Python Functions and I/O

Loriano Storchi

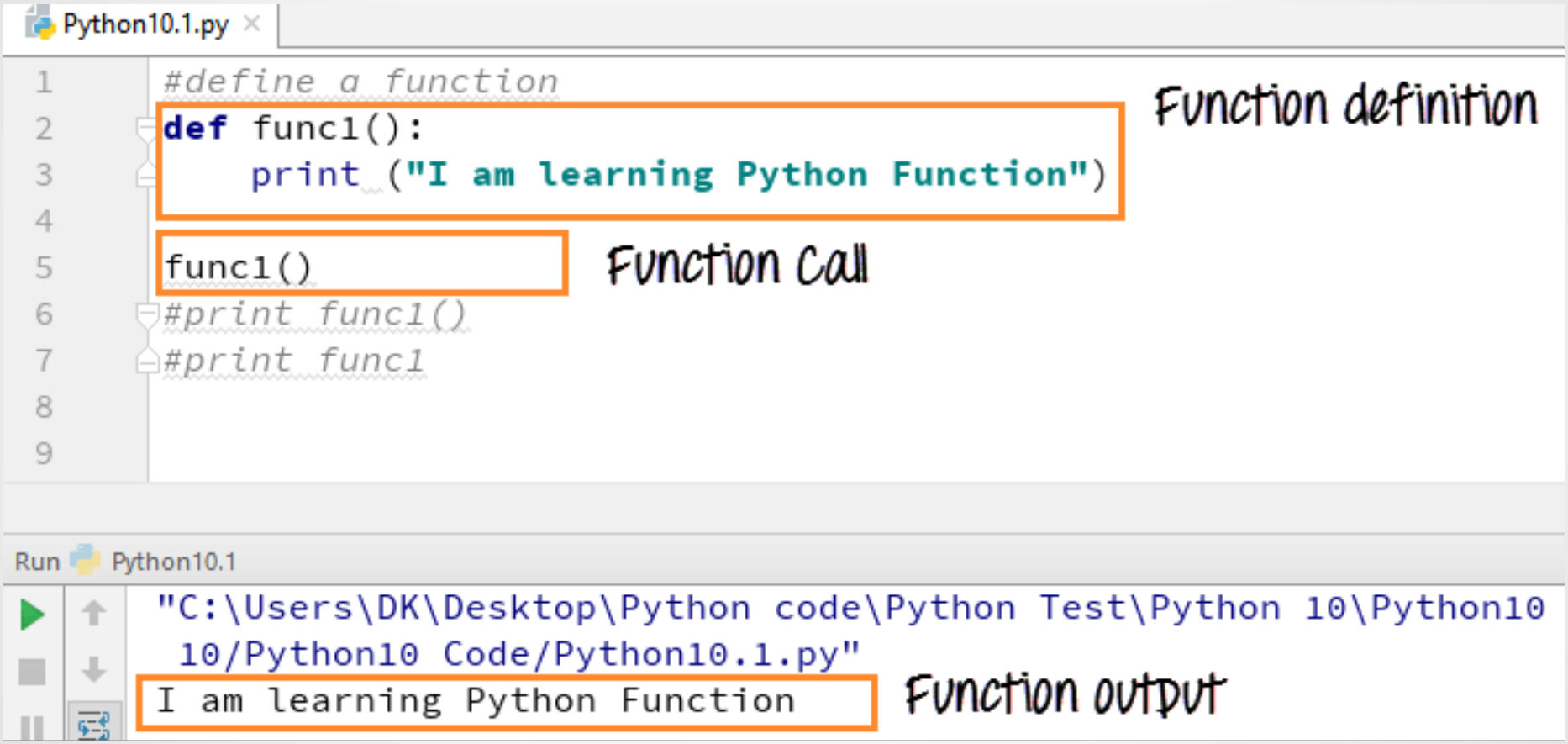
loriano@storchi.org

http::/www.storchi.org/

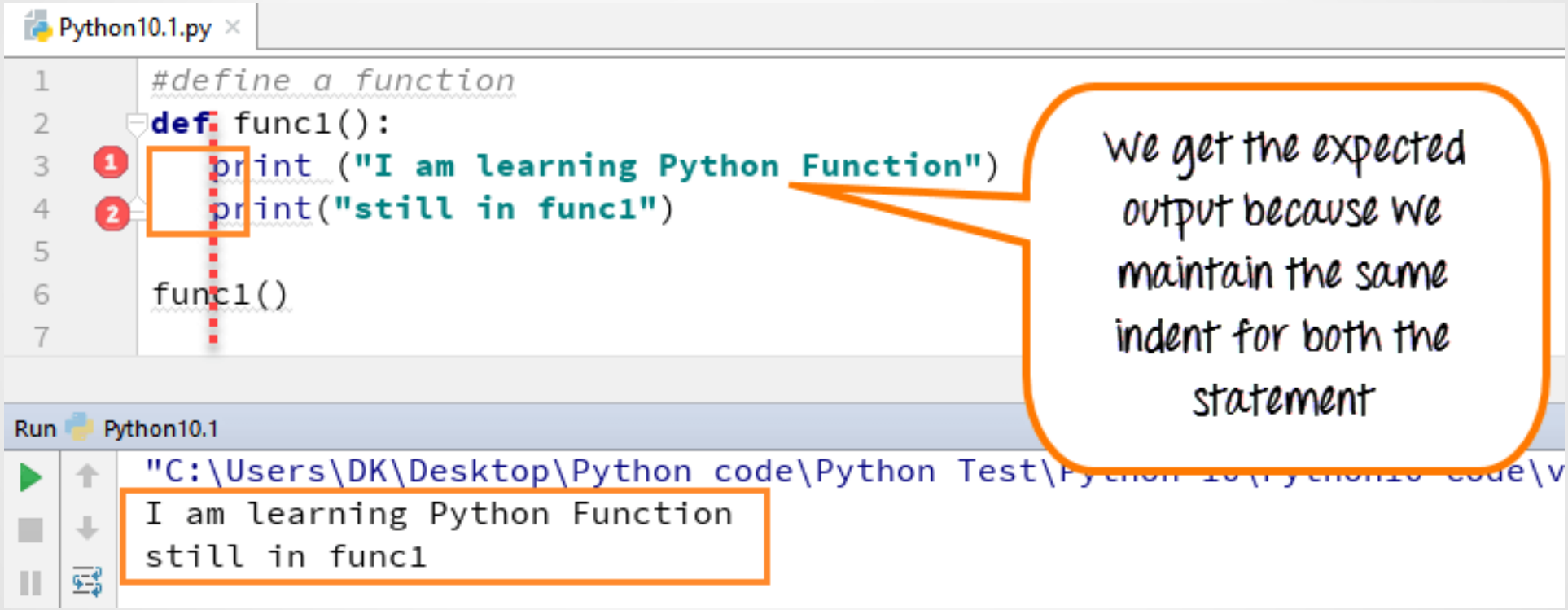
Functions

- A function generally receives parameters that are generally memory addresses or values. A function receives the parameters in input then executes a certain sequence of operations and returns some result
- The parameters (**formal parameters**) **are the variables that I find in the definition of the function**. For example in python: **def function_name (a, b, c)**
- While the **arguments** (**or actual parameters**) **are the actual variables passed in input when the function is called**. For example in python: **function_name (x, y, z)**

Functions


Python10.1.py
#define a function
def func1():
print ("I am learning Python Function")
func1()
#print func1()
#print func1
Function definition
Function Call
Run
Python10.1
"C:\Users\DK\Desktop\Python code\Python Test\Python 10\Python10
10/Python10 Code/Python10.1.py"
I am learning Python Function
Function output

Functions


Python10.1.py
#define a function
def func1():
print ("I am learning Python Function")
print("still in func1")
func1()
We get the expected output because we maintain the same indent for both the statement
Run
Python10.1
"C:\Users\DK\Desktop\Python code\Python Test
I am learning Python Function
still in func1

Functions

Python10.2.py

```
1  def square(x):
2      y=x*x
3
4  print(square(4))
5
6
```

square()

Run Python10.2

```
"C:\Users\DK\Desktop\Python code\P
None
```

```
1  def square(x):
2      return x*x
3
4  print(square(4))
5
6
```

Run Python10.2

```
"C:\Users\DK\Desktop\Pyth
16
```

Here we have used "return command" to return the value of function, which is square of (4) i.e 16

PARAMETERS

Parameters

- **Passing parameters by value** implies that the **actual parameters are copied into the formal parameters**, **and therefore the function works on a copy of the values** when they are modified this is not reflected yes actual parameters
- **Passing the parameters by reference**: in this case **a "pointer" is passed then the function can modify the formal parameter**, for example, and this will affect the current parameter

By value or by reference

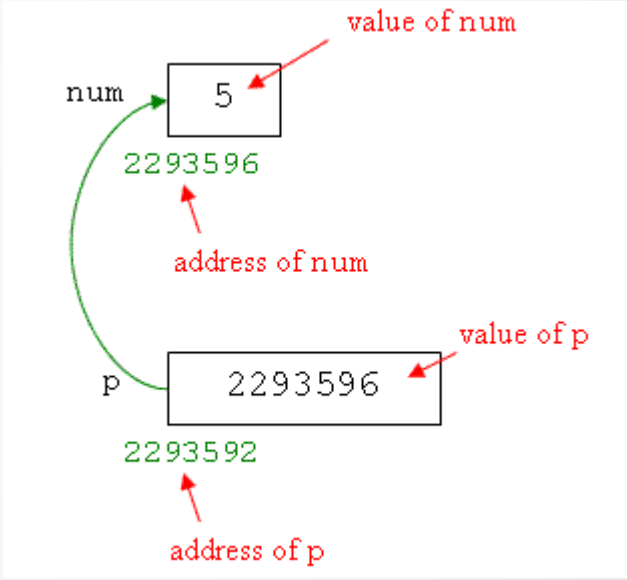
```
#include <iostream>

void funcval (int a, int b)
{
  a = 1;
  b = 1;
}

void funcref (int & a, int & b)
{
  a = 1;
  b = 1;
}

int main (int argc, char ** argv)
{
  int a, b;

  a = 0;
  b = 0;

  funcval (a, b);
  std::cout << "a: " << a << " b: " << b << std::endl;

  funcref (a, b);
  std::cout << "a: " << a << " b: " << b << std::endl;

  return 0;
}
```

```
a: 0 b: 0
a: 1 b: 1
```

Brief digression: pointers

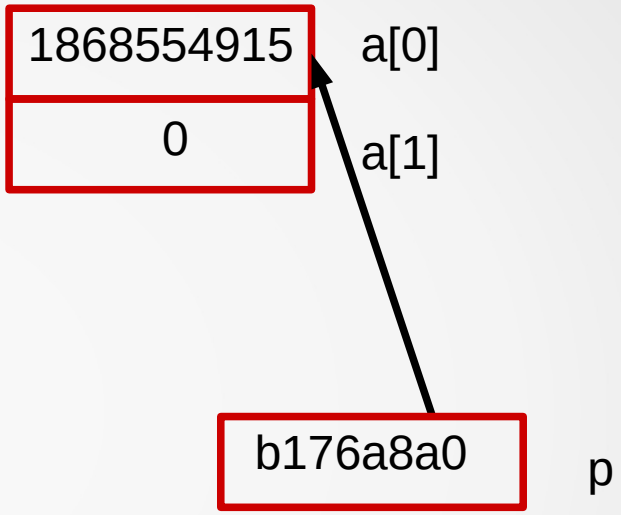
- A pointer is a "special" variable that contains the address to a zone of memory

```
#include <stdio.h>

void func (int * a, int * b)
{
  *a = 1;
  *b = 1;
}

int main (int argc, char ** argv)
{
  int a, b;

  a = 0;
  b = 0;

  func (&a, &b);
  fprintf (stdout, "a: %d b: %d \n", a, b);

  return 0;
}
[redo@banquo functionsc (master)]$ ./pointer
a: 1 b: 1
```



Brief degerssion 2

```c
#include <stdio.h>

int main (int argc, char ** argv)
{
  int a[2];
  char * p;

  a[0] = 1868654915;
  a[1] = 0;

  p = (char *) a;
  fprintf (stdout, "%d %d %d %x\n", sizeof(int),
      sizeof(double), sizeof(p), a);
  fprintf (stdout, "%x %s \n", p, p);

  return 0;
}
```




```
[redo@banquo functionsc (master)]$ ./poir
4 8 8 b176a8a0
b176a8a0 Ciao
```

hexadecimal

PARAMETERS PYTHON

Mutable or not

- Arguments are passed by assigning objects to local names.
- Remember what happens when we assign a value to a variable, in reality we are simply pointing that variable to a given memory location.
- If the object that I pass to the function is mutable I can then modify it
- If the object is not mutable I will not be able to modify it

Mutable or Immutable

- **Immutable Objects** : These are of **in-built types like int, float, bool, string, tuple**. In simple words, an immutable object can't be changed after it is created.

```
tuple1 = (0, 1, 2, 3)
tuple1[0] = 4
print(tuple1)
```

```
Traceback (most recent call last):
  File "test.py", line 2, in <module>
    tuple1[0] = 4
TypeError: 'tuple' object does not supp
```

```
message = "Welcome"
message[0] = 'p'
print(message)
```

```
Traceback (most recent call last):
  File "test.py", line 2, in <module
    message[0] = 'p'
TypeError: 'str' object does not sup
```

Mutable or Immutable

- **Mutable Objects** : These are of type **list, dict, set** . Custom classes are generally mutable.

```
color = ["red", "blue", "green"]
print(color)

color[0] = "pink"
color[-1] = "orange"
print(color)
```

```
['red', 'blue', 'green']
['pink', 'blue', 'orange']
```

Mutable or not

- Mutable and immutable objects are handled differently in python. **Immutable objects are quicker to access and are expensive to change, because it involves creation of a copy. Whereas mutable objects are easy to change.**
- Use of mutable objects is recommended when there is a need to change the size or content of the object.
- As a rule of thumb, **Generally Primitive-like types are probably immutable and Customized Container-like types are mostly mutable**.

Mutable: lists

```
def editlista (lista):
  lista.append("tre")

def assignlista (lista):
  lista = ["nuova", "lista"]

lista = ["uno", "due"]
editlista(lista)
print(lista)
assignlista(lista)
print(lista)
```

```
['uno', 'due', 'tre']
['uno', 'due', 'tre']
```

Immutable

- Python strings are not mutable, so what happens when I pass a string as a parameter to a function?

```
def editstring (valin):
  valin = "output"
  print("modificata in: ", valin)

val = "input"
editstring (val)
print("dopo la call: ", val)
```

```
modificata in:  output
dopo la call:  input
```

When inside the function I write valin = "output" I am creating a new string object that contains the value "output" and I am making valin point to this new memory address

Immutable 2

- if I need to act to change the value of the string passed in input:

```
def editstring (valin):
  valin = "output"
  return valin

val = "input"
val = editstring (val)
print("dopo la call: ", val)
```

```
dopo la call:  output
```

FUNCTIONS PYTHON

Functions

- To define a function in python, use the keyword **def**
- Let's take a simple example a function that calculates the average value given as input a list of numbers

```
def calcola (vals):
  sum = 0.0
  for v in vals:
    sum += v

  return sum/len(vals)

valori = [1.1, 4.0, 5.0, 9.5, 5.6]
m = calcola(valori)
print("valore medio: ", m)
```

```
valore medio:  5.040000000000001
```

Functions

- A function in python is able to return more than one value

Scope: **n** exists only within the function

```
import math

def calcola (vals):
  n = float(len(vals))

  m = 0.0
  for v in vals:
    m += v
  m = m / n

  s = 0.0
  for v in vals:
    s = (v-m)**2
  s = s / n
  s = math.sqrt(s)

  return m, s

valori = [1.1, 4.0, 5.0, 9.5, 5.6]
m, s = calcola(valori)
print("valore medio: ", m , " stdev: ", s)
```

```
valore medio:  5.040000000000001  stdev:  0.25043961347997584
```

Functions

- A function can have default parameters. For example a function that divides all the elements of the list for a given number

```
def divide (vals, d = 2.0):
  res = []
  for v in vals:
    res.append(v / d)

  return res

vals = [1.0, 3.5, 5.6, 7.8]
print(vals)
print(divide(vals))
print(divide(vals, 3.0))
```

```
[1.0, 3.5, 5.6, 7.8]
[0.5, 1.75, 2.8, 3.9]
[0.3333333333333333, 1.1666666666666667, 1.8666666666666665, 2.6]
```

EXERCISE

Exercise 1

- Write a program that calculates the solution of a quadratic equation using a function (you may start from solv.py)

```
[redo@buchner functions]$ python secondord.py
insert a:1
insert b:2
insert c:3
a =  1.0  b =  2.0  c =  3.0
non ci sono soluzioni reali
[redo@buchner functions]$ python secondord.py
insert a:1
insert b:3
insert c:-4
a =  1.0  b =  3.0  c =  -4.0
soluzioni:  1.0 -4.0
```

Exercise 2

- Write a function that: given a list of numbers, a scalar and a character executes, accordingly to the character's value, one of the four fundamental operations +, - *, /

```
vals = [1.0, 3.5, 5.6, 7.8]
print vals
print opera(vals)
print opera(vals, 3.0)
print opera(vals, c = "+")
```

This is the main

```
[redo@buchner functions]$ python operaz.py
[1.0, 3.5, 5.6, 7.8]
[0.5, 1.75, 2.8, 3.9]
[0.3333333333333333, 1.1666666666666667, 1.8666666666666665, 2.6]
[3.0, 5.5, 7.6, 9.8]
```

Exercise 2

- Input list , scalar and character
- for value in list
- If character equal to +
- append to the result list value + scalar
- else if character equal to -
- append to the result list value - scalar
- else if character equal to *
- append to the result list value * scalar
- else if character equal to /
- append to the result list value / scalar
- Return result list

Exercise 2

- We may use also a dictionary where the key is a character and value is a function name (reference)

```
def divido(a, b):
    return a/b

def moltiplico(a, b):
    return a * b

def sommo (a, b):
    return a + b

def sottraggo():
    return a - b

# un semplice dizionario
operazioni = {"/" : divido,
              "*" : moltiplico,
              "+" : sommo,
              "-" : sottraggo,
}
```

```
def opera (vals, d = 2.0, c = "/"):
  res = []
  for v in vals:
    res.append(operazioni[c] (v, d))

  return res
```

```
[redo@buchner functions]$ python operaz2.py
[1.0, 3.5, 5.6, 7.8]
[0.5, 1.75, 2.8, 3.9]
[0.3333333333333333, 1.1666666666666667, 1.8666666666666665, 2.6]
[3.0, 5.5, 7.6, 9.8]
```

FILE I/O

Files

- A file is a container in a computer system for storing information. Files used in computers are similar in features to that of paper documents used in library and office files. There are different types of files such as text files, data files, directory files, binary and graphic files, and these different types of files store different types of information. In a computer operating system, files can be stored on optical drives, hard drives or other types of storage devices.

Input and output: files

- The **open**() function opens a file, and returns it as a file object.

  f = open (“/path/nomefile”, ”r”)

  r = read-only

  w = to write and ASCII file

  a = append mode

  the **close** method can be used to close a file: f.close()

Read from a file

- The method **readlines**() reads until **EOF** using **readline**() and **returns a list containing the lines**. If the optional sizehint argument is present, instead of reading up to EOF, whole lines totalling approximately sizehint bytes (possibly after rounding up to an internal buffer size) are read.
- The **method seek() sets the file's current position at the offset**. The whence argument is optional and defaults to 0, which means absolute file positioning, other values are 1 which means seek relative to the current position and 2 means seek relative to the file's end.

Read from a file : iofileread.py

```
import sys

filename = ""

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

f = open (filename, "r")

l = f.readline()
print(l)
lines = f.readlines()
print(lines)

f.seek(0, 0) # offset primo parametro, when
             # 1 posizione relativa rispett
             # 2 poszione relativa alla fin

for l in f:
  print(l)

f.close()
```

```
line1

['line2\n', 'line3\n', 'line4\n', 'li
line1

line2

line3

line4

line5

line6
```

Read from a file : iofileread.py

```
import sys

filename = ""

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

f = open (filename, "r")

l = f.readline()
print(l)
lines = f.readlines()
print(lines)

f.seek(0, 0) # offset primo parametro, when
             # 1 posizione relativa rispett
             # 2 poszione relativa alla fin

for l in f:
  print(l)

f.close()
```

```
line1

['line2\n', 'line3\n', 'line4\n', 'li
line1

line2

line3

line4

line5

line6
```

Read from a file : iofileread.py

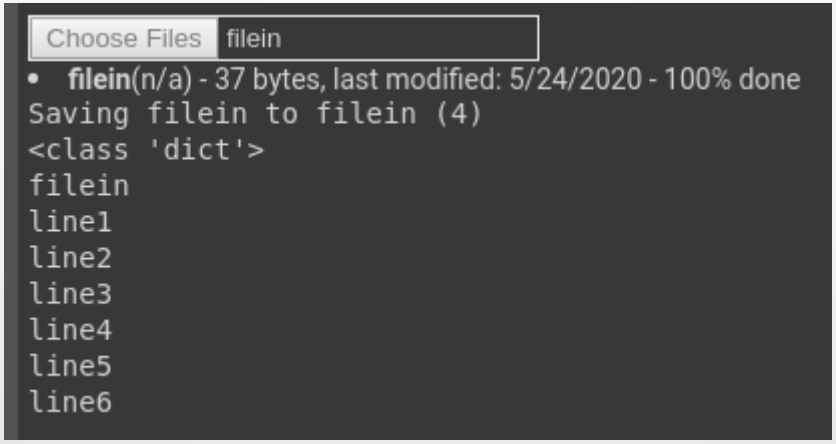
The **method seek() sets the file's current position at the offset**.

```
f.seek(1, 0) # offset pri
             # 1 posizion
             # 2 poszione
l = f.readline()
print l
```

```
ine1
```

Read from a file Colab

```
from google.colab import files

uploaded = files.upload()
print(type(uploaded))

for fname in uploaded.keys():
  print(fname)
  for l in uploaded[fname].decode("utf-8").split("\n"):
    print(l)
```

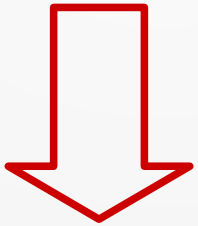


Write into a file

- **write**() to writes one line at a Time to a File in Python
- **writelines**(): Writing All The Lines at a Time to a File
- We may use also **print** to write something to a file in python

Write into a file: iofilewrite.py

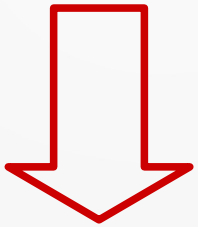
```
f = open (filename, "w")

lista = ["linea1", "linea2", "li
for l in lista:
  f.write(l+"\n")

f.writelines(lista)
f.close()
```



```
[redo@buchner iofiles (master)]$ python iofilewrite.py test.txt
[redo@buchner iofiles (master)]$ cat test.txt
linea1
linea2
linea3
linea1linea2linea3[redo@buchner iofiles (master)]$
```

Write into a file: iofilewrite1.py

```
f = open (filename, "w")

lista = ["linea1", "linea2", "linea3"]
for l in lista:
    print(l, file=f)

f.close()
```



```
[redo@buchner iofiles (master)]$ python iofilewrite1.py test.txt
[redo@buchner iofiles (master)]$ cat test.txt
linea1
linea2
linea3
```

Write into a file: Colab

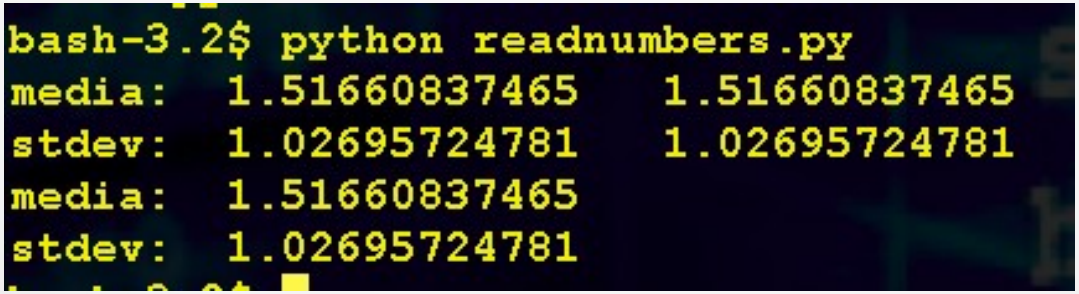
```
from google.colab import files

filename = "test.txt"
f = open (filename, "w")

lista = ["linea1", "linea2", "linea3"]
for l in lista:
  f.write(l+"\n")

f.writelines(lista)
f.close()
files.download(filename)
```


```
└─ $ cat test.txt
linea1
linea2
linea3
linea1linea2linea3|redo@buchner /home/redo/Downloads
```

EXERCISE

Exercise

- Read n numbers from a file calculates mean and standard deviation (suggestion maybe you'll need to use the split method). Calculation of the standard deviation see Welford method (also in Donald Knuth's Art of Computer Programming) and see numpy
- https://www.storchi.org/lecturenotes/ipfi/8/numbers.txt



Exercise

- varianceavg(values):
- moldm = values[0] mnewm = values[0]
- molds = 0.0 mnews = 0.0
- for k from 1 to N:
- x = values[i]
- mnewm = moldm + (x - moldm)/(i+1)
- mnews = molds + (x - moldm)*(x - mnewm)
- moldm = mnewm
- molds = mnews
- s = math.sqrt(mnews/(i+1))
- m = mnewm
- return S/(N-1)

Exercise: or use the naive algorithm

- Let n ← 0, Sum ← 0, SumSq ← 0
- For each data in x:
- n ← n + 1
- Sum ← Sum + x
- SumSq ← SumSq + x × x
- Var = (SumSq − (Sum × Sum) / n) / (n − 1)