
        self.intensities = intensities

    def get_nearest_peak(self, target_mz: float) -> tuple:
        # Dum dum dum du, dum du dum du duu  (Jeopardy!-melody)
        return  (mz, intensity, index, distance)
```

Object oriented programming

- **Lets build some mass spectra from our template**

```
# Create a mass spec
a_spec = MassSpec([103.4, ...], [3040, ...])
# Create another mass spec
another_spec = MassSpec([405.4, ...], [1000, ...])

# We can also arrange them in a list
mass_specs = [a_spec, another_spec]

# Or directly in a list
mass_specs = [
    MassSpec([103.4, ...], [3040, ...]),
    MassSpec([405.4, ...], [1000, ...]))
]
```

Object oriented programming

- **Lets play with out mass spectra**

```
for spec in mass_specs:
    # print mz
    print(spec.mz)
    # find nearest peak to 356.8 in all mass specs
    print(spec.find_nearest_peak(356.8)
```

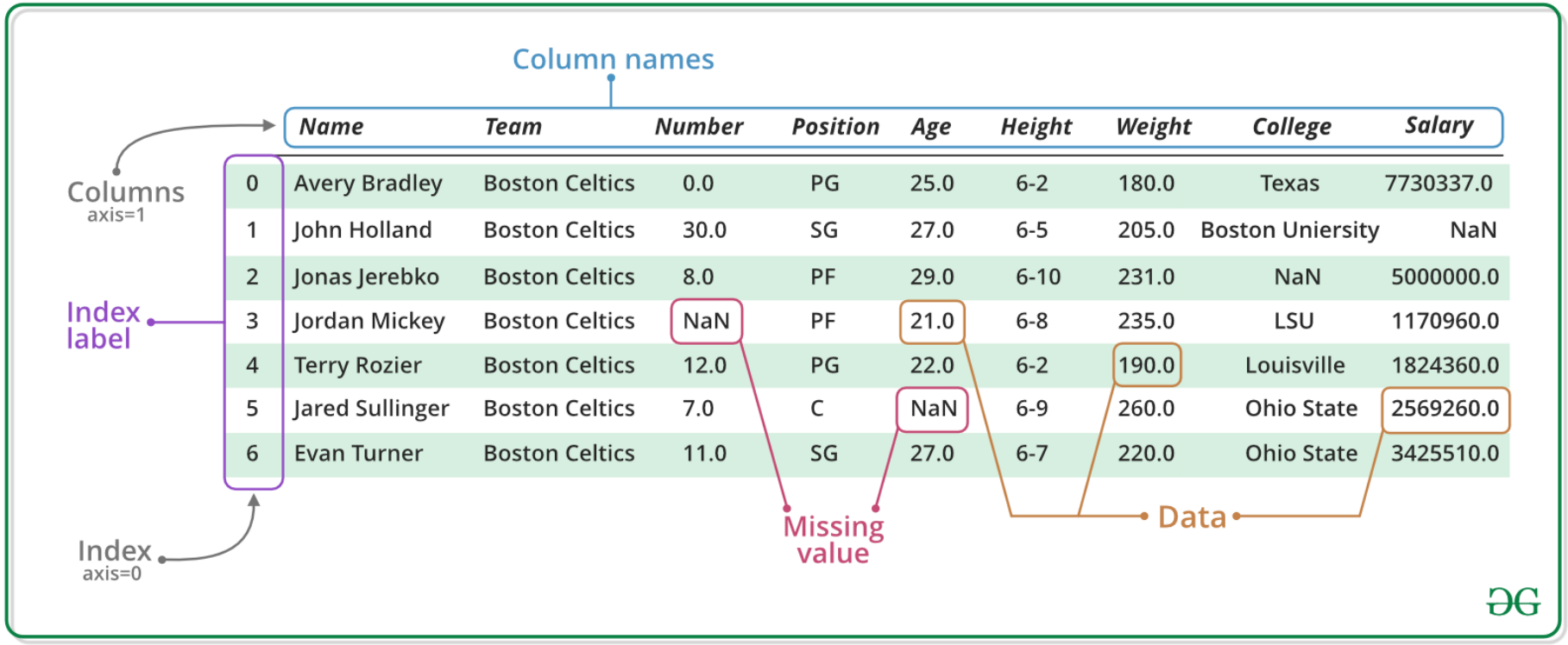
Object oriented programming

- **Working with objects before**
  - List, dictionary, set, string
  - They just have special constructors

Pandas

- **Open source python package for data analysis**
- **Used for data exploration, cleaning and transformation/manipulation**
  - "… a more sophisticated Excel"
- **Works with data series (one dimensional array with indices) and *dataframes* (table 2D)**
- **Pandas documentation can be found here: https://pandas.pydata.org/docs/index.html**
- **And some starter tutorial: https://www.w3schools.com/python/pandas/default.asp**

Pandas - Dataframe



	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston Uniersity	NaN
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0	6-8	235.0	LSU	1170960.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN	6-9	260.0	Ohio State	2569260.0
6	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0

Pandas – load your data

- **Pandas supports reading of a variety of file formats like excel, csv, tsv, json, hdf, html, sql, ...)**
- **We focus on the three most used:**

```
# Excel
df_sheet = pd.read_excel(„sample.xlsx“)
# CSV
df_csv = pd.read_csv(“sample.csv“)
# JSON
df_json = pd.read_json(“sample.json“)
```

- **The „read“ functions comes with a lot of parameters:**
  - Excel: `sheet_name(1)` or `sheet_name(“sheet_name1“)` or `sheet_name(None)`
  - All: `decimal = “,“` or `decimal = “.“`
  - Read all possibilities up in the docs!

Pandas – check your data

- **Always check if your file was loaded correctly.**
- **Either look at the whole dataframe:**

```
print(df)
```

- **But much more recommended:**

```
print(df.head()) # prints first 5 rows
```

- **Check if the columns are right:**

```
print(df.columns)
```

- **Check what your dataframe contains:**

```
print(df.info())
```

Pandas - Dropping

- **Dropping missing values from the whole dataframe:**

```
df.dropna()
```

- **Drop missing values from specific columns:**

```
df.dropna(subset = [“col1“, “col2“, “col3“, ...])
```

- **Remove all columns containing more than 20% NaNs**

```
df_20_missing = df.dropna(axis=1, thresh = int(0.8*df.shape[0]))
```

- **Delete rows with specific value in a specific column:**

```
new_df = df[df.column1 != 0]
```

- **Delete a specific column:**

```
new_df = df.drop(„column1“, axis = 1)
```

Pandas - Imputation

- **We can impute all missing values with the function fillna()**
- **With scalar:**

```
df.fillna(0)
df.fillna(“spongebob“)
```

- **With object:**

```
df.fillna(df.mean()) # (median, sum, ,...)
```

Pandas - Correlation

- **Pandas can calculate correlation on the whole dataframe, automatically excluding all NA/null values**
- **Different methods available: pearson , kendall and spearman**
- **Returns a dataframe with the correlation matrix**
- **Correlation within a dataframe:**

```
corr = df.corr(method="pearson", numeric_only=False)
```

- **Correlation with another dataframe:**

```
df.corrwith(another_df, method="pearson")
```

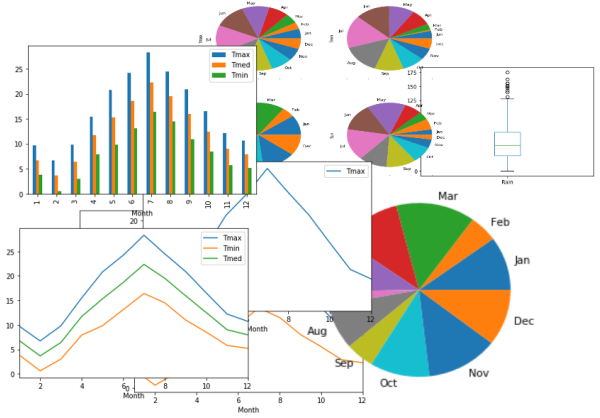
- **In jupyter notebook, we can visualize the matrix as a heatmap with**

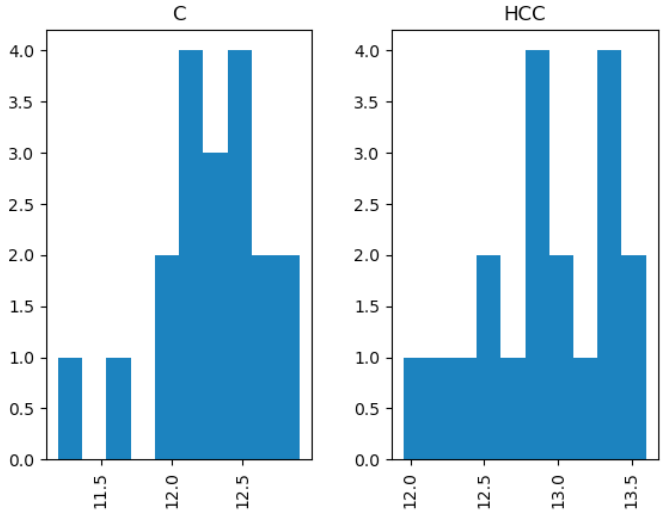
```
corr.style.background_gradient(cmap="coolwarm")
```

Pandas – Plots

- **Pandas supports multiple plot types like bar, hist, box, density, area, scatter and pie plots:**

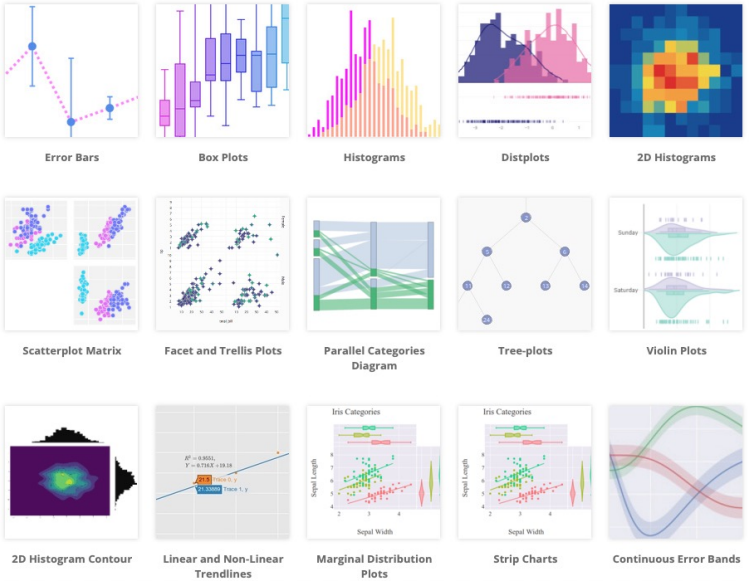
```
df.plot.hist() or df.hist()

df.plot.box() or df.boxplot()

df.plot.pie(subplots=True, figsize=(8,4));
```





Plotly (Why? ´Cause it is beautiful)

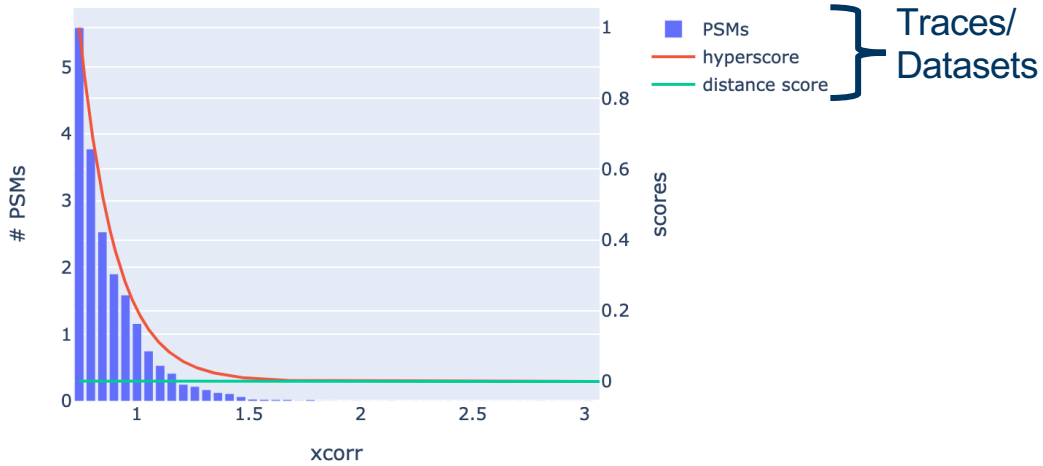
- **Plotly is a graphical library that supports multiple programming languages like Python, R, Julia, Javascript, F#...**
- **Can create**
  - Interactive plots (Jupyter Notebook, HTML/JS)
  - Static images (SVG, PNG, JPEG, PDF, …)
  - JSON for interchange
    - Languages (Python □ R)
    - Server □ Client
- **Can also be created from gglots2!**



Plotly (Why? ´Cause it is beautiful)

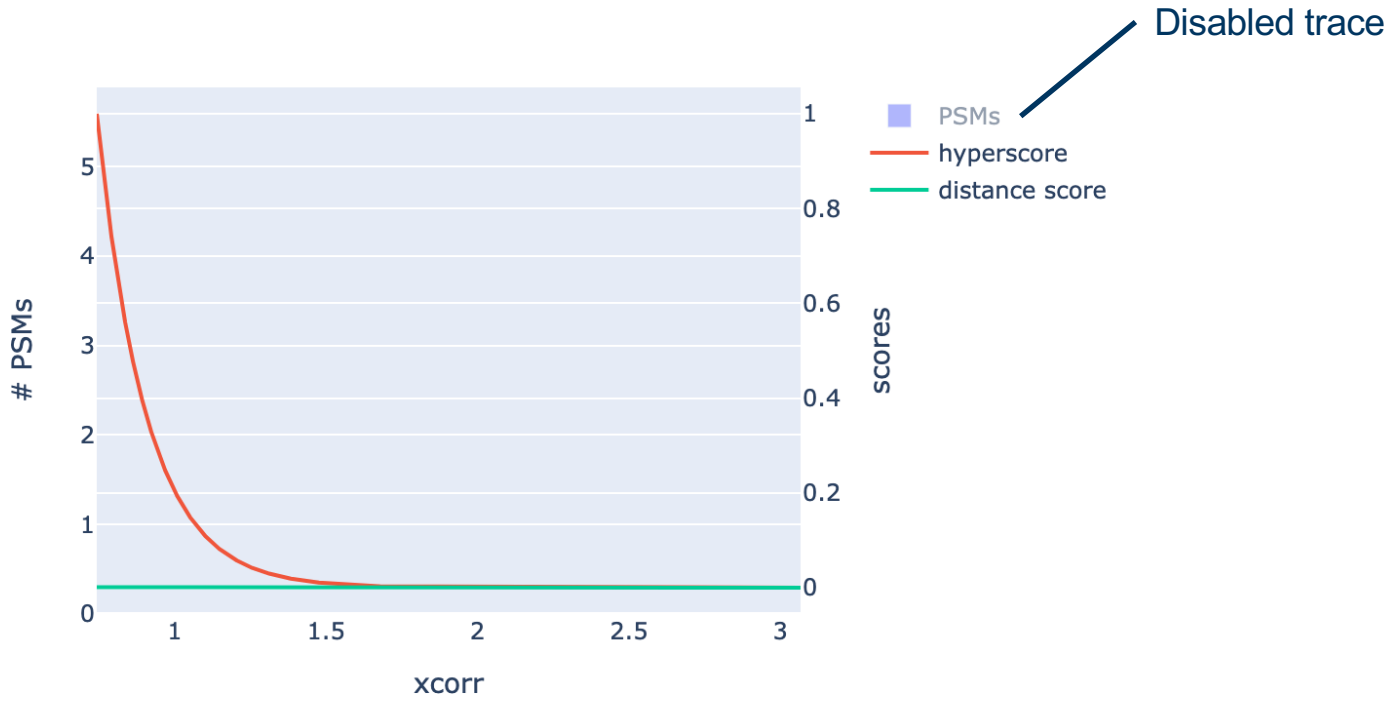
Interactivity!

Controls (zoom, download, …)



Plotly (Why? ´Cause it is beautiful)

Interactivity!



Plotly (Why? ´Cause it is beautiful)

```
import plotly.graph_objects as go
fig = go.Figure()
fig.add_trace(go.Bar(
    x = df[“a_col”],
    y = df[“another_col”]
))
fig.show()
fig.write_image("fig1.png")
```

Sweetviz

- **Open-source Python library to „kickstart“ Exploratory Data Analysis (EDA)**
- **High-density visualization**
- **HTML or jupyter notebook**
- **Compare 2 Dataframes or 2 subsets of the same dataframe**
- **More libraries like this:**
  - Dtale
  - Autoviz
  - YData Profiling

Exercise 4

https://l.rub.de/3a11cda2

Scientific libraries

- **scipy**
  - **Mathematical functions (algebra, interpolation, distributions, ...)**
- **scikit-learn**
  - **„Classic“ machine learning (Random Forest, SVM, ...)**
- **PyTorch / Keras**
  - **Deep Learning**
    - **Layers types: Dense, Convolution, LSTMs, …**
    - **Activation functions: ReLU, Sigmoid, …**
    - **Optimizers: SDG, Adam, Nadam, …**

Where to go?

- **Get more proficient in object oriented programming**
  - Dataclasses & slots
  - Properties
- **Dependency management**
  - You need to make your code installable and reproducible. Very important for reviewers
- **GIT**
  - Helps you to keep track of your changes as code will improve iteratively
  - Make your code collaboratively available on GitHub / GitLab etc.
  - The wrong way: Nextcloud, Google Cloud, Dropbox, Word …

Where to go?

- **(Unit-) Tests**
  - Automatic tests will help you keep your code functional after changes
- **Practice practice practice**
  - Automate small task
  - Leet code: https://leetcode.com/
    - Provides you with small code challenges and checks your results