



Politecnico
di Torino



Data Science Lab

Introduction to Python

Andrea Pasini
Flavio Giobergia
Elena Baralis

DataBase and Data Mining Group



- **Python engine**
 - Basic components and setup
- **Python language**
 - Data types, object oriented programming
- **Numpy library**
 - Computation with multi-dimensional arrays
- **Pandas library**
 - Tabular data and data preprocessing
- **Scikit-Learn library**
 - Machine learning and data science tools



- Python language
 - Clean and concise syntax
 - No semi-colons to end instructions
 - No braces to define if clauses and for loops
 - No need to specify variable types
 - ...

Java

```
List<String> l = new LinkedList<>();  
for (int i=0; i<10; i++) {  
    l.add(i);  
}
```

Python

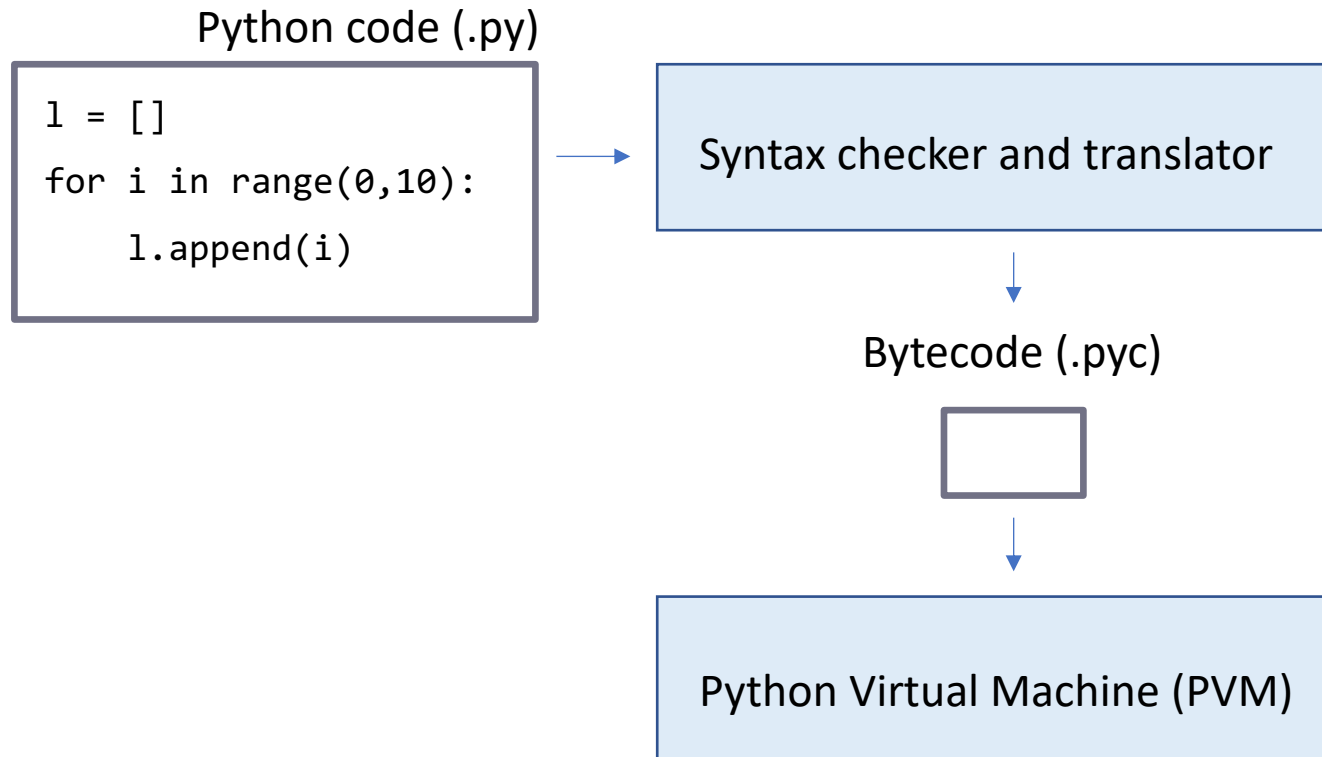
```
l = []  
for i in range(0,10):  
    l.append(i)
```



- Python is an **interpreted** language
 - Code is not compiled to machine language
 - However the source code is compiled to an intermediate level, called **bytecode**
 - For this reason, to run Python programs, you need an **interpreter** that is able to execute the bytecode



- Sequence of operations executed by the interpreter



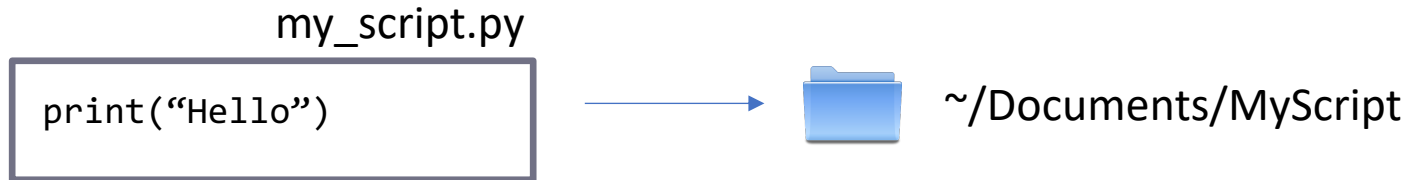


- A common Python 3 setup on a **Linux** System
- Typically in the `/usr/bin` folder:
 - “**python3**” executable: run Python programs
 - “**pip3**” executable: install Python packages
 - “**ipython3**” executable: run programs line by line
 - “**jupyter**” executable: run a jupyter notebook
- To find where your python commands live:
 - `$ which <command>`

```
[fgiobergia@localhost $ which python3  
/usr/local/bin/python3  
fgiobergia@localhost $ █
```



- Executing a Python program



- Type in your terminal:
 - `cd ~/Documents/MyScript`
 - `python3 my_script.py`



- Running Python line by line with IPython
- Type in your terminal:
 - `ipython3` (or `ipython`, depending on your installation)

```
IPython: home/andrea
File Modifica Visualizza Cerca Terminale Aiuto
andrea@andrea:~$ ipython3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
Type "copyright", "credits" or "license" for more information.
```




- Write your program line by line to see the results step by step...

```
IPython: home/andrea
File Modifica Visualizza Cerca Terminale Aiuto

andrea@andrea:~$ ipython3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
Type "copyright", "credits" or "license" for more information.

IPython 5.5.0 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
%quickref        -> Quick reference.
help             -> Python's own help system.
object?         -> Details about 'object', use 'object??' for extra details.

In [1]: mystring = "hello"

In [2]: print(mystring)
hello

In [3]:
```



- **Python** and **IPython** programs are the core for executing scripts, but...
- There are two typical scenarios:
 1. Develop your Python **project** with an **IDE**
 - Example: Visual Studio Code, PyCharm
 - **Debug** and **run** your code inside the IDE
 2. Develop and test a Python **script** with **Jupyter notebook**
 - Inspect **step by step** the results
 - Keep the history of the output of the script

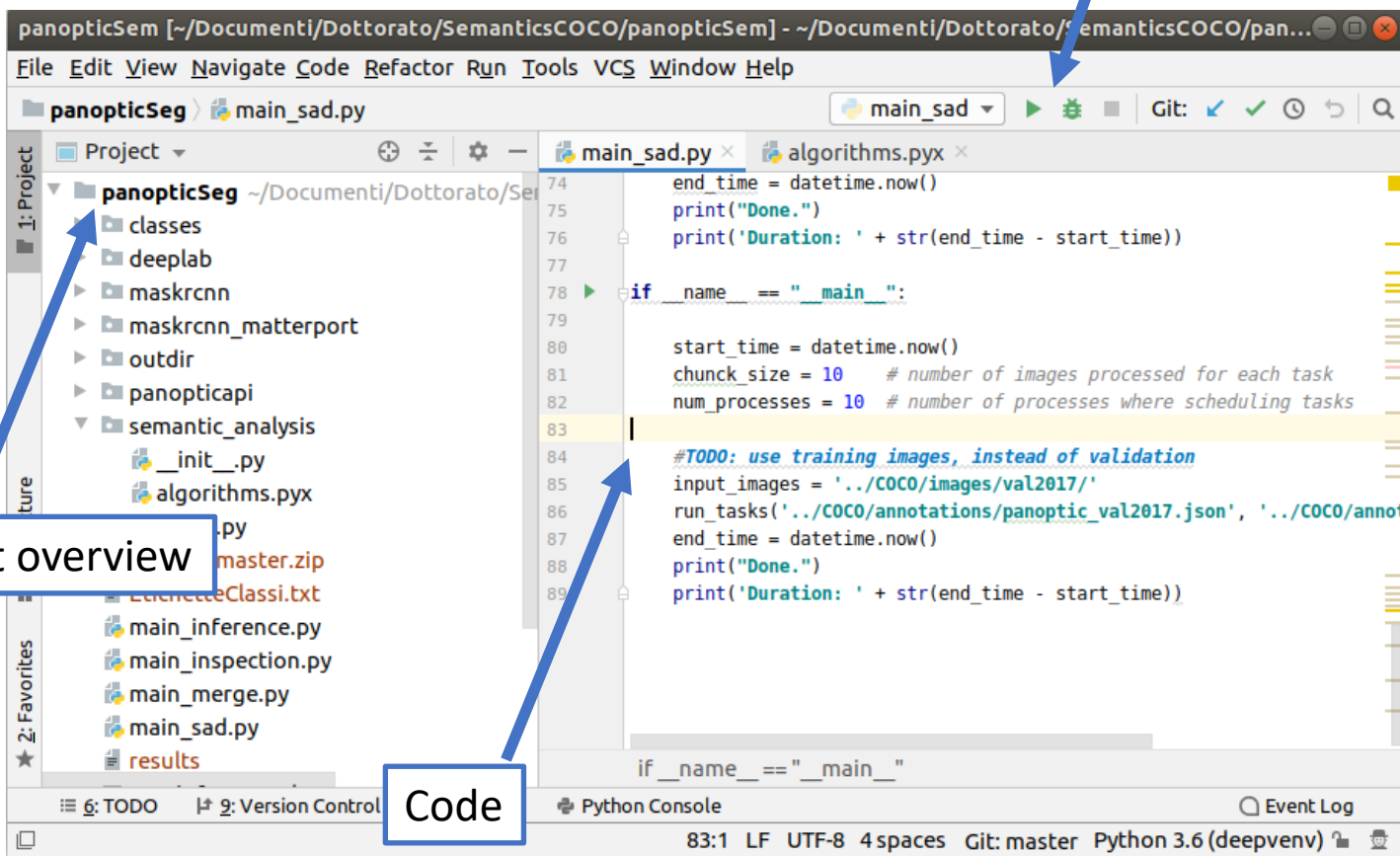


■ Scenario 1: PyCharm (IDE)

Run/Debug commands

Project overview

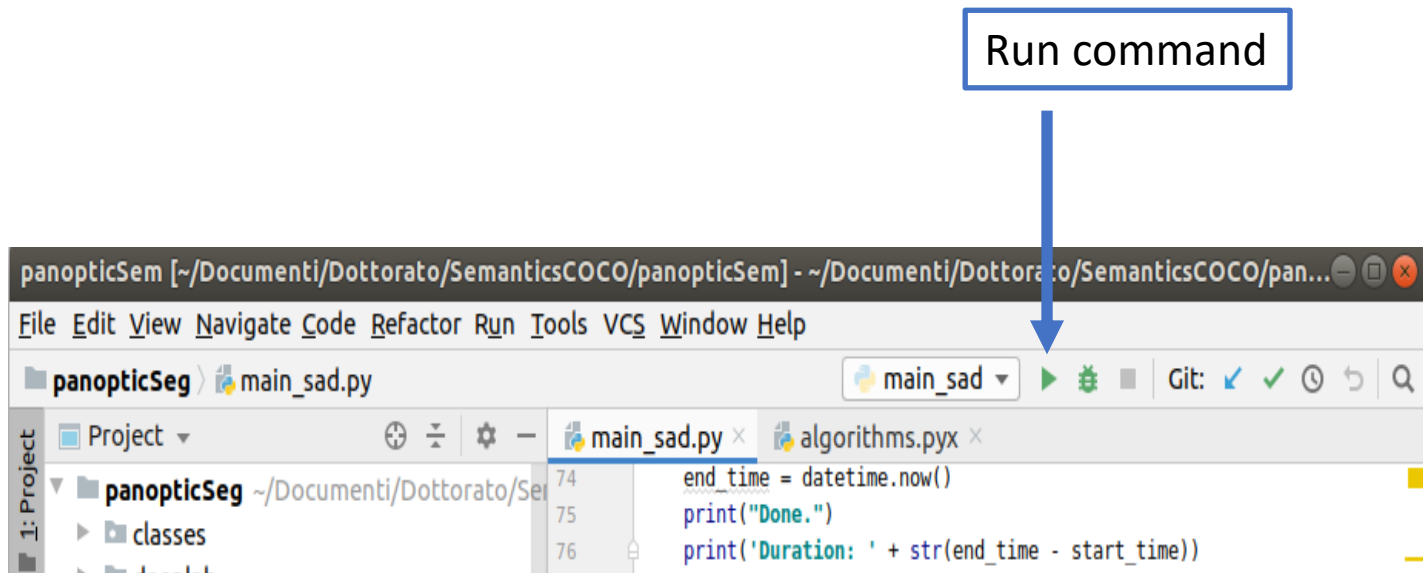
Code





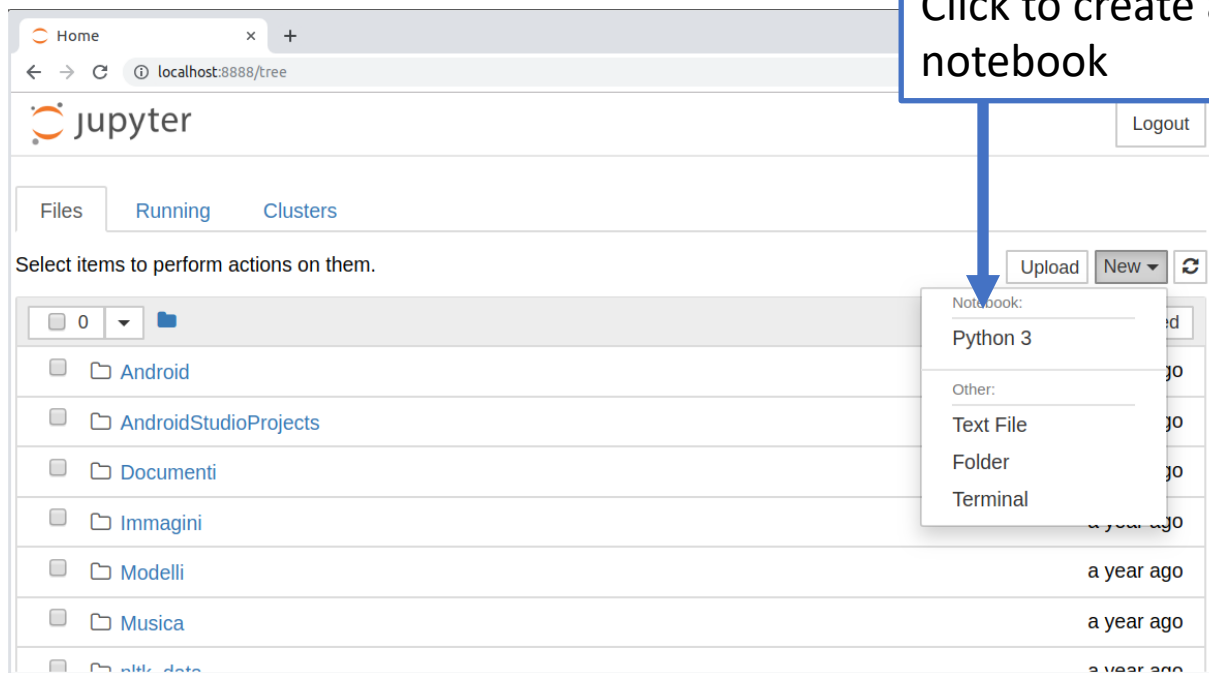
■ Scenario 1: PyCharm (IDE)

- When you click on the run button, the IDE will automatically call the “**python**” command to execute your script





- **Scenario 2: Jupyter notebook**
 - Type in your terminal
 - jupyter notebook
 - Jupyter will open on your browser





■ Scenario 2: Jupyter notebook

The screenshot shows a Jupyter notebook interface. At the top is a toolbar with icons for file operations, running, and code execution. Below the toolbar, the notebook content is organized into cells. A text cell contains the heading "1. Simple linear regression" and the sub-heading "Generating a dataset". Below this is a code cell containing Python code for generating a dataset. The code is as follows:

```
In [26]: # Make dataset
err = np.random.normal(0,1, 100) # gaussian data, mean=0, std=1
x = 10*np.random.rand(100) # 100 data points in [0, 10]
y = (2*x + 2) + err # target is a linear function of the input with some noise
```

Below the code cell is another code cell:

```
In [27]: # Plots
plt.scatter(x, y, s=10, c='grey')
plt.show()
```

At the bottom of the notebook is a plot cell displaying a scatter plot of the generated data. The x-axis ranges from 0 to 10, and the y-axis ranges from 0 to 20. The plot shows a clear positive linear correlation between the two variables, with some scatter around the line. Three blue arrows point from text boxes to specific parts of the notebook: "Markdown cell" points to the heading, "Code cell" points to the first code cell, and "Result cell" points to the scatter plot.



- **Scenario 2: Jupyter notebook**
 - Based on **IPython** command
 - Each code **cell** can be executed **separately** by pressing CTRL + ENTER



1. Simple linear regression

Generating a dataset

```
In [26]: # Make dataset
err = np.random.normal(0,1, 100) # gaussian data, mean=0, std=1
x = 10*np.random.rand(100)      # 100 data points in [0, 10]
y = (2*x + 2) + err             # target is a linear function of the i
```

```
In [27]: # Plots
plt.scatter(x, y, s=10, c='grey')
plt.show()
```

Code cell 1

Code cell 2



IDE vs Jupyter notebook

■ IDE

- For more **complex** projects (many files)
- More powerful debug commands
- More powerful code editing tools

■ Jupyter notebook

- For simple scripts and prototypes
- Great **visualization** tool
 - Example: **report** with Python code and text for explanations



■ Installing libraries

- Python language is provided with many useful libraries:
 - Numpy, Pandas, Matplotlib, Scikit-learn, SciPy, ...
- To use any of them you first have to install it with the **pip** command: `pip3 install <package>`
 - `pip3 install numpy`
 - `pip3 install pandas`

```
andrea@andrea
File Modifica Visualizza Cerca Terminale Aiuto
andrea@andrea:~$ pip3 install numpy
```



■ Virtual environments

- The pip command will associate the libraries to your **default Python installation**
- A more powerful way of managing libraries is to use a Python **environment (virtualenv)**
 - Designed when you want to design **different projects** that use different libraries and **configurations (e.g. versions)**
 - Each projects is associated to a virtual environment



■ Virtual environments

- To create a new environment:
 - `cd ~/Documents/My_project`
 - `virtualenv myenv`
- It will create a new environment in your project folder

```
Documenti MyProject
myenv my_script.py
andrea@andrea-XPS-13-9360: ~/Documenti/MyProject
File Modifica Visualizza Cerca Terminale Aiuto
andrea@andrea:~/Documenti/MyProject$ virtualenv myenv
Running virtualenv with interpreter /usr/bin/python2
New python executable in /home/andrea/Documenti/MyProject/myenv/bin/python2
Also creating executable in /home/andrea/Documenti/MyProject/myenv/bin/python
Installing setuptools, pkg_resources, pip, wheel...done.
andrea@andrea:~/Documenti/MyProject$
```



■ Virtual environments

- To **activate** the created environment:
 - `cd ~/Documents/My_project`
 - `source myenv/bin/activate`

```
File Modifica Visualizza Cerca Terminale Aiuto  
andrea@andrea:~/Documenti/MyProject$  
(myenv) andrea@andrea:~/Documenti/MyProject$
```



■ Virtual environments

- After activation you can use the terminal to work within the environment

```
File Modifica Visualizza Cerca Terminale Aiuto  
andrea@andrea:~/Documenti/MyProject$  
(myenv) andrea@andrea:~/Documenti/MyProject$
```

- Install libraries in the *current* environment
 - pip3 install my_library
- Execute a script/notebook within the environment
 - python3 my_script.py
 - jupyter notebook