Python Crash Course

Session II - Introduction to ABM

Ali Seyhun Saral (Uni. Bologna)

13 June 2022

What is Python?

- `Python` is the one of the most popular general-purpose programming languages.
- It is a **multi-paradigm** programming language.
- Applications include -but not limited to- web development, software development, data analysis, automated tasks…
- It has a magnificent ecosystem of libraries and frameworks.

How to interact with python

- Python is essentially a **command-line interpreter** in its core.

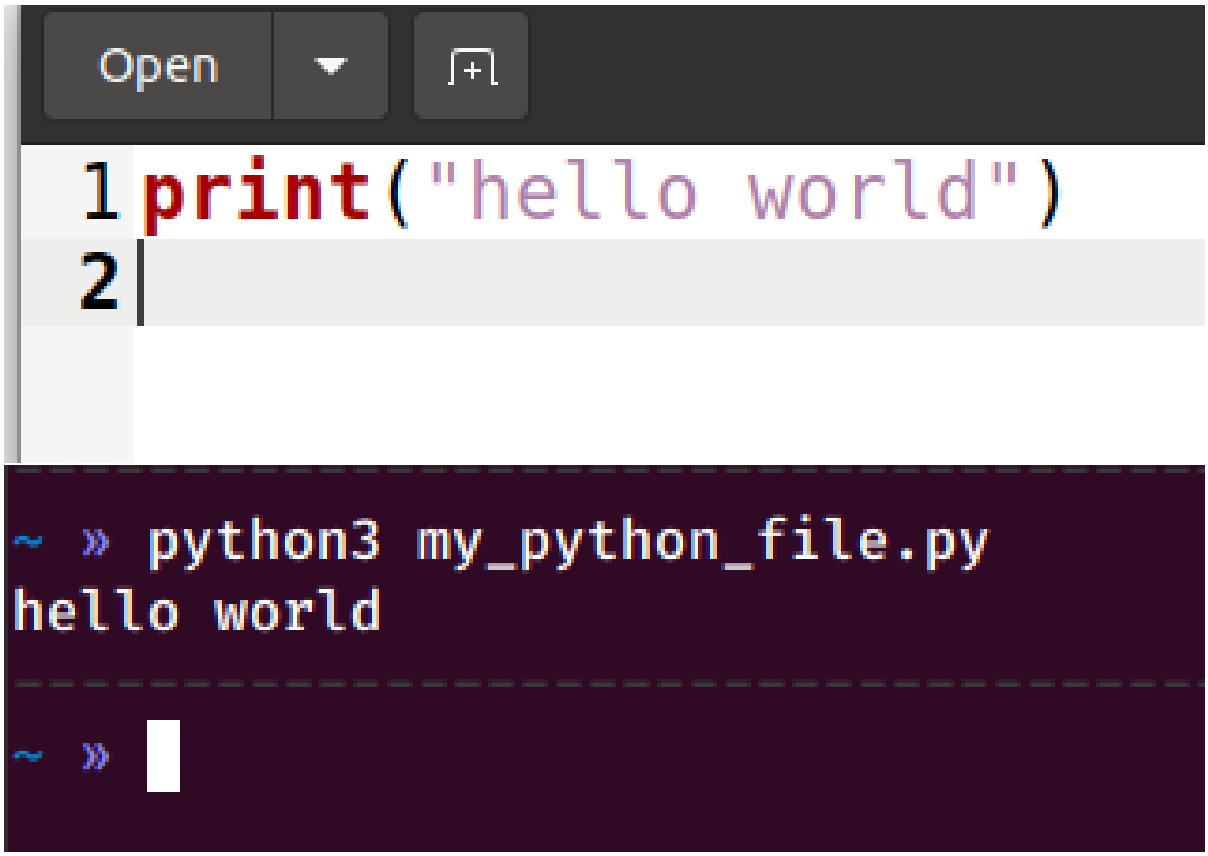
1- Python Interpreter

```
~ » python3                                                    aliseyhun@satur
Python 3.8.10 (default, Sep 28 2021, 16:10:42)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello world")
hello world
>>>
```

How to interact with python

2- Running a Python Script from the Terminal

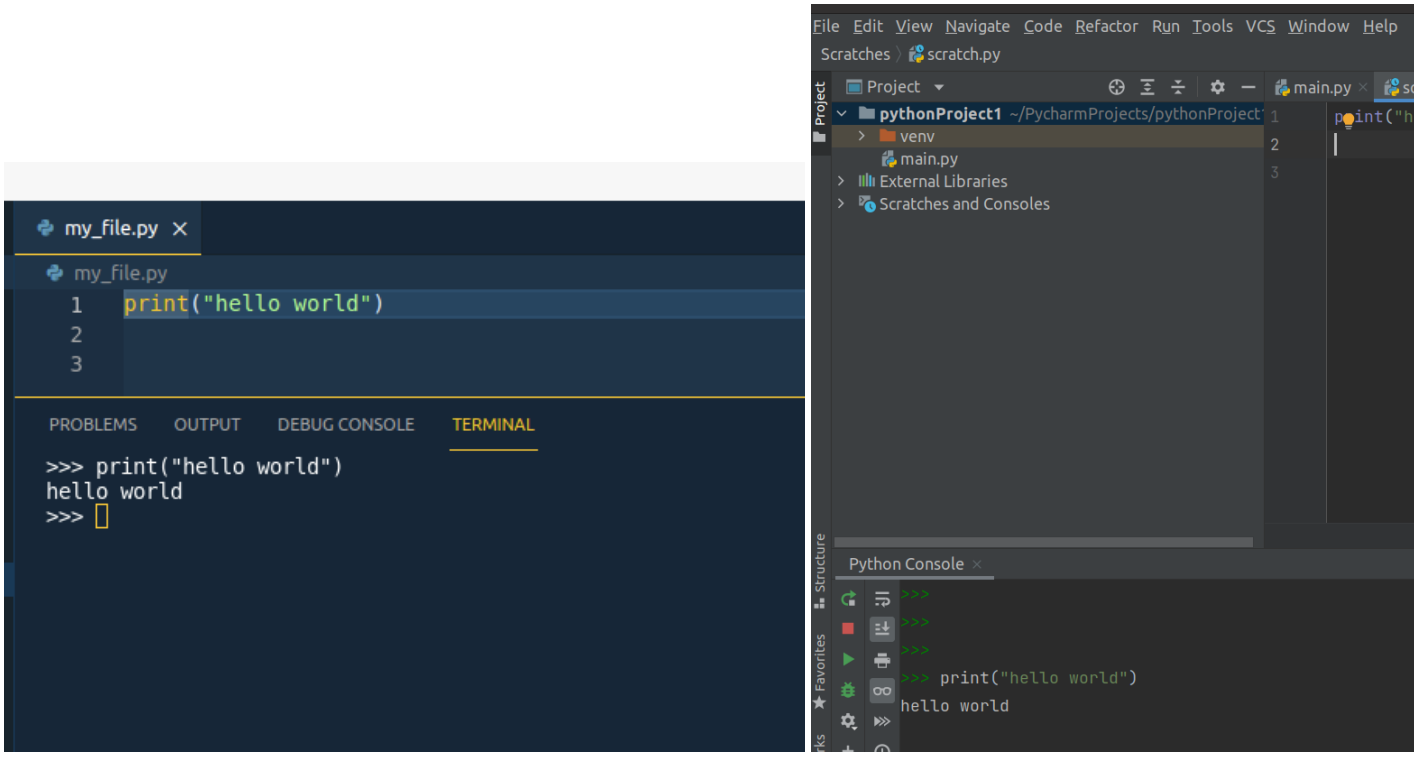
- Python scripts have the extension `.py`



How to interact with Python

3- Using a development environment

- Some of them are `PyCharm`, `VSCode`, `RStudio` and so on.

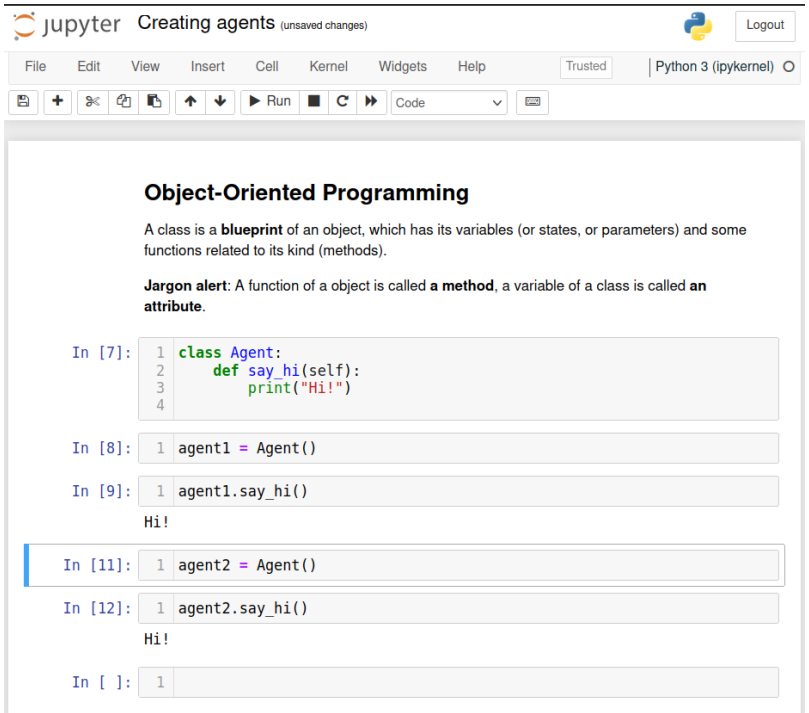


- Usually `Shift + Enter` or `Ctrl + Enter` to send the command to shell

How to interact with Python

4- Jupyter Notebook

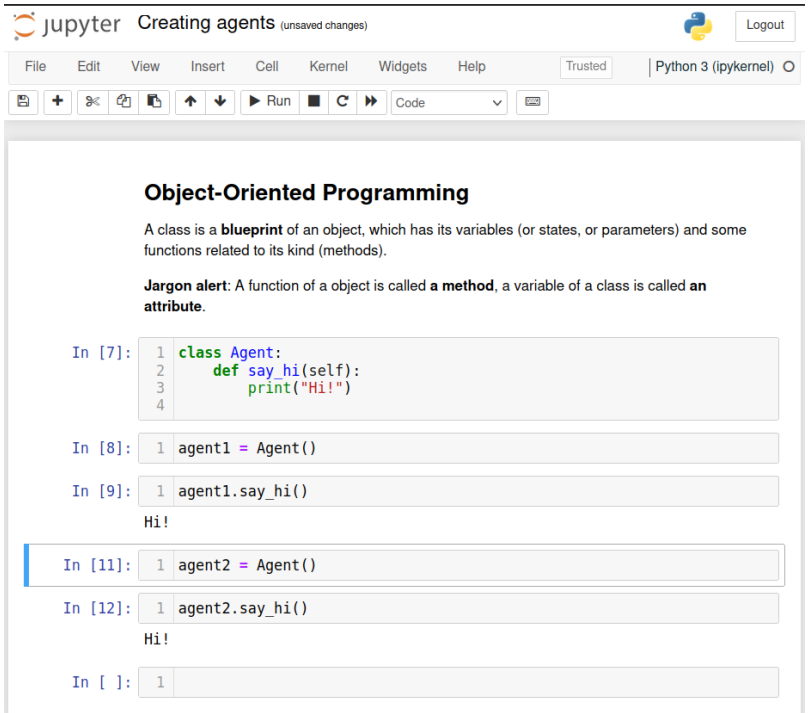
- Browser environment for writing and running interactive Python code.
- You can combine text and code cells to create a notebook.



How to interact with Python

4- Jupyter Notebook

Keyboard Shortcut	Description
`Shift + Enter`	Send the cell to the kernel for execution
`Ctrl + Enter`	Run the cell and advance to the next cell
`Enter`	Edit the cell
`Esc`	Stop Editing the cell
`H`	Help
`M`	Cell to Markdown (text)
`Y`	Cell to code



Hello World!

- To print things in the console, one should use `print()` function.

. . .

- Let's go ahead and print `Hello World`:

. . .

```
print("Hello World!")
```

```
Hello World!
```

Using Python as calculator

- Using `python` as calculator:

. . .

```
42 + 38
```

```
80
```

. . .

```python
36 * 5
```

```
180
```

Arithmethic Operators in Python

`a + b`	Addition
`a - b`	Subtraction
`a * b`	Mutiplication
`a / b`	Division
`a ** b`	Exponention (\(a^b \))
`a % b`	Modulo (\(a \: mod \: b \))

Variables in Python

- Links a value to a name
- Case-sensitive
- Can contain any alphanumeric characters and underscores (A-z, 0-9, _)
- Cannot start with a number
- You can print the value of a parameter with the `print()` function.
- Oftentimes just a variable name is enough to print it.

. . .

```python
height = 100 * 2
print(height)
```

```
200
```

. . .

```python
height
```

```
200
```

. . .

```
height = 297
width = 210

area_mm2 = height * width
print(area_mm2)
```

62370

Commenting in Python

If you have the character **#** on a line, Python interpreter will not read the rest of the line

. . .

```
# Area of an A4 paper
height = 297 # in mm
width = 210  # in mm

area_mm2 = height * width  # in mm2
print(area_mm2)
```

62370

Types in Python

```
# Area of an A4 paper
height = 297 # in mm
type(height)
```

`int`

. . .

```
# Area of an A4 paper
height = 297.0 # in mm
type(height)
```

```
float
```

. . .

```python
# Area of an A4 paper
measure_name = "area"
type(measure_name)
```

```
str
```

. . .

- You can check the type of a variable using the `type()` function.

Types in Python

We will deal with four main variable types:

- Integers (`int`): `1`, `2`, `-5` ,`2910`

- Float (`float`): `1.0`, `2.5`, `-5.3` (decimal numbers)
- String (`str`): `"hi"`, `'good morning!'`, `'Wow!  '` (text)
- Boolean (`bool`): `True`, `False` (logical)

Types in Python

- Each type has its own set of rules

```python
3 + 5
```

```
8
```

. . .

```python
'3' + '5'
```

```
'35'
```

. . .

```
"Hello" + " World!"
```

```
'Hello World!'
```

Practice

python_intro/Ex1.ipyb

python_intro/Ex2.ipyb

Lists in Python

- A list is a collection of items in a particular order.
- Lists can contain different types of items.

. . .

```
a = [10,20,30]
print(a)
```

```
[10, 20, 30]
```

. . .

```
b = ["ali", "bob", "chiara"]
print(b)
```

```
['ali', 'bob', 'chiara']
```

. . .

```
c = ["ali", 2]
print(c)
```

```
['ali', 2]
```

Reaching into List Items in Python

- You can access the items in a list using the index.
- The index starts from 0.

my_list =	["ali",	"bob",	"chiara"]
	0	1	2

. . .

. . .

```
my_list[0]
```

```
'ali'
```

. . .

```
my_list[1]
```

```
'bob'
```

. . .

```
my_list[2]
```

```
'chiara'
```

Reaching from the end

- You can access the items from the end using a negative index starting from -1.

my_list =	["ali",	"bob",	"chiara"]
	-3	-2	-1

. . .

. . .

```
my_list[-1]
```

```
'chiara'
```

. . .

```
my_list[-3]
```

```
'ali'
```

List Slicing in Python

- You can access a subset of a list using the slice operator `:`.
- Basic usage

[start : end]

- `end` is not included in the slice.

. . .

```
my_list = ["ali", "bob", "chiara", "dominique", "elizabeth"]
```

. . .

```
# Get first three items
my_list[0:3]
```

```
['ali', 'bob', 'chiara']
```

. . .

```
# Get three items starting from 1
my_list[1:4]
```

```
['bob', 'chiara', 'dominique']
```

List Slicing in Python

- If you leave the index empty, python will take from the beggining/end.

```
my_list = ["ali", "bob", "chiara", "dominique", "elizabeth"]
```

. . .

```
# From the beginning until the index 4
my_list[:4]
```

```
['ali', 'bob', 'chiara', 'dominique']
```

. . .

```
# From index 1 until the end
my_list[1:]
```

```
['bob', 'chiara', 'dominique', 'elizabeth']
```

Updating List Items in Python

- You can update the items in a list using the index.

```
my_list = ["ali", "bob", "chiara"]

my_list[0] = "alice"
print(my_list)
```

```
['alice', 'bob', 'chiara']
```

List Type and Methods

- Lists has the type `list`

. . .

```
my_list = ["ali", "bob", "chiara"]
type(my_list)
```

```
list
```

. . .

- There are specific operations that can be performed on lists.

. . .

- You can add items to a list using the `append()` function.

. . .

```
my_list.append("dominique")

print(my_list)
```

```
['ali', 'bob', 'chiara', 'dominique']
```

Removing Items from a List in Python

- You can remove items from a list using the `pop()` function.
- The `pop()` function removes the last item in the list.

. . .

```
my_list = ["ali", "bob", "chiara"]

my_list.pop()
print(my_list)
```

```
['ali', 'bob']
```

. . .

- The `pop(index)` function removes the item at the specified index.

. . .

```python
my_list = ["ali", "bob", "chiara"]

my_list.pop(1)
print(my_list)
```

```
['ali', 'chiara']
```

Check if an Item Exists in a List in Python

- You can check if an item exists in a list using the `in` operator.
- Example:

```python
my_list = ["ali", "bob", "chiara"]

"ali" in my_list
```

```
True
```

. . .

```python
my_list = ["ali", "bob", "chiara"]

"alessandro" in my_list
```

```
False
```

Combining two lists

- You can combine two lists with `+` operator.
- Example:

```python
my_list1 = ["ali", "bob", "chiara"]
my_list2 = ["zoe", "yoshua"]

new_list = my_list1 + my_list2
print(new_list)
```

```
['ali', 'bob', 'chiara', 'zoe', 'yoshua']
```

Practice

Dictionaries

- Also a collection of objects like `lists`
- Unlike lists, items have keys (in other words keywords)
- Can be created with:
  - `{key1: value1, key2: value2}`, or
  - `dict(key1=value1, key2=value2)`

```
my_dictionary = {'ali': 1987, 'bob': 1953, 'chiara':1980}
print(my_dictionary)
```

```
{'ali': 1987, 'bob': 1953, 'chiara': 1980}
```

. . .

```
my_dictionary2 = dict(ali=1987, bob=1953, chiara=1980)

print(my_dictionary2)
```

```
{'ali': 1987, 'bob': 1953, 'chiara': 1980}
```

Retrieve an item from dictionary

- You can retreive an item from a dictionary by its key.

```
birthyears = {"ali": 1987, "bob": 1953, "chiara":1980}
```

```
birthyears["bob"]
```

```
1953
```

Adding a new item to dictionary

```
birthyears = {"ali": 1987,
              "bob": 1953,
              "chiara":1980}

birthyears['dana'] = 1992
print(birthyears)
```

```
{'ali': 1987, 'bob': 1953, 'chiara': 1980, 'dana': 1992}
```

Practice

Logical Operators

- We have two logical values: `True` and `False`
- `and` and `or` and `not` are the logical operators
- `and` means that both conditions must be true
- `or` means that at least one condition must be true
- `not` reverses the logical value

```
a = 5
b = 10
```

```
print(a == 5 and b == 10)
```

```
True
```

. . .

```
print(a == 5 and b == 5)
```

```
False
```

. . .

```
print(a == 1 or b == 10)
```

```
True
```

. . .

```
print(not a == 1)
```

```
True
```

Logical Operators

```
a = 5
b = 10
```

. . .

```
print(a == 1 or b == 10)
```

```
True
```

. . .

```
print(not a == 1)
```

```
True
```

`if` statement

```
my_variable = 42
```

```
my_variable = 42
if my_variable < 50:
    print("the variable is smaller than 50")
```

```
the variable is smaller than 50
```

Very important: Indentation

```
my_variable = 42

if my_variable < 50:
    print("the variable is smaller than 50")
```

```
the variable is smaller than 50
```

- Python doesn't have braces like `{ }` or `end` statements to indicate the span.

. . .

- Instead the hierarchy/ownership of the statements are determined by indents.

. . .

- `Four spaces` is the accepted convention but you can use `tab` or the another number of spaces as long as it is consistent.

. . .

- Graphical user interfaces often add four spaces instead of a tab.

`else` statement

```
my_variable = 42

if my_variable < 50:
    print("the variable is smaller than 50")
else:
    print("the variable is bigger than 50")
```

```
the variable is smaller than 50
```

. . .

`elif` statement

```
my_variable = 42

if my_variable < 40:
    print("the variable is smaller than 40")
elif my_variable < 50:
    print("the variable is between 40 and 50")
else:
    print("the variable is bigger than 50")
```

```
the variable is between 40 and 50
```

Functions in Python

- Python has many built-in functions

```
ages = [39,48,21,59]

max(ages)
```

```
59
```

```
my_name = "ali"
len(my_name)
```

```
3
```

We can build our own functions

- Functions are defined with `def` keyword.

```
def min_max_difference(x):
    return(max(x) - min(x))
    # Beware of indentation
```

```
ages = [39,48,21,59]
min_max_difference(ages)
```

```
38
```

Methods

- Some functions are associated to objects. They are called methods.
- The syntax for methods is `my_object.do_something()`

```
ages = [39,48,21,59]

ages.index(21) # find the index of a given item
```

```
2
```

Loop over a range of numbers

- Python creates a range object that is iterable with `range()` function.
- Then one can loop over it to make calculations.

```
for i in range(3,6):
  print(i**2)
```

```
9
16
25
```

For loop over list items

- Lists are not just good for collecting items
- Also for looping over them

. . .

```
my_list = [1,2,3,4]
```

```
for x in my_list:
  print(x ** 2)
```

```
1
4
9
16
```

Creating a new list using for loop

- We can generate an empty list and add items recursively.

. . .

- `list_name.append()` to add an item to a list

. . .

Example:

```
my_list = [1,2,3,4]

squares = []

for x in my_list:
  squares.append(x ** 2)
```

. . .

```
print(squares)
```

```
[1, 4, 9, 16]
```

List Comprehension

- List comprehension is a shorter syntax when you create a new list based on another list.

```
my_list = [1,2,3,4]
```

```
squares = []

for x in my_list:
  squares.append(x ** 2)
```

. . .

we can write instead:

```
squares = [x ** 2 for x in my_list]
```

. . .

List Comprehension

```
my_list = [1,2,3,4]
squares = [x ** 2 for x in my_list]
print(squares)
```

```
[1, 4, 9, 16]
```

. . .

* We can also add conditions

```
my_list = [1,2,3,4]
squares = [x ** 2 for x in my_list if x > 2]
print(squares)
```

```
[9, 16]
```

Modules

- One great advantage of python that it has a vast ecosystem of packages.
- Some packages are build in, but still needs to be imported.
- Python use the syntax `import packagename` to import a package.

- The functions, methods etc. comes as a subset of the package, which can be reached by a dot.

. . .

```
import random

random.choice(['ali', "bob", "chiara"])
```

```
'chiara'
```

Modules

- You can also import all objects directly. Then you wont need to call the package name before. . . .

```
from random import *

choice(['ali', "bob", "chiara"])
```

```
'bob'
```

. . .

- Or a subset:

```
from random import choice

choice(['ali', "bob", "chiara"])
```

```
'chiara'
```

Modules

- Or you can use an alias for the module . . .

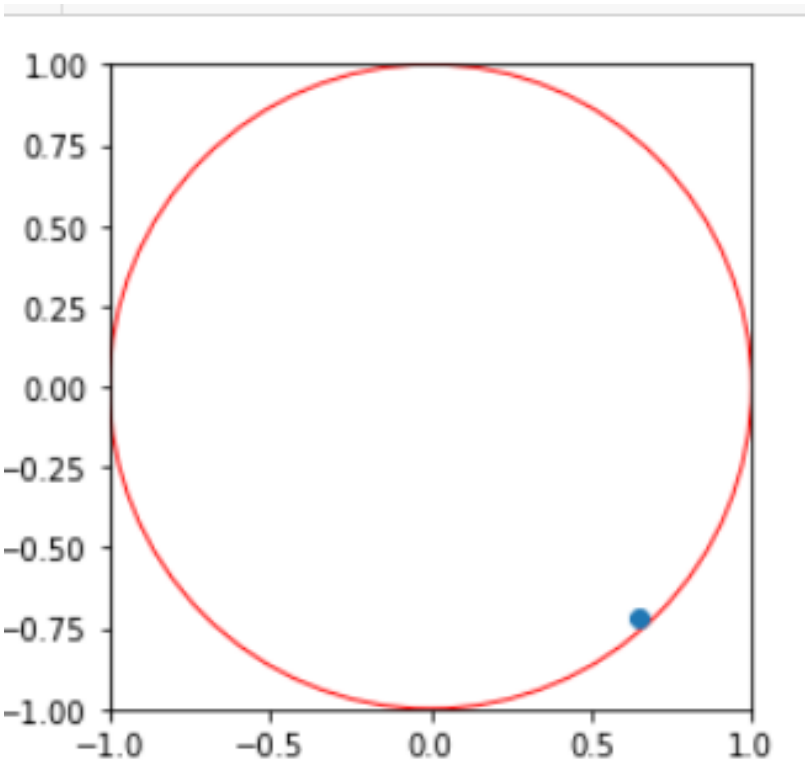
```
import random as rnd

rnd.choice(['ali', "bob", "chiara"])
```

```
'bob'
```

Exercise: Estimating the pi with with Monte Carlo simulation

Open `monte_carlo.ipynb`

- We have a square with with height and width of 1.
- We have a circle with radius of 0.1
- We want to estimate the value of Pi.
- We are gonna shoot random arrows inside the square. We will only know if they hit the circle or not.


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.0
−0.5
0.0
0.5
1.0