Python (data structures)

Loriano Storchi

loriano@storchi.org

http::/www.storchi.org/

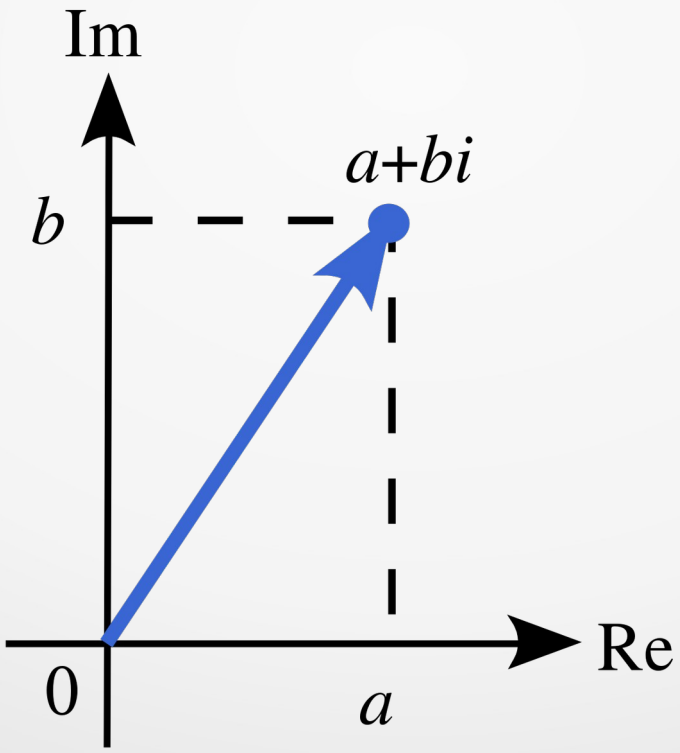
Data structures

- Data structures allow to organize data in order to make their use and their handling more efficient
- We will see in particular strings, lists, tuples, dictionaries and the sets.
- We will only see the minimum bases useful to carry out basic exercises

COMPLEX NUMBERS

Complex numbers

- A complex number is a number that can be expressed in the form **a + ib**, where **a** and **b** are real numbers, and **i** is a solution of the equation $x^2 = -1$. Because no real number satisfies this equation, **i is called an imaginary number**.



Complex numbers

- Complex numbers are intrinsically defined in python

```
a = 2 + 3j
b = 4 - 1j

print(a)
print(a.real, " ", a.imag)
print(b)
print(b.real, " ", b.imag)
```

```
(2+3j)
2.0   3.0
(4-1j)
4.0   -1.0
```

Complex numbers

- Complex numbers and the type function type() function is mostly used for debugging purposes. Two different types of arguments can be passed to type() function, single and three argument. If single argument type(obj) is passed, it returns the type of given object.

```
c = a * b
print(type(c), " valore ", c)

if type(c) == complex:
    print("c e\' un numero complesso")
```

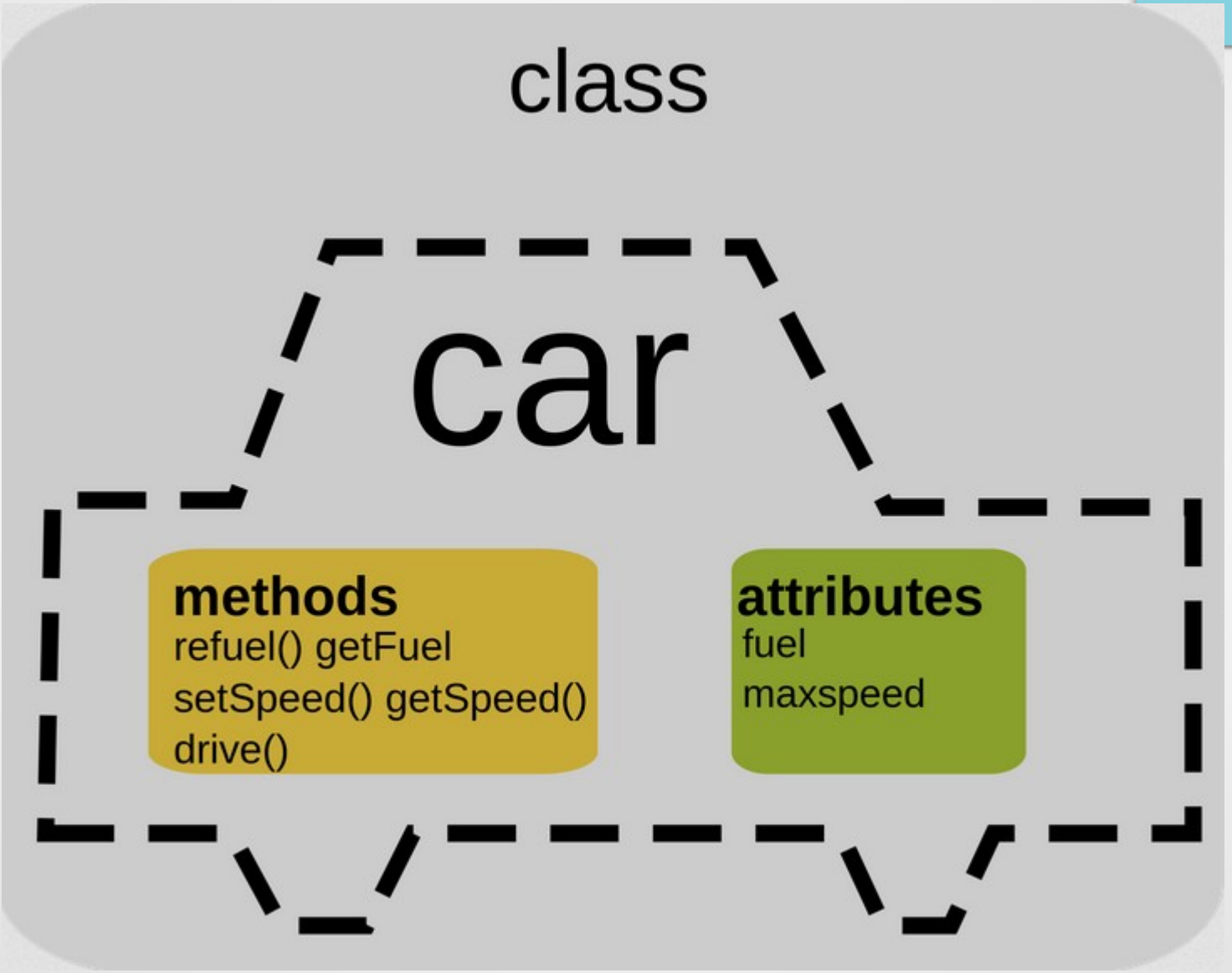
```
<class 'complex'>  valore  (11+10j)
c e' un numero complesso
```

Digression on OOP and classes

OOP

- Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data, in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods). (Wikipedia)
- The most used OOP languages are based on class.
- Python is multi-paradigm and supports OOP
  - A class specifies a new type within a class there are data items and methods

OOP


class
car
methods
refuel() getFuel
setSpeed() getSpeed()
drive()
attributes
fuel
maxspeed

OOP


Class
Definition of objects that share structure, properties and behaviours.
Building
class
Dog
class
Computer
class
Instance
Concrete object, created from a certain class.
Empire State
instance of Building
Lassie
instance of Dog
Your computer
instance of Computer

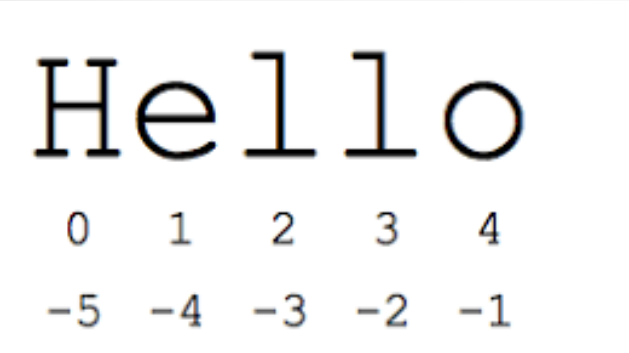
STRINGS

Strings

- A string is a sequence of characters, in python there are several methods / operations useful for the manipulation of strings, **a string in general can be seen as an array of characters**
- Operations between strings

Strings

- Extract sub-strings, strings are arrays of characters

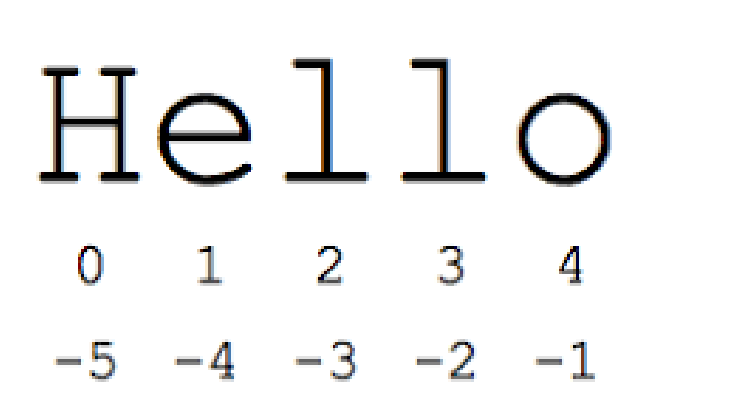



```
str1 = "Hello"
print(str1[0])
print(str1[0:3])
print(str1[-2:])
```

```
H
Hel
lo
```

Strings : delimiter

- : is the delimiter of the slice syntax to 'slice out' sub-parts in sequences , [start:end]
- [1:5] is equivalent to "from 1 to 5" (5 not included)
- [1:] is equivalent to "1 to end"
- [:-2] from “begin to -2” -2 not included

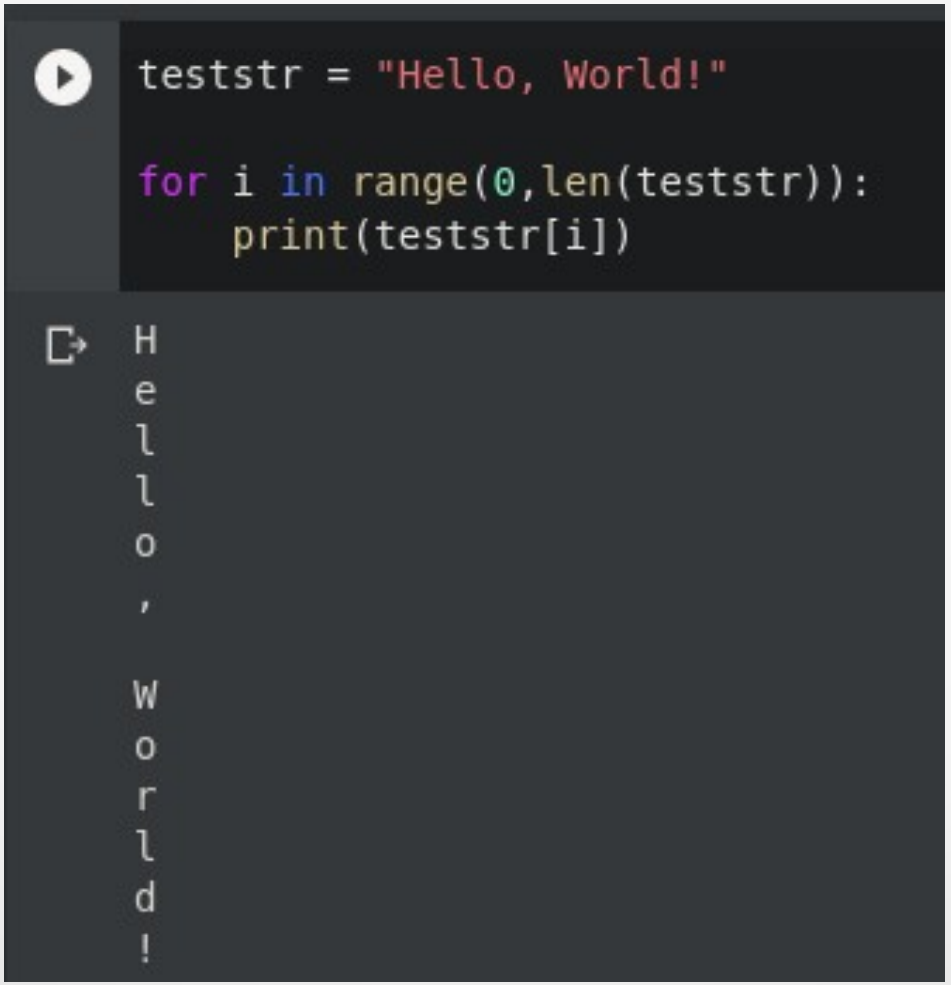



```
str1 = "Hello"
print(str1[0])
print(str1[0:3])
print(str1[-2:])
print(str1[-1])
print(str1[:-2])
```

```
H
Hel
lo
o
Hel
```

Strings

- Extract sub-strings, strings are arrays of characters



Strings

- String’s class in python has several methods
- **find()** It determines if string str occurs in string, or in a substring of string if starting index beg and ending index end are given.

```
str = "Hello, World!"
index = str.find("ll")
if index >= 0:
  print("a ", index, " trovato ", str[index:])
```

```
a  2  trovato  llo, World!
```

Strings

- **split()** method returns a list of strings after breaking the given string by the specified separator.

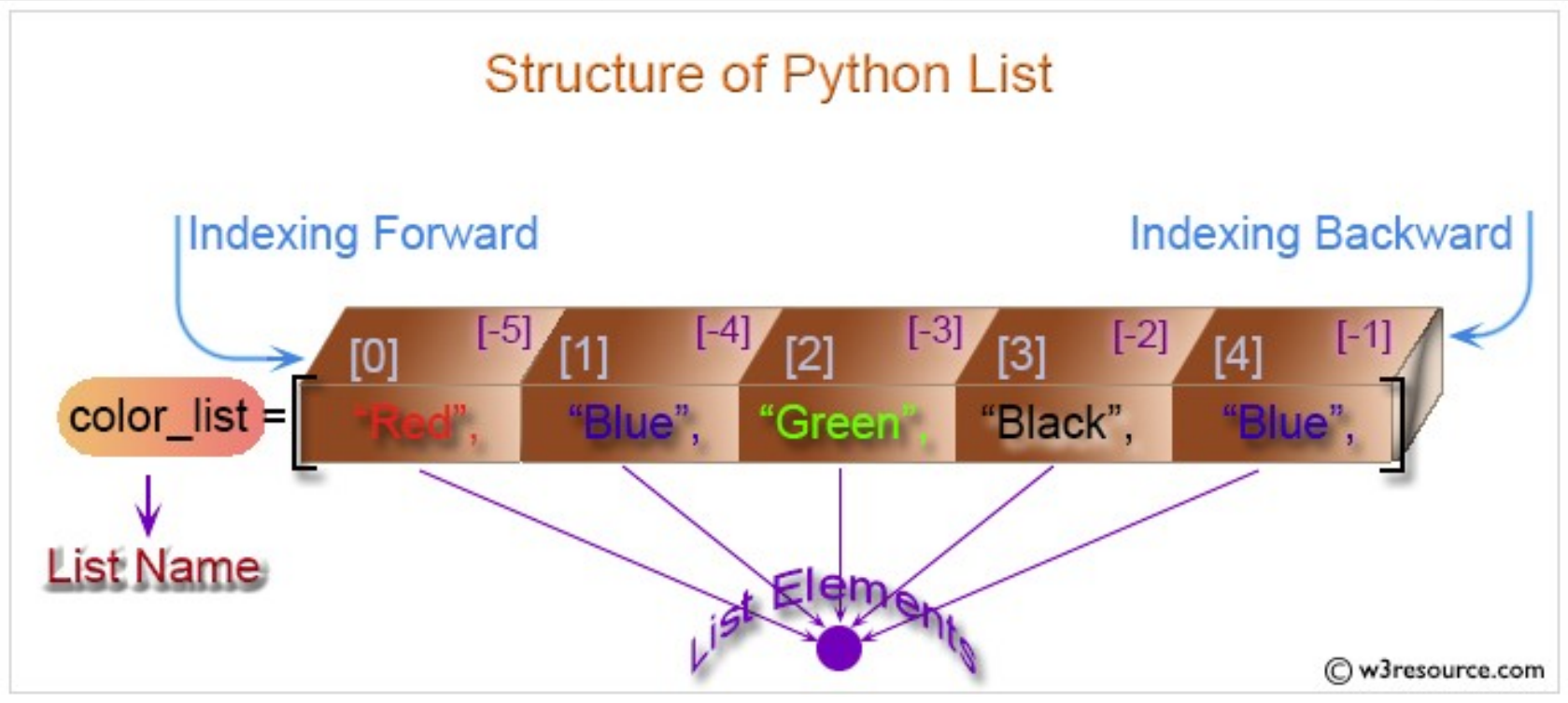
```
str = "Hello, World!"
res = str.split(" ")
idx = 0
for r in res:
    idx += 1
    print(idx , " - ", r)
```

```
1  -  Hello,
2  -  World!
```

LISTS

Lists

- **List** is a collection which is ordered and changeable. Allows duplicate members.



Lists

- Lists are ordered sequences of objects, many basic operations are in common with strings

```
a = [1, "pippo", 4.5, "pluto"]
print(a[0])
for i in range(0,len(a)):
    print(a[i])

for l in a:
    print(l)
```

```
1
1
pippo
4.5
pluto
1
pippo
4.5
pluto
```

Lists

- Lists are ordered sequences of objects, many basic operations are in common with strings

```
a = [1, 3.5, -6.0, 5]
print(a)
a[1] = "pluto"
print(a)
```

```
[1, 3.5, -6.0, 5]
[1, 'pluto', -6.0, 5]
```

Lists

- Lists are ordered sequences of objects, many basic operations are in common with strings, but remember:

A string does not support

item assignment

```
>>> str = "Hello"
>>> str[0] = "v"
Traceback (most recent
  File "<stdin>", line
TypeError: 'str' objec
>>> str = "pluto"
```

Lists

- Lists have many methods we will see only the most important
  - **list.append(elem)** -- adds a single element to the end of the list.
  - **list.insert(index, elem)** -- inserts the element at the given index, shifting elements to the right.
  - **list.extend(list2)** adds the elements in list2 to the end of the list.

Lists

- Lists adding elements

```
a = [1, 3.5, -6.0, 5]
a.append(46)
a.append("pluto")
print(a)
```

```
[1, 3.5, -6.0, 5, 46, 'pluto']
```

Lists

- Lists adding elements

```
a = [1, 3.5, -6.0, 5]
a.insert(1, "pippo")
print(a)
```

```
[1, 'pippo', 3.5, -6.0, 5]
```

Lists

- Lists adding elements

```
a = [1, 3.5, -6.0, 5]
b = ["second", "list"]
a.extend(b)
print(a)
```

```
[1, 3.5, -6.0, 5, 'second', 'list']
```

Lists

- Remove elements from a list
  - **remove** removes the first matching value, not a specific index

```
a = [1, 2, 3, 4, 2, 5]
a.remove(2)
print(a)
a.remove(2)
print(a)
```

```
[1, 3, 4, 2, 5]
[1, 3, 4, 5]
```

Lists

- Remove elements from a list
  - **del** removes the item at a specific index

```
a = [1, 2, 3, 4, 2, 5]
del a[2]
print(a)
del a[2]
print(a)
```

```
[1, 2, 4, 2, 5]
[1, 2, 2, 5]
```

Lists

- Remove elements from a list
  - **pop** removes the item at a specific index and returns it

```
a = [1, 2, 3, 4, 2, 5]
val = a.pop(2)
print(a, val)
```

```
[1, 2, 4, 2, 5] 3
```

Lists

- List has a **sort** method

```
a = [1, 4, 3, 5, 9, 34, 7]
a.sort()
print(a)
c = ["a", "pluto", "paperino", "b"]
c.sort()
print(c)
```

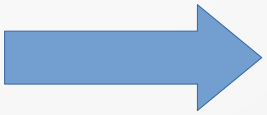
```
[1, 3, 4, 5, 7, 9, 34]
['a', 'b', 'paperino', 'pluto']
```

TUPLES

Tuples

- Python tuples are very similar to lists but their manipulation is faster since they are "**immutable**"

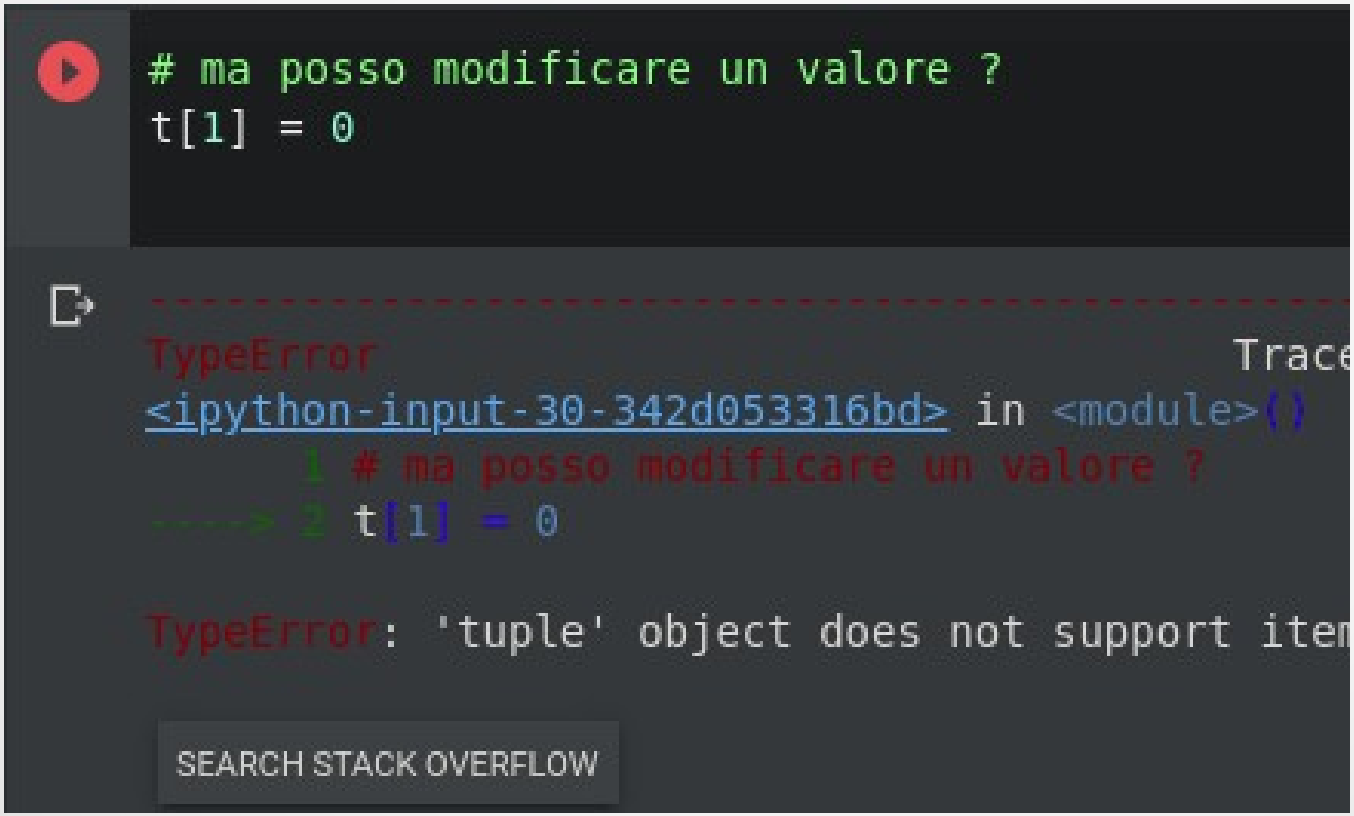
```
t = (1,3.5,8,10.0)
for i in range(len(t)):
    print(t[i])

for val in t:
    print(val)
```



```
<terminated
1
3.5
8
10.0
1
3.5
8
10.0
```

Tuples

- But I can not modify a value



But...

- But a tuples element can be a mutable one so...

```
t = (1, 4, 5, 6 , "se", 4, [7, 9, 0])
print(t)
t[6].append("last one")
print(t)
t[6][0] = "primo"
print(t)
```

```
(1, 4, 5, 6, 'se', 4, [7, 9, 0])
(1, 4, 5, 6, 'se', 4, [7, 9, 0, 'last one'])
(1, 4, 5, 6, 'se', 4, ['primo', 9, 0, 'last one'])
```

DICTIONARY

Dictionary

- A dictionary is a **sequence of elements**, **each element is** a **pair key : value**. The keys are unique and dictionaries are **created using {**

```
d = {"k1" : 1, "k2" : 2, 4 : "val3"}
print(d[4], d["k1"])
d["k1"] = 1.5
d["quattro"] = 4
print(d)
```

```
val3 1
{'k1': 1.5, 'k2': 2, 4: 'val3', 'quattro': 4}
```

Dictionary

- **We can check if a key is present or not easily**

```
diz = {'k1': 1, 'k2': 2, 4: 'val3', 'quattro': 4}
if "quattro" in diz:
  print("yes and value is ", diz["quattro"])
if not 5 in  diz:
  print("key it is not present")
```

```
yes and value is  4
key it is not present
```

Dictionary

- **We can check if a value is present or not easily**

```
diz = {'k1': 1, 'k2': 2, 4: 'val3', 'quattro': 4}

if "val3" in diz.values():
  print("value value3 is present")
```

```
value value3 is present
```

Dictionary

- dict.items() returns of the dictionary a list of tuples

```
diz = {'k1': 1, 'k2': 2, 4: 'val3', 'quattro': 4}
for x in diz.items():
  print(x)
```

```
('k1', 1)
('k2', 2)
(4, 'val3')
('quattro', 4)
```

Dizionari

```
diz = {'k1': 1, 'k2': 2, 4: 'val3', 'quattro': 4}
for x in diz.items():
  print(x)
for k in diz.keys():
  print(k)
for  v in diz.values():
  print(v)
```

```
('k1', 1)
('k2', 2)
(4, 'val3')
('quattro', 4)
k1
k2
4
quattro
1
2
val3
4
```

Dizionari

- **clear**(): removes all items
- **pop**(): removes and returns element having given key
- **del** statement removes the given item from the dictionary. If given key is not present in dictionary then it will throw KeyError.

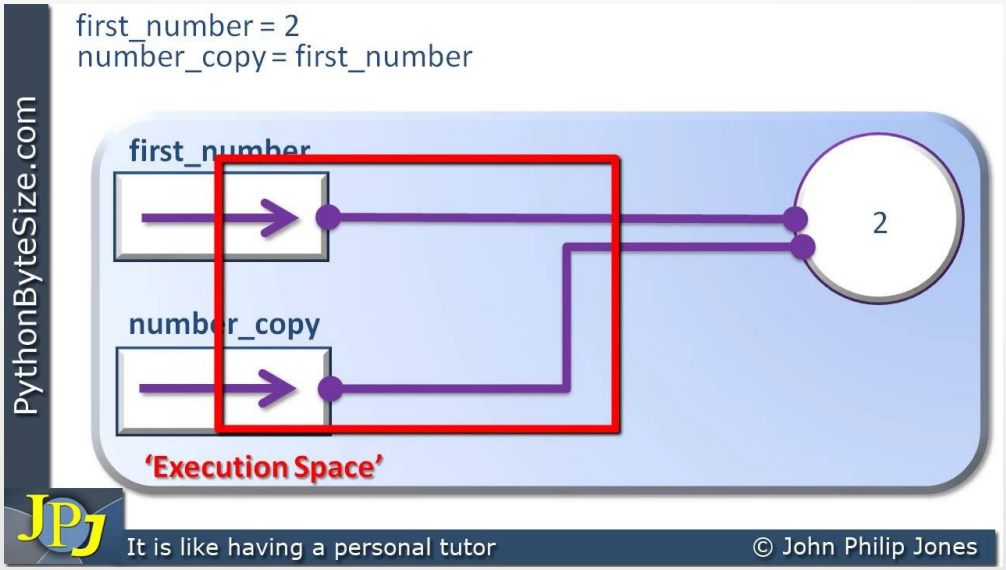
```
diz = {'k1': 1, 'k2': 2, 4: 'val3', 'quattro': 4}
del diz["k1"]
print(diz)
v = diz.pop(4)
print(diz, v)
diz.clear()
print(diz)
```

```
{'k2': 2, 4: 'val3', 'quattro': 4}
{'k2': 2, 'quattro': 4} val3
{}
```

PYTHON SOME DETAILS ABOUT MEMORY MANAGEMENT

Reference

- When I declare a variable I am asking for a certain amount of memory
- In python, the assignment operation manipulates the references, so **x = y** does not create a copy of the data contained in **y**, but simply creates a reference to **y**, that is **x** will point to **y**

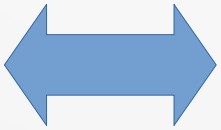


Reference

- **When I write x = 4 we allocate the memory space necessary to contain the integer 4 and then we "stored the address of the" (created the reference to) the memory location in x**

```
x = 3
y = x
x = 4
print(y)
```

```
3
```



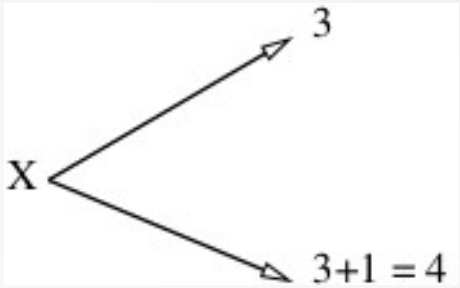
```
a = [1,2,3,4]
b = a
a.append(5)
print(b)
```

```
[1, 2, 3, 4, 5]
```

Reference

```
x = 3
x = x + 1
print(x)
```

```
4
```



- What really happens when increment x?
  - The interpreter retrieves the value contained in the memory address to which x refers
  - The result of the 3 + 1 operation is calculated and the result is stored in a new memory location
  - You change the reference in x, x will now refer to the new address in memory where the value 4 is stored
  - Python has a garbage collector that eliminates free all the memory allocated when there are no more names referring to the memory areas in question

EXECISE MATRIX MULTIPLICATION

Matrix

- I can use a list of lists to store a matrix in a simple way

```
import random
import math

A = [[0.0, 0.0, 0.0], \
     [0.0, 0.0, 0.0], \
     [0.0, 0.0, 0.0]]

for i in range(len(A)):
  for j in range(len(A[0])):
    A[i][j] = random.uniform(0.0, 1.0)

print("Matrix A")
print(A)
```

```
Matrix A
[[0.109058751134825, 0.2460718519519258
```

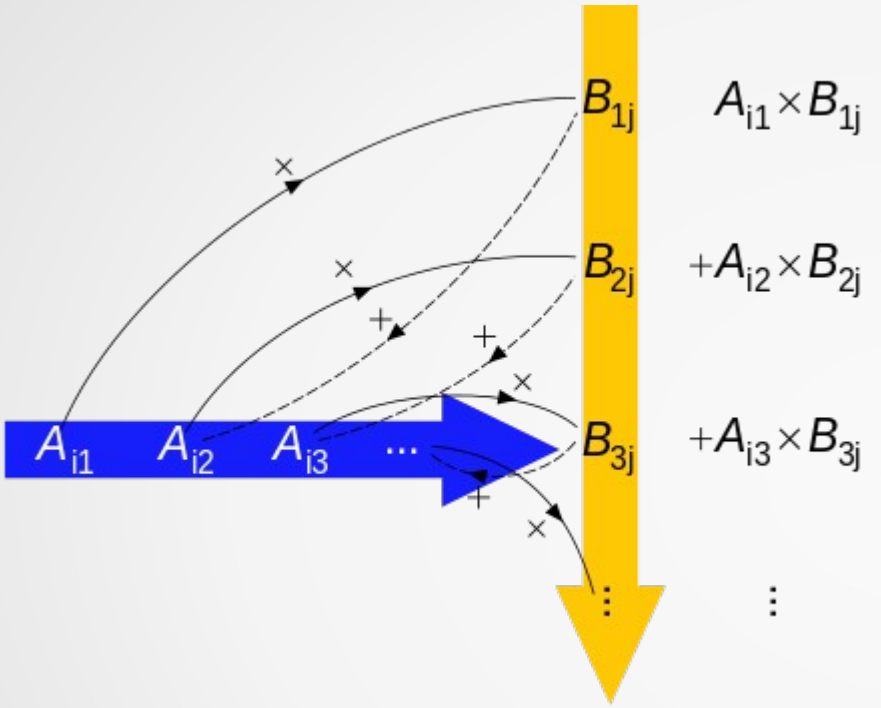
Matrix multiplication

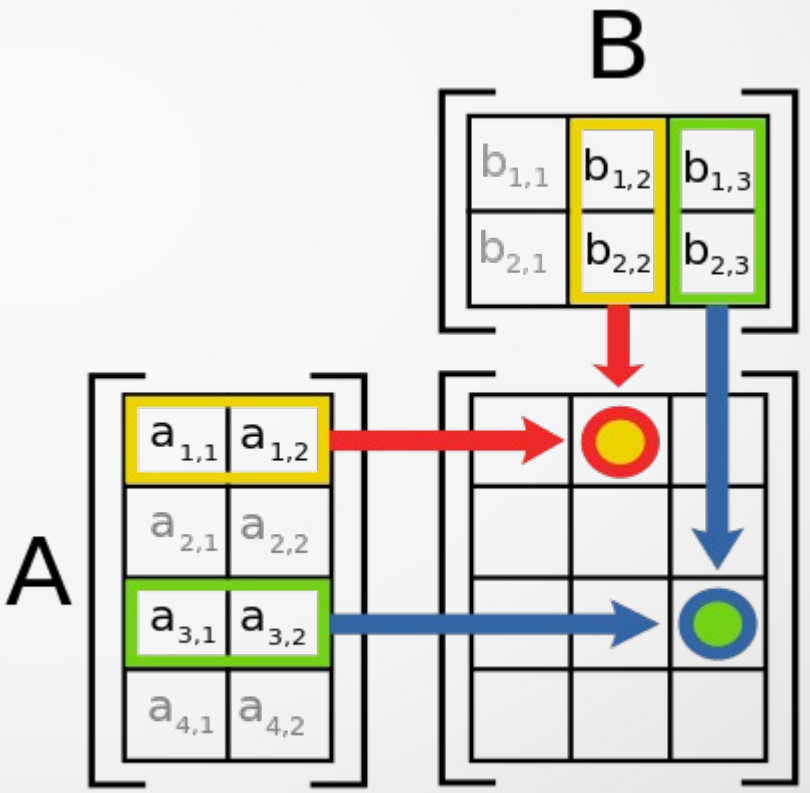
$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1p} \\ B_{21} & B_{22} & \cdots & B_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mp} \end{pmatrix}$$

$$(\mathbf{AB})_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj} \ .$$

The product is defined only for matrices with compatible dimensions

Moltiplicazione matrice matrice





Exercise

- Write a program that given two 3x3 arrays filled with random numbers computes and prints the result of the matrix multiplication

```
[redo@banquo mtxmtx (master)]$ python mtx.py
Matrix A
[0.1924696838424818 6, 0.969694776511462 9, 0.823732591468549 9]
[0.177986003994455 2, 0.975535311913036 9, 0.821708541333982 5]
[0.412794860282205 36, 0.05779395844699 2644, 0.40208982438481 4]
Matrix B
[0.719654428041583 5, 0.823525770039240 3, 0.934926373722345 8]
[0.273935242617288 85, 0.709371981871736 3, 0.44075517202906 7]
[0.126088196695921 42, 0.93912187585172 8, 0.46397180681599 48]
Matrix C
[0.508008191127318 5, 1.619963346520345 6, 0.98953167040022 02]
[0.498929666441104 87, 1.610277945334311 2, 0.97762564010939 06]
[0.36360023197034 72, 0.75855597016305 82, 0.59796413023455 79]
```

Matrix multiplication simple algorithm

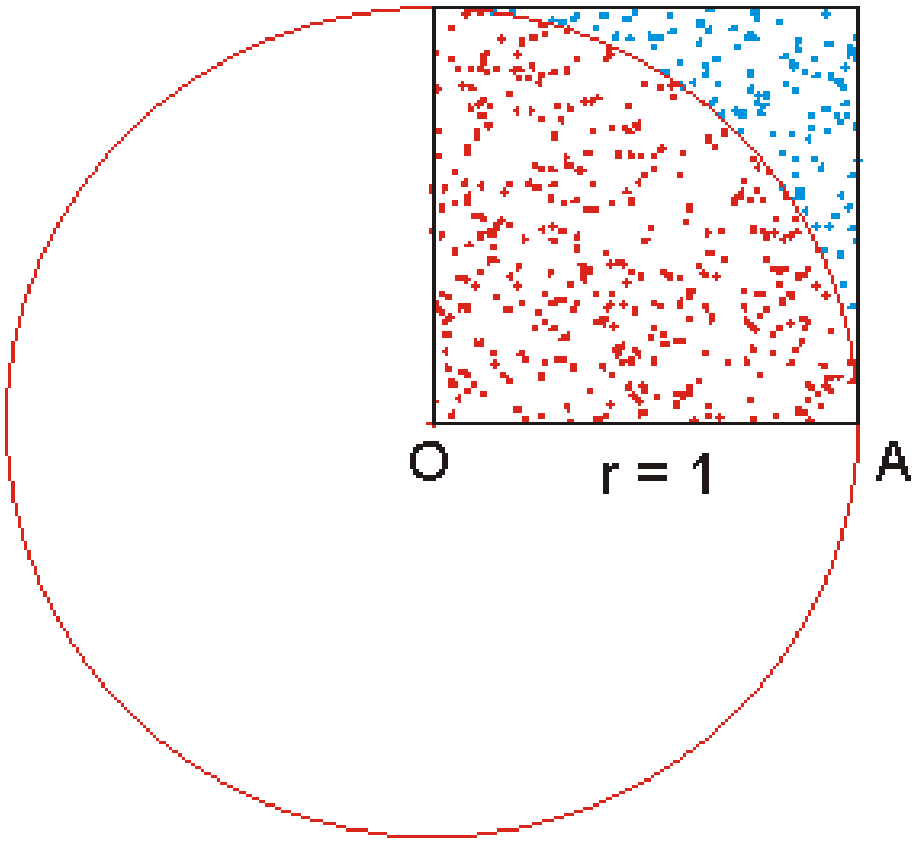
- Input: matrices A and B
- Let C be a new matrix of the appropriate size
- For i from 0 to n-1:
- For j from 0 to p-1:
- Let sum = 0
- For k from 0 to m-1:
- Set sum = sum + $A_{ik} \times B_{kj}$
- Set C_{ij} = sum
- Return C

EXAMPLE MONTECARLO TO COMPUTE PI

Monte Carlo methods

- In the nuclear expressions, a fast particle fully hits the nucleus of an atom. This shatters into many particles, which hit the nuclei of other neighboring atoms, which are shattered in turn, according to a chain reaction, during which a great deal of energy is released. The process will last until it involves the whole universe or will it stop after a certain number of reactions?
- **The Monte Carlo method consists in seeking the solution of a problem, representing it as a parameter of a hypothetical population and in estimating this parameter by examining a sample of the population obtained through sequences of random numbers**

Compute PI using MC methos



$$A_S = (2r)^2 = 4r^2$$

$$A_C = \pi r^2$$

$$\pi = 4 \times \frac{A_C}{A_S}$$

```python
import random
import math
import sys

DIM = 100000

"""
if len(sys.argv) != 2:
    print("usage: ", sys.argv[0] , " NUM ")
    exit(1)
else:
    DIM = int(sys.argv[1])
"""

circle_count = 0

for i in range(0,DIM):

   x = random.uniform(0.0, 1.0)
   y = random.uniform(0.0, 1.0)

   if (math.sqrt((math.pow(x, 2.0) + math.pow(y, 2.0))) < 1.0):
     circle_count = circle_count + 1

pi = float(circle_count) / float(DIM)

print(4.0 * pi)
```

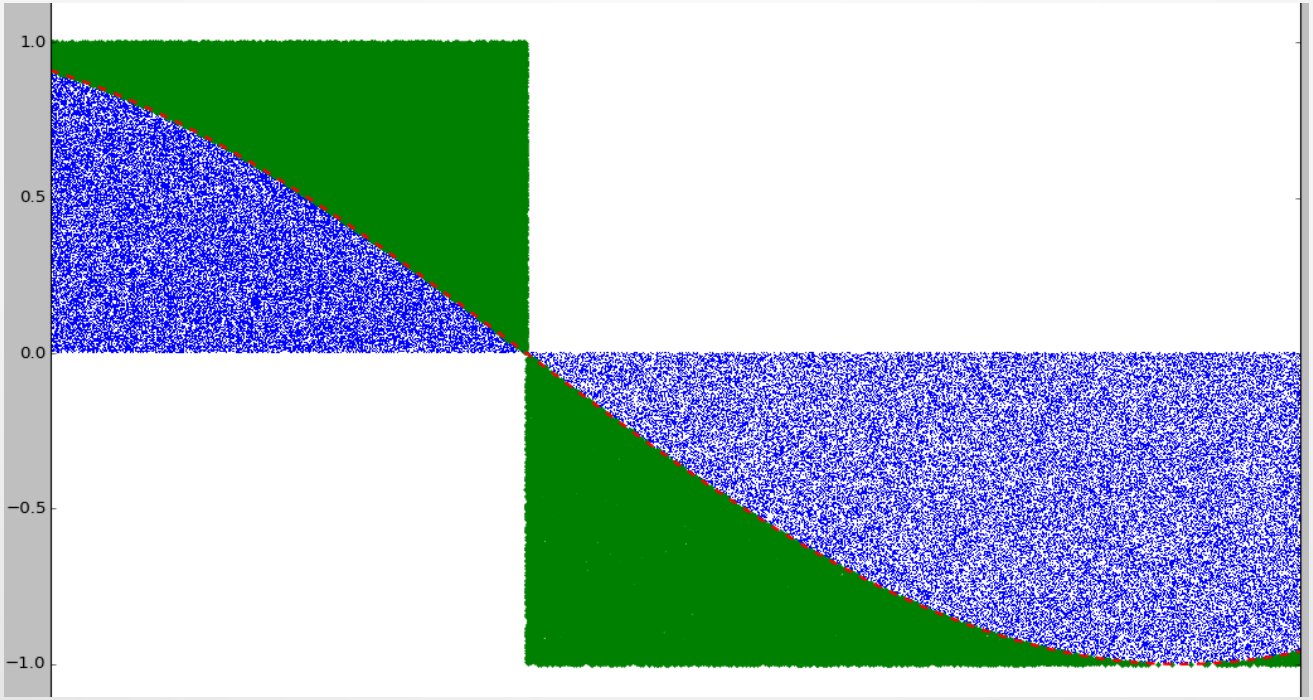
```
[redo@banquo mcpi (master)]$ python mcpi.py 10
2.8
[redo@banquo mcpi (master)]$ python mcpi.py 100
3.48
[redo@banquo mcpi (master)]$ python mcpi.py 1000
3.104
[redo@banquo mcpi (master)]$ python mcpi.py 10000
3.1252
[redo@banquo mcpi (master)]$ python mcpi.py 100000
3.14592
```

MONTECARLO TO COMPUTE AN INTEGRAL

Exercise

- Compute the integral using MC method:

$$\int_2^5 \sin(x)\,dx = \cos(2) - \cos(5) = -0{,}69981$$



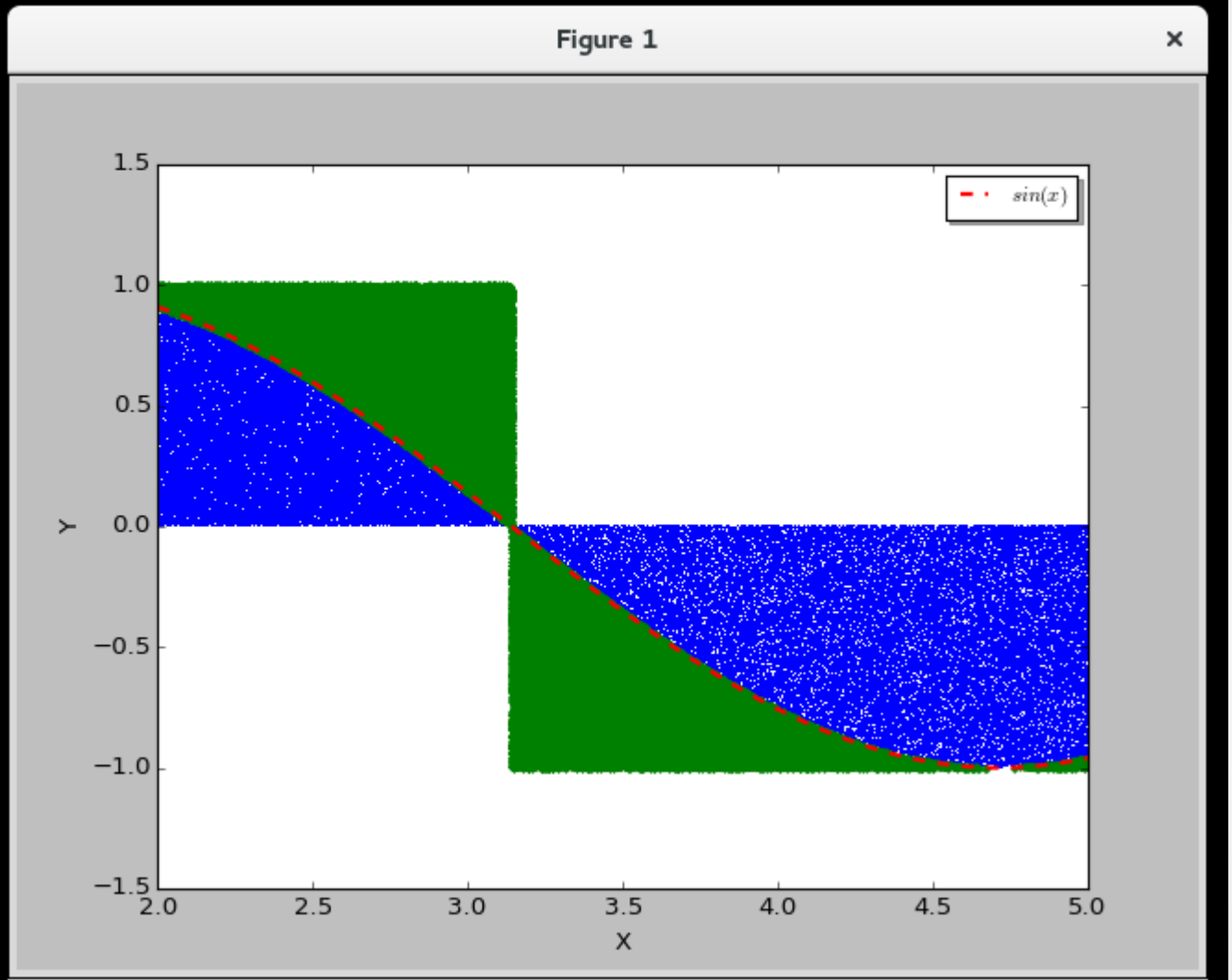
Exercise

$$\int_{2}^{5} \sin() dx = \cos(2) - \cos(5) = -0{,}69981$$

- Set rect_count to 0 , xmin to 2.0, xmax to 5.0 , ymin to -1.0 and ymax to 1.0
- for i from 0 to N :
- Generate x random between xmin and xmax
- Generate y random between ymin and ymax
- if sin(x) > 0.0 and (y <= sin(x) and (y >= 0.0):
- rect_count = rect_count + 1
- else if sin(x) < 0.0 and y >= sin(x) and y <= 0.0
- rect_count = rect_count – 1
- integral is equal to (xmax - xmin) * (ymax - ymin) * (rect_count / N)

```
[redo@banquo mcsin (master)]$ time python mcsin.py 100
-0.18

real    0m0.013s
user    0m0.010s
sys     0m0.003s
[redo@banquo mcsin (master)]$ time python mcsin.py 1000
-0.726

real    0m0.017s
user    0m0.014s
sys     0m0.003s
[redo@banquo mcsin (master)]$ time python mcsin.py 10000
-0.6834

real    0m0.049s
user    0m0.040s
sys     0m0.008s
[redo@banquo mcsin (master)]$ time python mcsin.py 100000
-0.70938

real    0m0.115s
user    0m0.108s
sys     0m0.007s
[redo@banquo mcsin (master)]$ time python mcsin.py 1000000
-0.703062

real    0m1.049s
user    0m1.021s
sys     0m0.027s
[redo@banquo mcsin (master)]$ time python mcsin.py 10000000
-0.7001124

real    0m10.226s
user    0m10.105s
sys     0m0.116s
[redo@banquo mcsin (master)]$
```

```
[redo@banquo mcsin (master)]$ python mcsin_wplt.py 100000
-0.696206603137
```



Let's see a first example of the use of numpy and matplotlib

EXERCISE

Exercise

- Write a program to represent the cos function in the interval -5.0 < x < 5.0