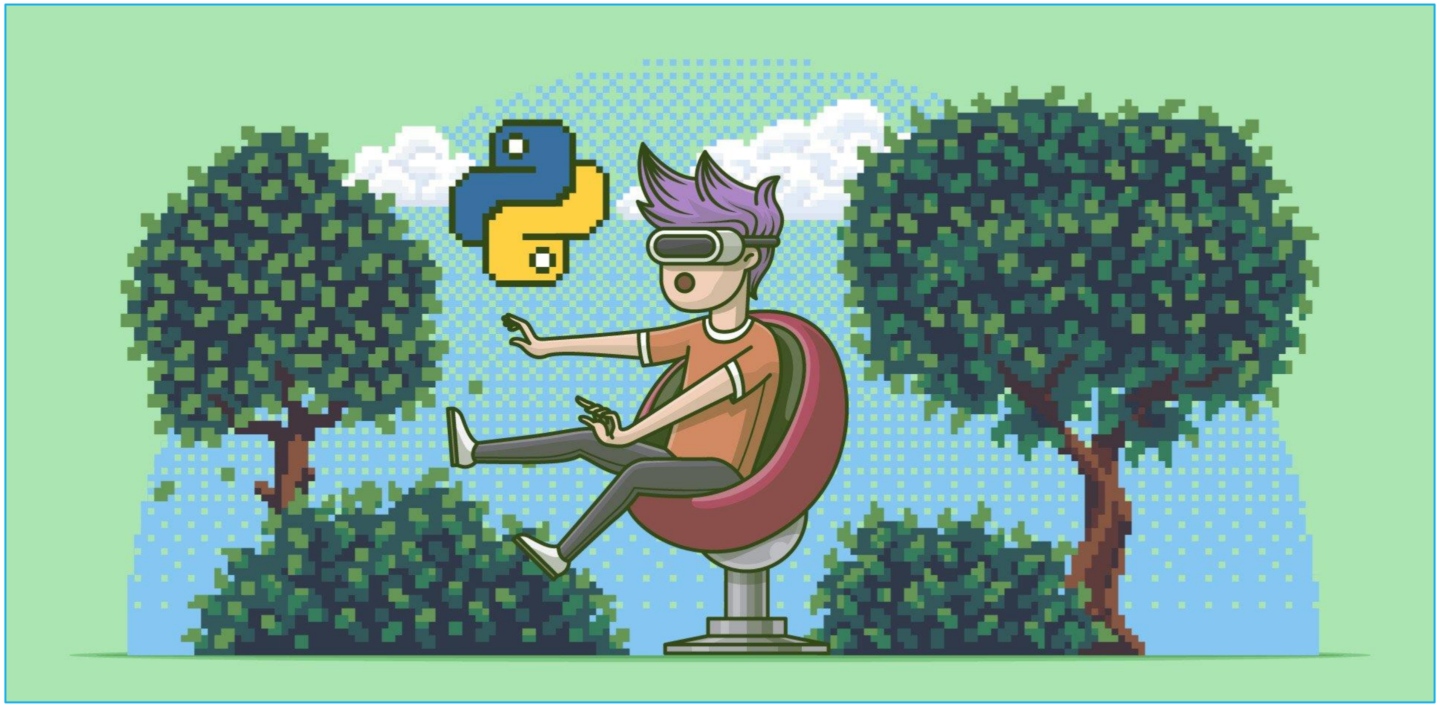
PYTHON AND PYCHARM

Antonio Luca Alfeo

INTRODUCTION TO PYTHON

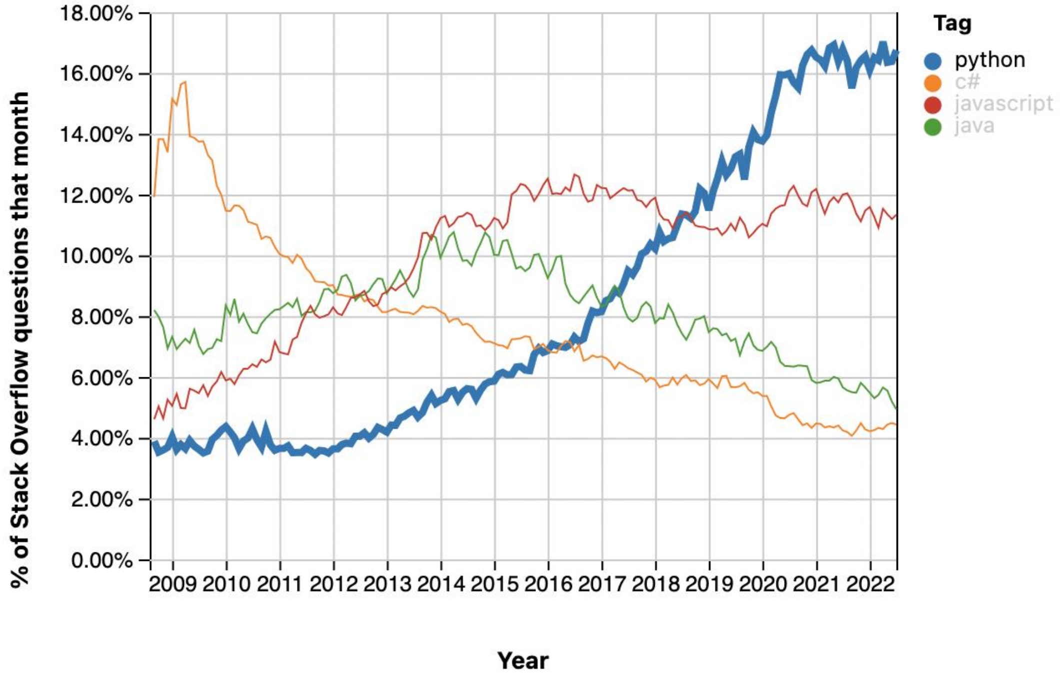


Based on previous lecture by F. Galatolo

WHY PYTHON?

Created by **Guido van Rossum** in the 1980s, to be a general-purpose language with a simple and readable syntax. Today is more and more required since it is:

- High level
- Open source
- Portable: write once run everywhere
- Extendible in C/C++
- Easy to learn
- With a mature and supportive community
- With hundreds of thousands of libraries, packages and frameworks, supported by big players (Google, Facebook, Amazon) and non-profit organizations



VIRTUAL ENVIRONMENT

- In each project, a number of Python packages are imported and used. Each of them may requires a given version of other packages and Python as well.
- A **virtual environment** is a self-contained directory tree that contains a Python installation for a particular version of Python, plus a number of additional packages
- In this way, the project-wide dependencies are stored, easily snapshotted and retrieved
- Create a Virtual Environment with Python X.Y in folder env

```
virtualenv --python=pythonX.Y env
```

- Activate the Virtual Environment

```
source ./env/bin/activate
```

```
. ./env/bin/activate
```

- Deactivate the Virtual Environment

```
deactivate
```

BASIC PACKAGES MANAGING

- Install package

```
pip install package
```

- Uninstall package

```
pip uninstall package
```

- Snapshot installed packages in *requirements.txt*

```
pip freeze > requirements.txt
```

- Install all packages snapshotted in *requirements.txt*

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

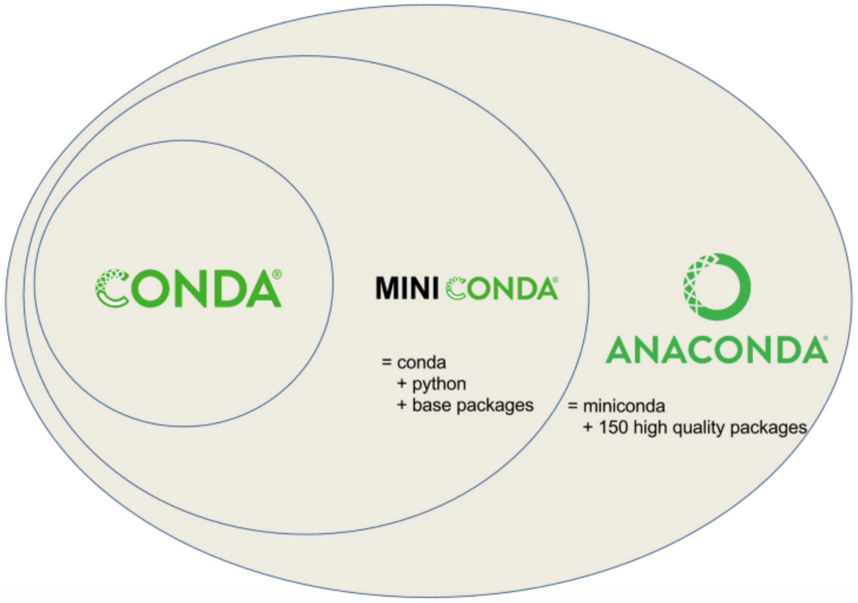
- Now you can import and use this packages in your project

ANACONDA

Anaconda is a distribution of Python that aims to simplify package management and deployment, suitable for Windows, Linux, and macOS.

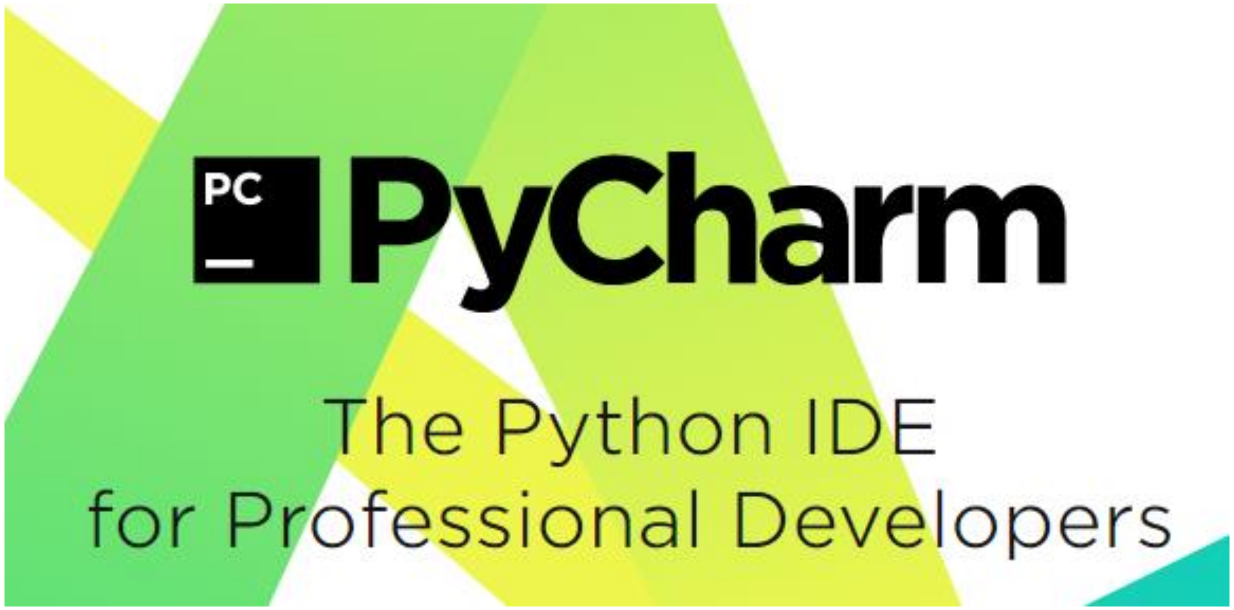
Package versions in Anaconda are managed by **conda**, an open source, cross-platform, language-agnostic package manager and environment management system

Conda analyses the current environment including everything currently installed, works out how to install/run/update a compatible set of dependencies



PYTHON IDE + MINICONDA = PYCHARM

Pycharm offers configurable python interpreter
and virtual environment support.



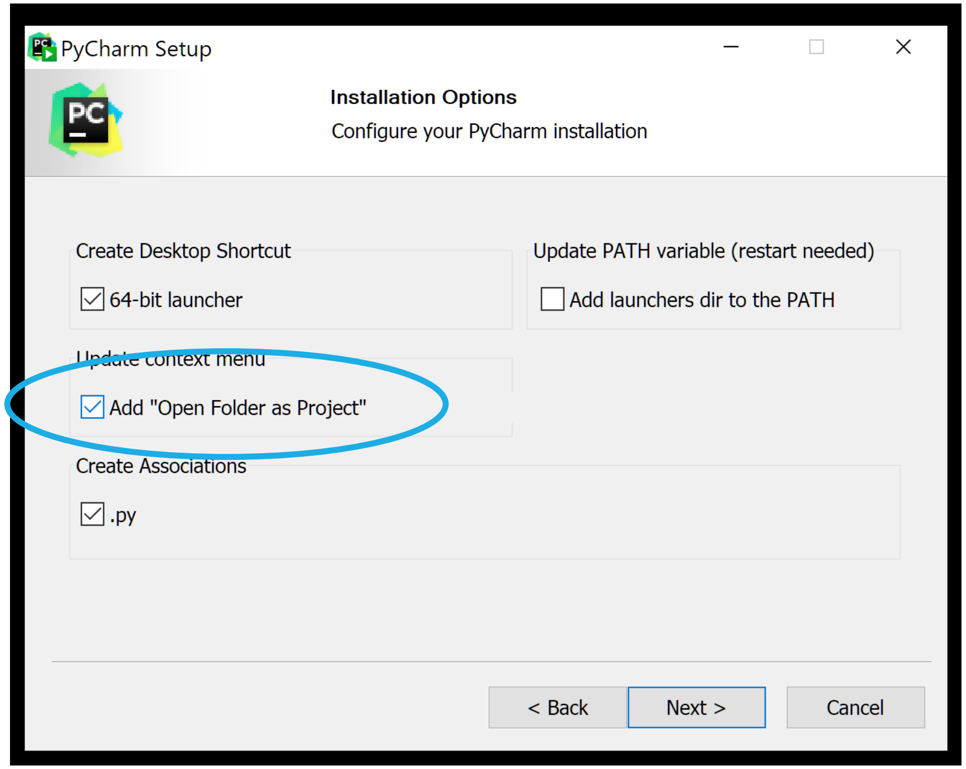
Based on previous lecture by A. L. Alfeo

INSTALLATION 1/2

1. Install **PyCharm** using this links:
   - Linux
   - Windows
   - MacOs

2. During PyCharm installation

   enable “open folder as project”



PyCharm Setup

Installation Options

Configure your PyCharm installation

Create Desktop Shortcut
- [✓] 64-bit launcher

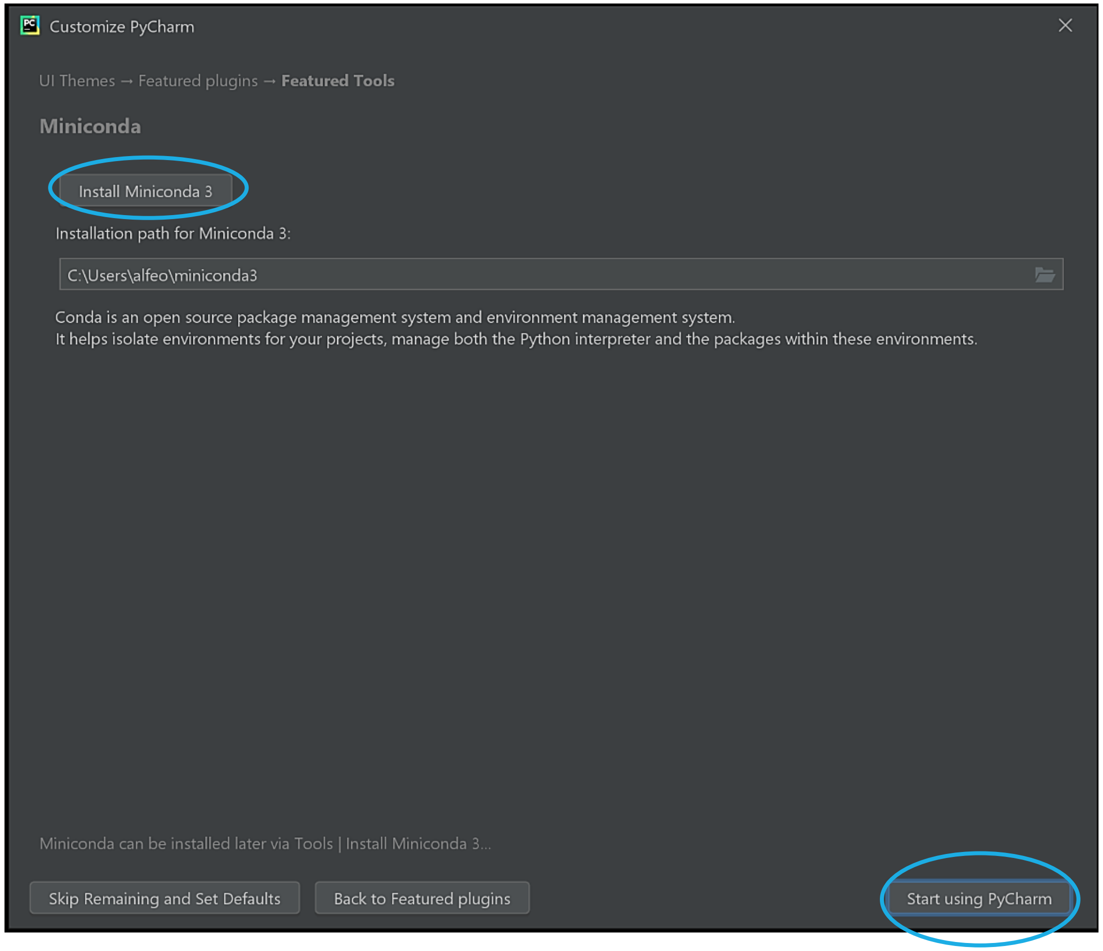
Update PATH variable (restart needed)
- [ ] Add launchers dir to the PATH

Update context menu
- [✓] Add "Open Folder as Project"

Create Associations
- [✓] .py

INSTALLATION 2/2

- Accept the JetBrains Privacy Policy
- Choose the UI theme you prefer
- Do not install any featured plugin
- Install Miniconda: includes the conda environment manager, Python, the packages they depend on, and a small number of other useful packages (e.g. pip).
- Remember, Miniconda can be installed at any time from **Tools -> Install Miniconda3**
- Start using PyCharm


Customize PyCharm

UI Themes → Featured plugins → **Featured Tools**

Miniconda

Install Miniconda 3

Installation path for Miniconda 3:

C:\Users\alfeo\miniconda3

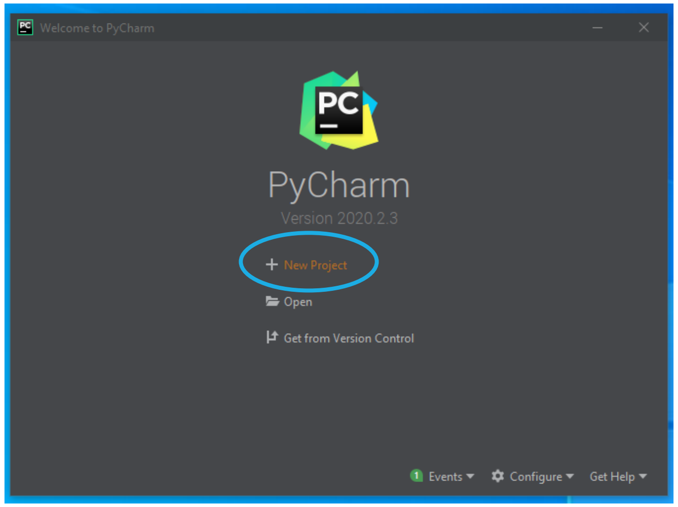
Conda is an open source package management system and environment management system.
It helps isolate environments for your projects, manage both the Python interpreter and the packages within these environments.

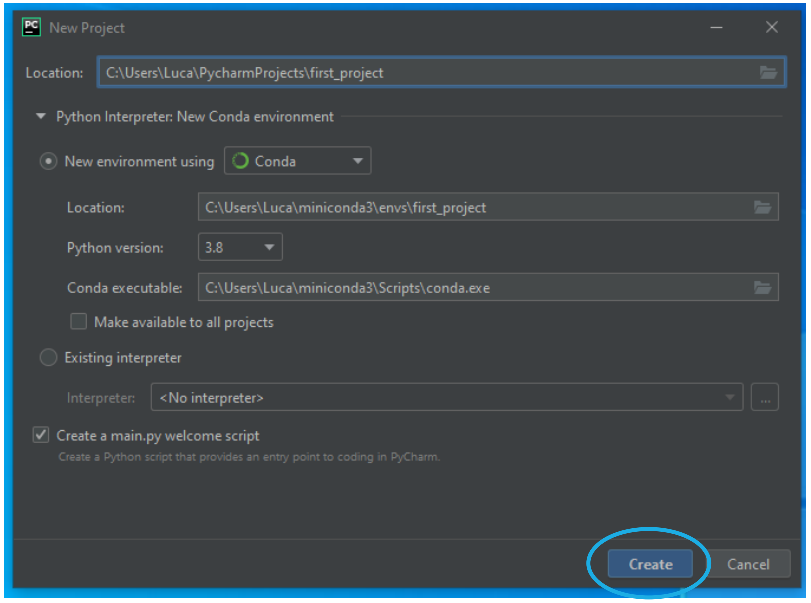
Miniconda can be installed later via Tools | Install Miniconda 3...

NEW PROJECT

Create a new project. Name it "first_project".

If needed set the **conda executable path**.




New Project

Location: C:\Users\Luca\PycharmProjects\first_project

Python Interpreter: New Conda environment

(●) New environment using Conda

Location: C:\Users\Luca\miniconda3\envs\first_project

Python version: 3.8

Conda executable: C:\Users\Luca\miniconda3\Scripts\conda.exe

☐ Make available to all projects

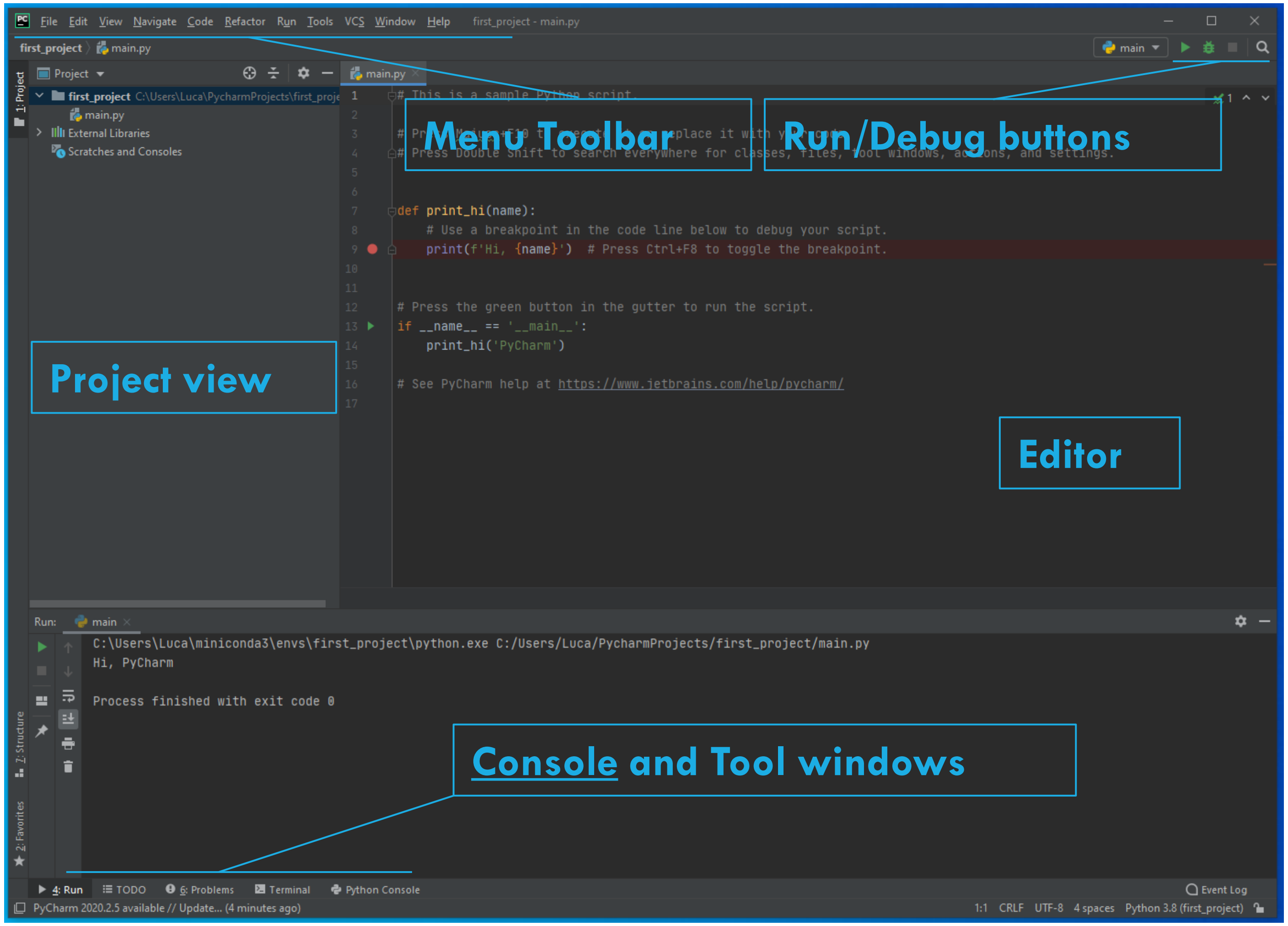
( ) Existing interpreter

Interpreter: <No interpreter>

☑ Create a main.py welcome script

Create a Python script that provides an entry point to coding in PyCharm.

GUI

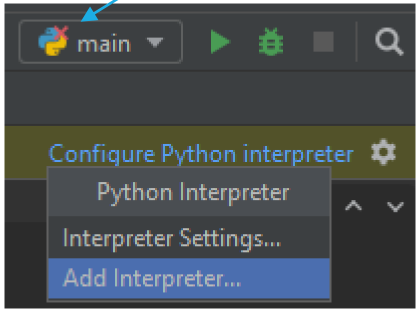

File Edit View Navigate Code Refactor Run Tools VCS Window Help first_project - main.py
first_project main.py
main
Project
first_project C:\Users\Luca\PycharmProjects\first_proje
main.py
External Libraries
Scratches and Consoles
main.py
Menu Toolbar
Run/Debug buttons
Project view
Editor
Console and Tool windows
Run: main
4: Run TODO 6: Problems Terminal Python Console Event Log
PyCharm 2020.2.5 available // Update... (4 minutes ago)
1:1 CRLF UTF-8 4 spaces Python 3.8 (first_project)

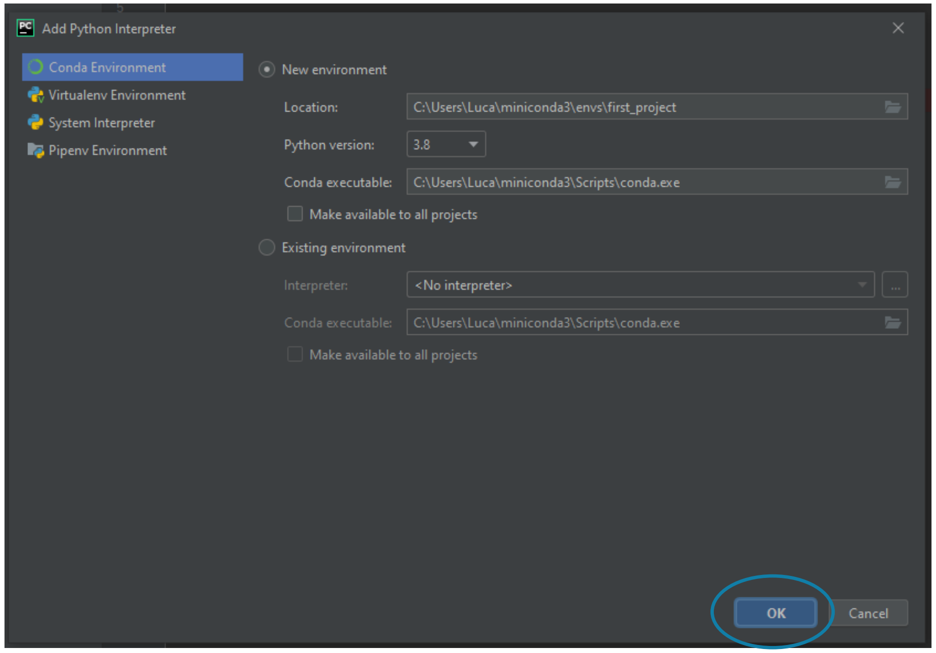
```
# This is a sample Python script.

# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool windows, actions, and settings.


def print_hi(name):
    # Use a breakpoint in the code line below to debug your script.
    print(f'Hi, {name}')  # Press Ctrl+F8 to toggle the breakpoint.


# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    print_hi('PyCharm')

# See PyCharm help at https://www.jetbrains.com/help/pycharm/
```

```
C:\Users\Luca\miniconda3\envs\first_project\python.exe C:/Users/Luca/PycharmProjects/first_project/main.py
Hi, PyCharm

Process finished with exit code 0
```

INTERPRETER: DEFAULT CONFIGURATIONS

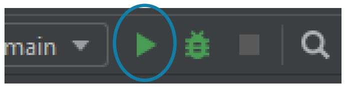
Is there a red X here?





New environment
Location: C:\Users\Luca\miniconda3\envs\first_project
Python version: 3.8
Conda executable: C:\Users\Luca\miniconda3\Scripts\conda.exe
Make available to all projects
Existing environment
Interpreter: <No interpreter>
Conda executable: C:\Users\Luca\miniconda3\Scripts\conda.exe
Make available to all projects

Now you can run it!



PYTHON BASICS

```
A dog contro
"""
image = games.load_image("kg.png")

def __init__(self):
    """ Initialize Dog object and create Text o
    super(Dog, self).__init__(image = Dog.image
                              x = games.mouse.x
                              bottom = games.sc

    self.score = games.Text(value = 0, size =
                            top = 5, right = ga
    add(self.score)
         (value = 0, size =
                 left = gam
```

INDENTATION

In python **code blocks are not surrounded by curly brackets**. Just use the correct **indentation**!

```
for(int i = 0; i < n; i++){
    int k = i % 3
    if(k == 0){
        // stuff...
    }
}
```

```
for i in range(0, n):
    k = i % 3
    if k == 0:
        # stuff...
```

BUILT-IN TYPES

Python is **dynamically typed**, i.e. types are determined at runtime.

Variables can **freely change type during the execution**.

In python there are a lot of built-in types, the most notables are:

- **Boolean** (*bool*)
- **Strings** (*str*)
- **Numbers** (*int*, *float*)
- **Sequences** (*list*, *tuple*)
- **Mapping** (*dict*)

VARIABLES

Variables can be assigned to given values…

```
pi = 3
name = “Pippo"
```

…Even multiple variables at once via **iterable unpacking**. Click for more on iterable data structures (e.g. lists, tuples…)!

```
pi, name = 3, “Pippo"
first, second, third = SomeSequence
```

In Python everything is stored and passed as **reference** with the only exception of Numbers.

```
a = [1, 2, 3]
b = a
b[0] = 5 # now a = [5, 2, 3]
```

CONDITIONAL INSTRUCTION 1/2

Simple conditional instruction with the **if** keyword.

```
if someConditions:
    someActions()
    someOtherActions()
```

Python uses **and** and **or** as logical operators instead of && and ||

```
if (C1 and C2) or C3:
    someActions()
    someOtherActions()
```

Inline conditional instructions can be used as it follows

```
value if Condition else otherValue
```

CONDITIONAL INSTRUCTION 2/2

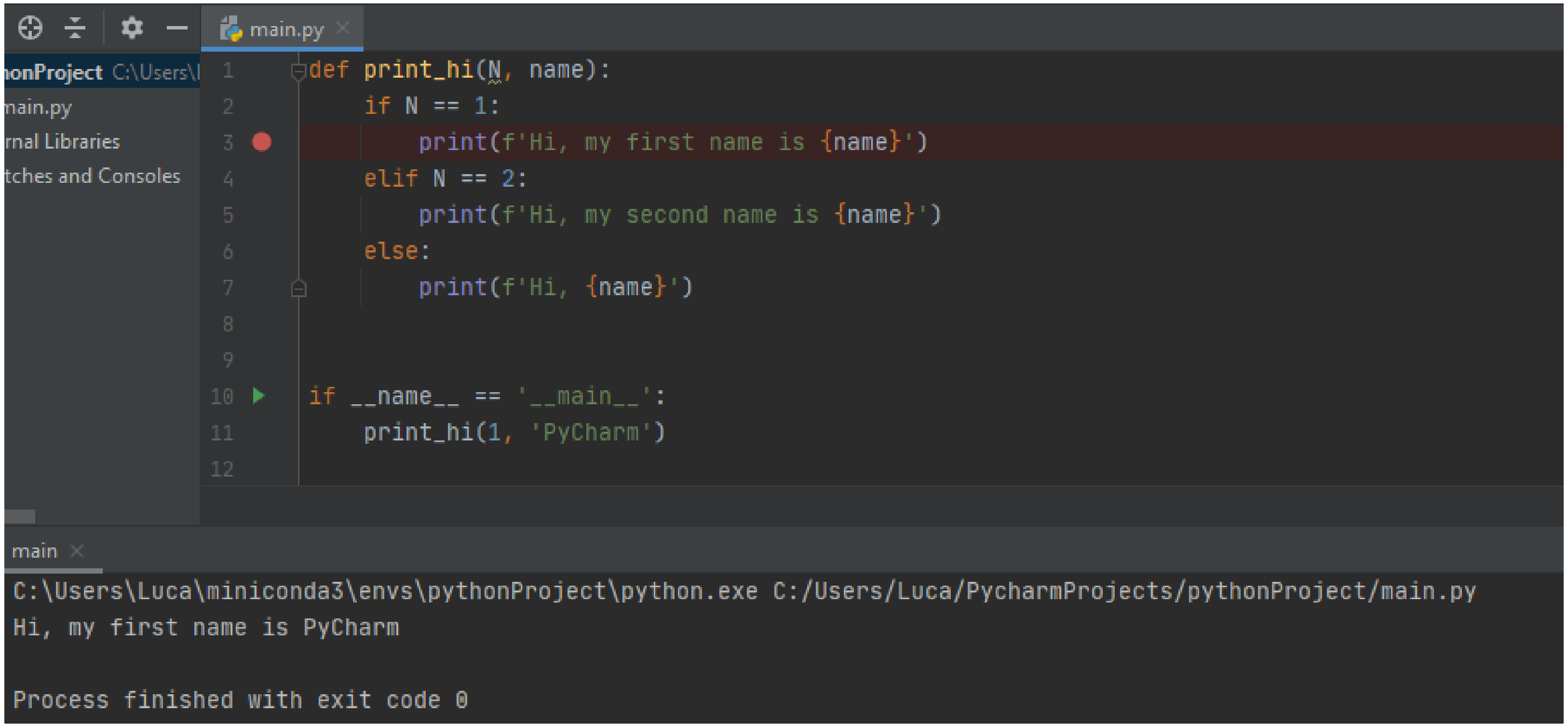
The **if-else** statement is used as it follows

```
if Condition:
    someActions()
else:
    someOtherActions()
```

There is no switch case statement in Python. Just use **if** and **elif**

```
if C1:
    A1()
elif C2:
    A2()
elif C3:
    A3()
else:
    A()
```

TRY IT YOURSELF!



```python
def print_hi(N, name):
    if N == 1:
        print(f'Hi, my first name is {name}')
    elif N == 2:
        print(f'Hi, my second name is {name}')
    else:
        print(f'Hi, {name}')


if __name__ == '__main__':
    print_hi(1, 'PyCharm')
```

```
C:\Users\Luca\miniconda3\envs\pythonProject\python.exe C:/Users/Luca/PycharmProjects/pythonProject/main.py
Hi, my first name is PyCharm

Process finished with exit code 0
```

LOOPS 1/2

There is no do-while loop, just **while** and **for** loops

```
while Conditions:
    Stuff()
    otherStuff()
```

```
for element in elements:
    doStuff(element)
```

Tuple unpacking can be used with loops iterations

```
for x, y in SequenceOfTuples:
    doStuff(x, y)
```

LOOPS 2/2

zip() combines one-by-one the elements of two or more iterables

```
L1 = [1, 2, 3]
L2 = [4, 5, 6]
for x, y in zip(L1, L2):
    print(x, y)
```

enumerate() returns a list of *(index, element)* tuples

```
names = ["Federico", "Mario", "Giovanni"]
for i, name in enumerate(names):
    print(i, name)
```

For efficient loops an iterable object can leverage the itertools packages.

The simplest iterable can be a **list** created via **list comprehension**

```
[someOperation(element) for element in elements]

squares = [i**2 for i in range(0, N)]
```

FUNCTIONS: DEFINITION

A new function can be defined by using the keyword **def**

```
def getCircleArea(r):
    return pi*r**2
```

Default argument are indicated with **=**

```
def getCircleArea(r, isEngineer=True):
    pi = 3 if isEngineer else 3.1415
    return pi*r**2
```

Inline functions can be created by using the **lambda** keyword

```
lambda comma, separated, arguments : expression
```

For example

```
norm2D = lambda x, y: math.sqrt(x**2 + y**2)
```

FUNCTIONS: POSITIONAL ARGUMENTS

A **variable** number of arguments can be defined via the * symbol

```
def sumOfSquares(*args):
    squares = [arg**2 for arg in args]
    return sum(squares)

result = sumOfSquares(1, 2, 3)
```

A **sequence** can be passed as positional arguments as it follows

```
def norm2D(x, y):
    return math.sqrt(x**2 + y**2)

vec = [2, 3]
norm = norm2D(*vec)
```

FUNCTIONS: KEYWORD ARGUMENTS

A function using keyword arguments needs to have the **** symbol as last argument**.

```
def greet(language = "en", **kwargs):
    if language == "it":
        print("Ciao "+kwargs["name"]+" "+kwargs["surname"])
    else:
        print("Hello "+kwargs["name"]+" "+kwargs["surname"])

greet("it", surname="Galatolo", name="Federico")
greet(name="Mario", surname="Cimino")
```

You can also pass a **dict** of keyword arguments using the symbol ** while calling the method

```
person = dict(name="Federico", surname="Galatolo")
greet("it", **person)
greet(**person)
```

TRY IT YOURSELF!

Modify the **print_hi** function to accept a **dict** as an argument. Print:

- the first *number* squares, if *number* is greater than zero
- "Just a zero?!" if *number* is zero
- «Hi, my name is *name*», otherwise

```
def print_hi(**kwargs):
    if kwargs["number"] > 0:
        squares = [i**2 for i in range(0, kwargs["number"])]
        for elem in squares:
            print(elem)
    elif kwargs["number"] == 0:
        print("Just a zero?!")
    else:
        print("Hi, my name is " + kwargs["name"])

if __name__ == '__main__':
    argv = dict(name="Luca", number=0)
    print_hi(**argv)
```

CLASSES: METHODS

In python classes are defined with the **class** keyword. Class methods are defined with the **def** keyword. **Class method** have the first argument equal to **self**, in contrast with **static method**.

```
class Person:
    def getName(self):
        return "Federico"
    def greet(self):
        return "Hi! I am "+self.getName()
```

```
class Person:
    greeting = "Hi!"
    def getGreeting():
        return Person.greeting

g = Person.getGreeting()
```

However, unless you want to use a decorator, Python does not know about static/non-static methods, **it is all about notation!**

```
p = Person()
p.greet() # ok
Person.greet(p) # still ok
```

CLASSES: ATTRIBUTES

In python you can create, modify and retrieve **instance attributes** using the dot (.) selector on the instance reference. You can create and assign an instance attribute everywhere in a class method.

```
class Person:
    def setName(self, name):
        self.name = name
    def greet(self):
        return "Hi! I am "+self.name
```

You can create **class attributes** specifying them after the class declaration. You can modify and retrieve class attributes using the dot (.) selector on the class reference

```
class Person:
    greeting = "Hi!"
    def setName(self, name):
        self.name = name
    def greet(self):
        return Person.greeting+" I am "+self.name
```

CLASSES: VISIBILITY

In python there is no such thing as a private method or attribute. **Everything is public**. The naming *convention* for “private” methods and attributes is to precede their name with the _ symbol.

```
class Person:
    def setName(self, name):
        self._name = name
    def greet(self):
        return "Hi! I am "+self._name
```

CLASSES: CONSTRUCTOR

In python the construct function is named __init__ and it is called at object instantiation.

You can specify one or more arguments. The first argument is the object instance reference.

```
class Person:
    def __init__(self, name):
        self._name = name
    def greet(self):
        return "Hi! I am "+self._name
p = Person("Federico")
```

CLASSES: INHERITANCE

You can **extend** a base class with another specifying the base class between the parenthesis at class definition. In order to get the base class reference you need to use the **super()** function.

```
class Person:
    def __init__(self, name):
        self._name = name
    def greet(self):
        return "Hi! I am "+self._name
```

```
class Student(Person):
    def greet(self):
        return "Leave me alone, I have to study"
```

TRY IT YOURSELF!

Include the modified version of **print_hi** in the **class** *Person*, use the class attributes, and derive the new class *Student*!

```python
class Person:
    def __init__(self, name, number):
        self.name = name
        self.number = number

    def print_hi(self):
        if self.number > 0:
            squares = [i**2 for i in range(0, self.number)]
            for elem in squares:
                print(elem)
        elif self.number == 0:
            print("Just a zero?!")
        else:
            print("Hi, my name is " + self.name)
```

```python
class Student(Person):
    def print_hi(self):
        print("I'm a student!")

if __name__ == '__main__':
    p = Person("Luca", 0)
    p.print_hi()
    print(p.name)

    s = Student("Luca",  -2)
    s.print_hi()
    print(s.name)
```

MORE ON CLASSES: DATA MODEL

The are a lot of built-in functions provided by the base class of all the classes object. Each of which provide a specific behavior, a few are:

- __len__ (self)

  Returns the "length" of the object (called by len())

- __str__ (self)

  Returns the object as a string (called by str())

- __lt__ (self, other), __lt__ (self, other), __eq__ (self,other)

  Called when the object is used in a comparison

- __getitem__ (self, key), __setitem__ (self, key, value)

  Called in square brackets access

WORKING WITH EXISTING PROJECTS

Pycharm provides different functionalities for project and code navigation.

Navigate the project with specialized project views, use shortcuts, jump between files, classes, methods and usages.

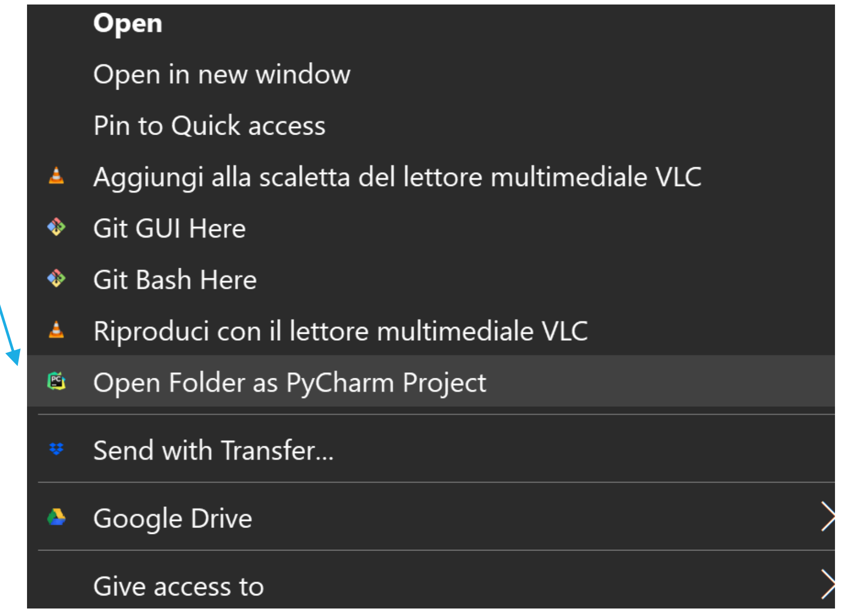
You can create a Python project by opening a folder as a project or even from scratch, by creating or importing the python files.

In this lecture we will not start from scratch…

OPEN A PROJECT

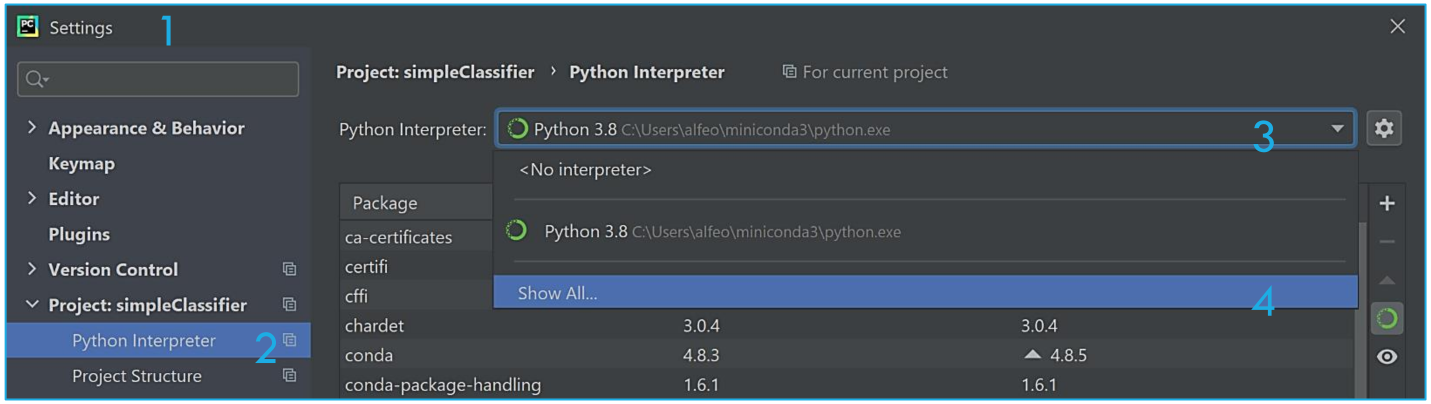
1. Unzip "simpleClassifier.rar"

2. Open the resulting folder "as a PyCharm project"


Open
Open in new window
Pin to Quick access
Aggiungi alla scaletta del lettore multimediale VLC
Git GUI Here
Git Bash Here
Riproduci con il lettore multimediale VLC
Open Folder as PyCharm Project
Send with Transfer...
Google Drive
Give access to

ALTERNATIVE INTERPRETER CONFIGURATION

File > Settings > Project > Python interpreter > Show all



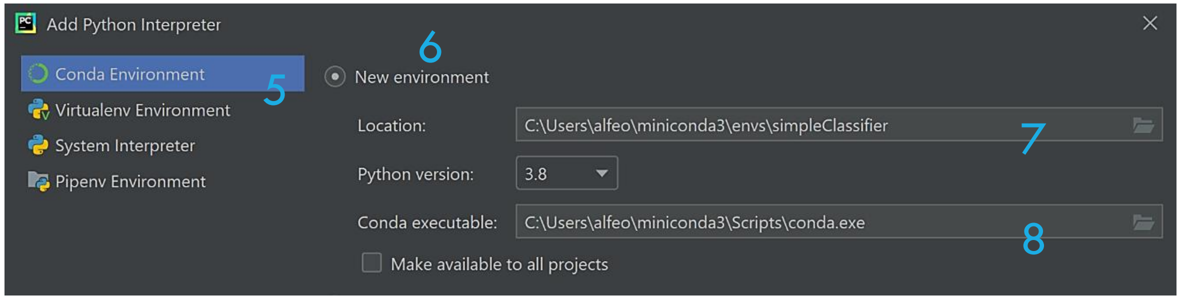
Python Interpreter: Python 3.8 C:\Users\alfeo\miniconda3\python.exe

Package		
ca-certificates		
certifi		
cffi		
chardet	3.0.4	3.0.4
conda	4.8.3	▲ 4.8.5
conda-package-handling	1.6.1	1.6.1

+ > Conda Environment > New environment

Location and conda executable may have slightly different paths (the first part) according to your miniconda3 location.

Apply > Ok



(●) New environment

Location: C:\Users\alfeo\miniconda3\envs\simpleClassifier

Python version: 3.8

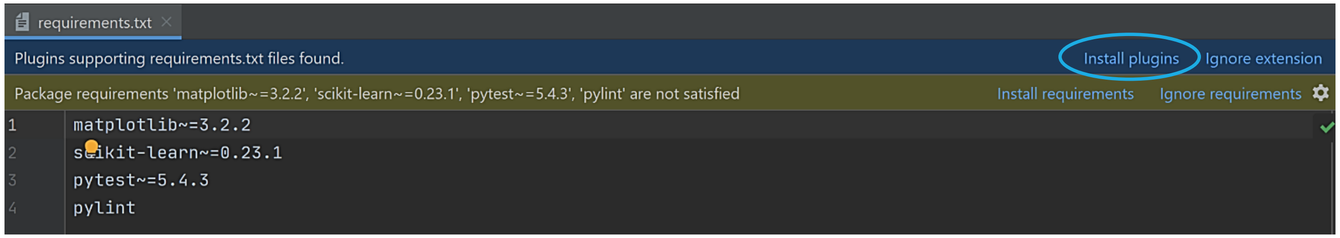
Conda executable: C:\Users\alfeo\miniconda3\Scripts\conda.exe

☐ Make available to all projects

PROJECT REQUIREMENTS 1/2

The requirements are all the packages that our software needs to run properly. Those can be installed with a PyCharm plugin.

Double click on “**requirements.txt**” in the Project View > Install Plugin



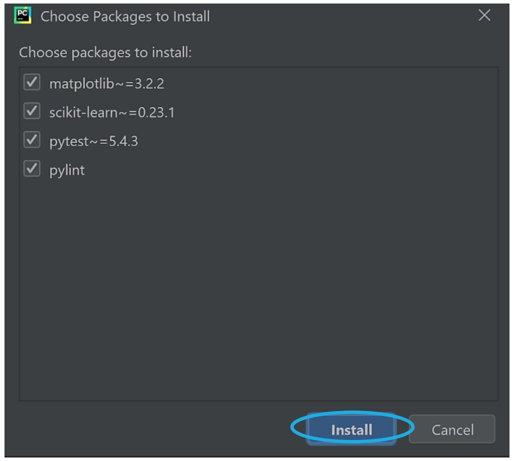
requirements.txt

Plugins supporting requirements.txt files found. Install plugins Ignore extension

Package requirements 'matplotlib~=3.2.2', 'scikit-learn~=0.23.1', 'pytest~=5.4.3', 'pylint' are not satisfied Install requirements Ignore requirements

```
matplotlib~=3.2.2
scikit-learn~=0.23.1
pytest~=5.4.3
pylint
```

Once the plugin is installed click on “Install requirements”, select all and click install.



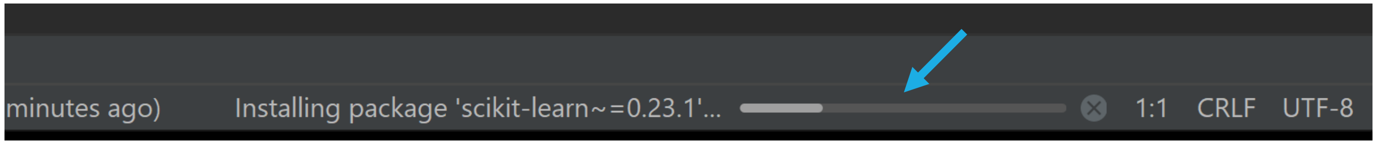
Choose Packages to Install

Choose packages to install:

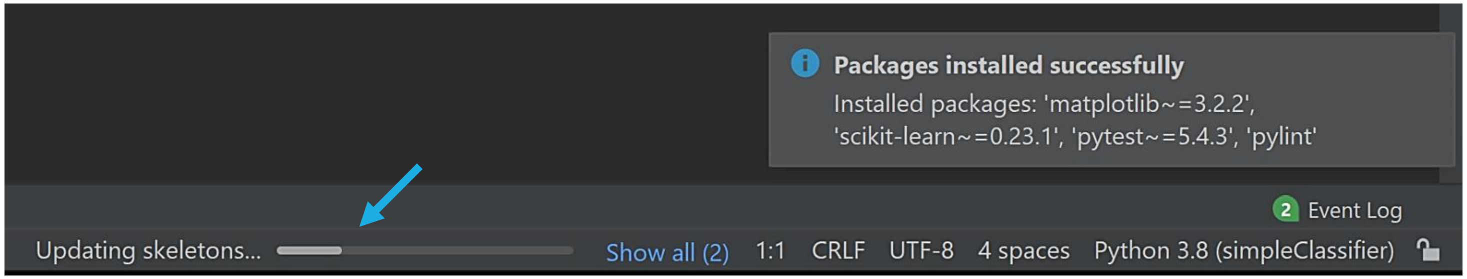
- ✓ matplotlib~=3.2.2
- ✓ scikit-learn~=0.23.1
- ✓ pytest~=5.4.3
- ✓ pylint

PROJECT REQUIREMENTS 2/2

- If everything goes smoothly you might see the package installation status progress at the bottom of the GUI.


minutes ago)   Installing package 'scikit-learn~=0.23.1'...   1:1 CRLF UTF-8

- Once the package installation is done a notification appears, but it is still **NOT** possible to go to next step, at least until PyCharm has finished "Updating skeletons".


Packages installed successfully
Installed packages: 'matplotlib~=3.2.2', 'scikit-learn~=0.23.1', 'pytest~=5.4.3', 'pylint'

Updating skeletons...   Show all (2) 1:1 CRLF UTF-8 4 spaces Python 3.8 (simpleClassifier)

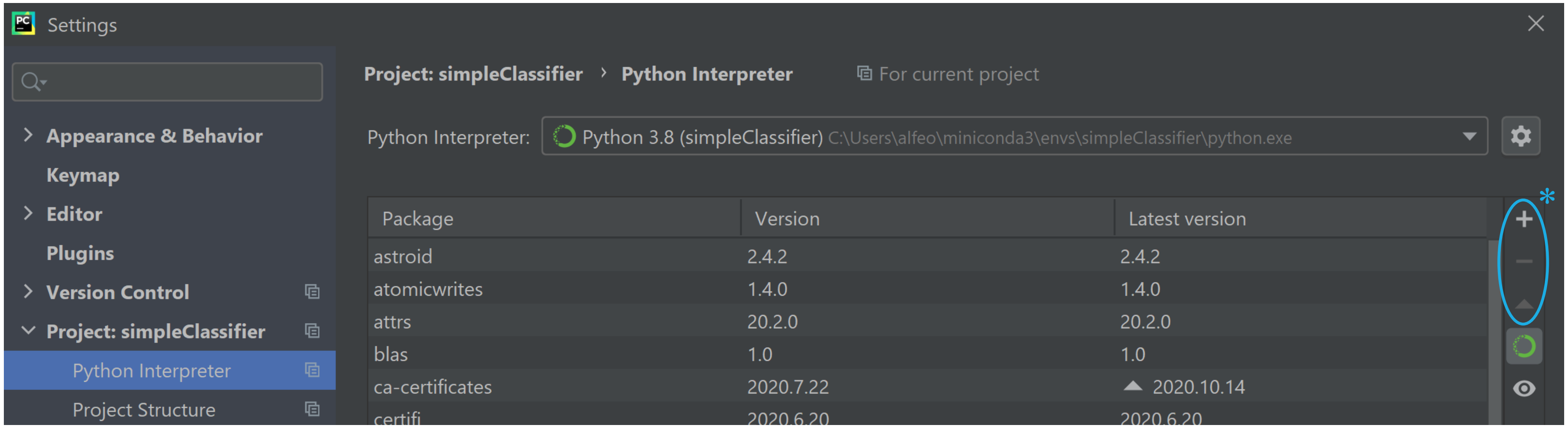
- Each package can also be installed via GUI or with the Terminal by using
  - "pip install <name_lib>" to install a single package
  - "pip install –r requirements.txt" to install the packages on the requirements.txt

CHECK THE PACKAGES AVAILABLE

It can take some minutes to complete the requirements installation. A restart may be required before moving to the next step!

Once done you can see (add, eliminate and upgrade*) the packages and libraries available in your virtual environment by checking :

File > Settings > Project > Python Interpreter



Settings

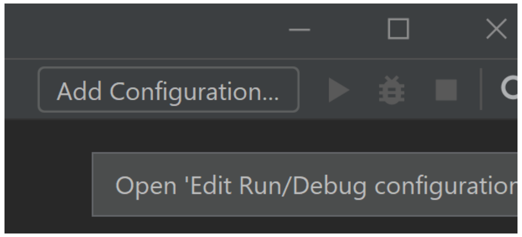
Project: simpleClassifier › Python Interpreter   For current project

Python Interpreter: Python 3.8 (simpleClassifier) C:\Users\alfeo\miniconda3\envs\simpleClassifier\python.exe

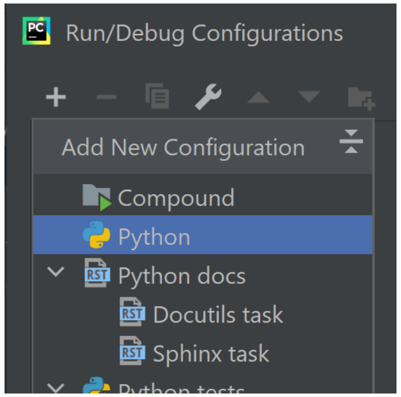
Package	Version	Latest version
astroid	2.4.2	2.4.2
atomicwrites	1.4.0	1.4.0
attrs	20.2.0	20.2.0
blas	1.0	1.0
ca-certificates	2020.7.22	▲ 2020.10.14
certifi	2020.6.20	2020.6.20

CONFIGURE THE FIRST RUN

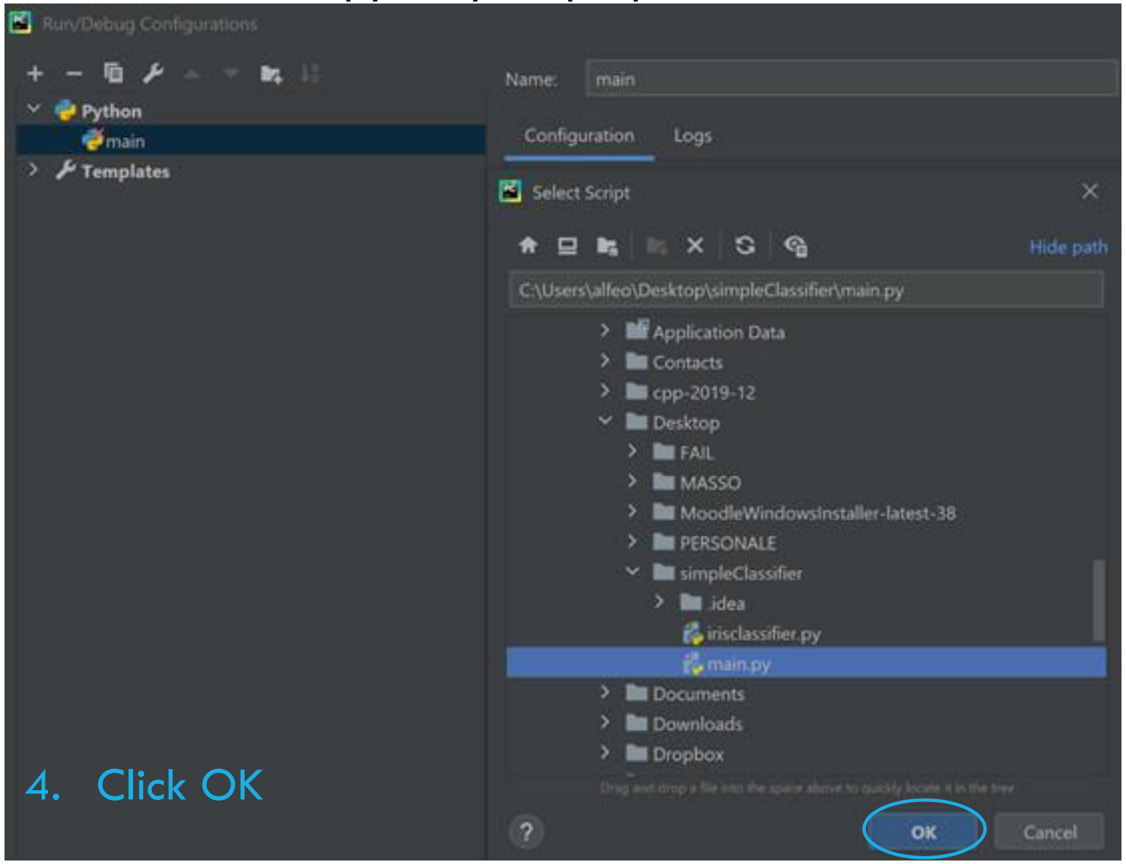
1. Click "Add configuration"



2. Select "Python"



3. Select "main.py" in your project folder.

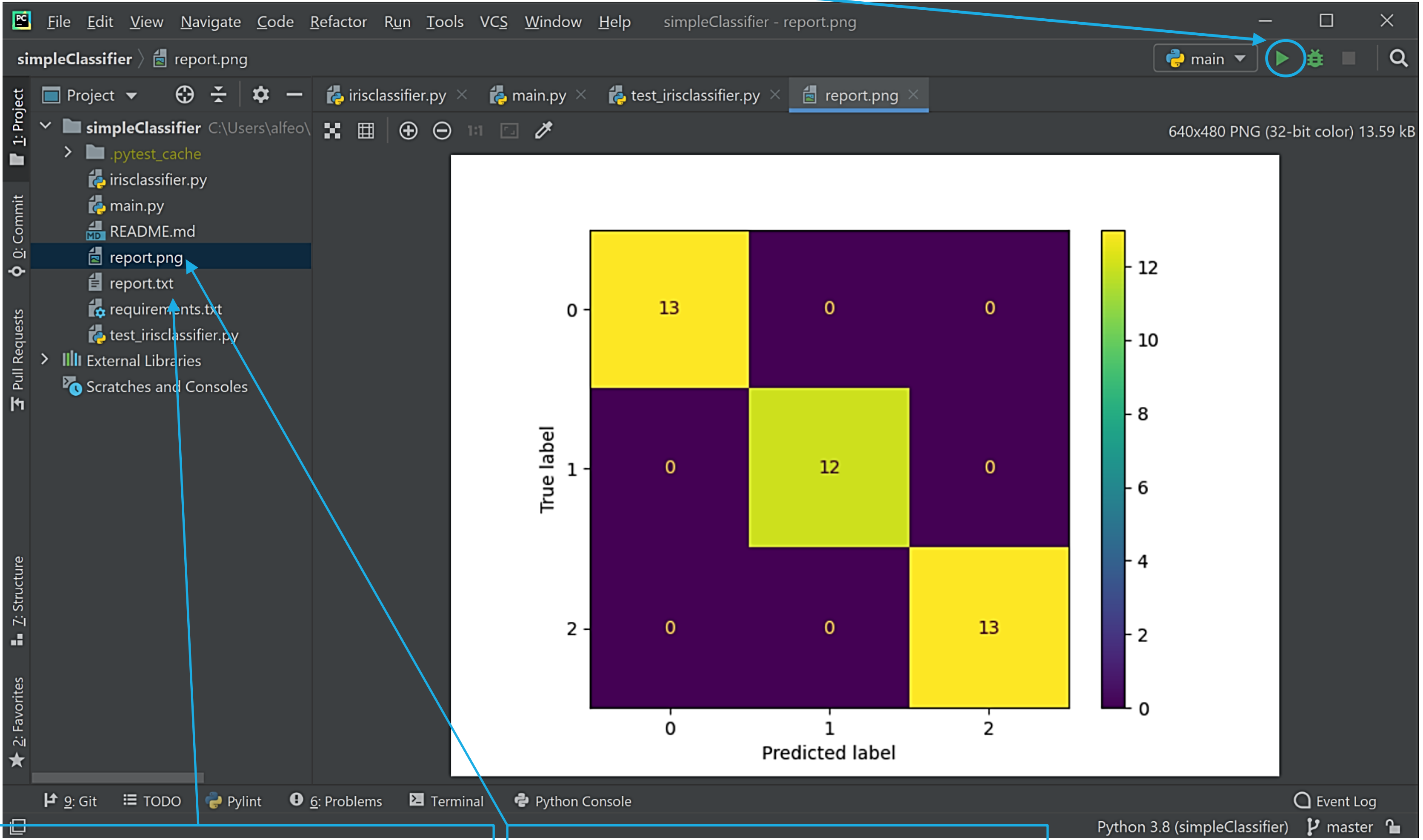


C:\Users\alfeo\Desktop\simpleClassifier\main.py

4. Click OK

RUN IT!

Click RUN



	0	1	2
0	13	0	0
1	0	12	0
2	0	0	13

Txt with evaluation score!

The confusion matrix!

QUESTIONS?