



# Python for Beginners

Modulo III

Alessio Miaschi

Dipartimento di Informatica, Università di Pisa  
ItaliaNLP Lab, Istituto di Linguistica Computazionale (ILC-CNR)

[alessio.miaschi@phd.unipi.it](mailto:alessio.miaschi@phd.unipi.it)  
<https://pages.di.unipi.it/miaschi/>

**I moduli (o  
librerie)**

# I moduli

- I moduli (o librerie) sono file python contenenti gruppi di funzioni correlate, utilizzati generalmente per operare su dati specifici
- Python include una lista di moduli predefiniti, detti standard (*standard library*)
- È però possibile scaricarne e definirne di nuovi

# I moduli

- Per accedere ai contenuti di un modulo, è necessario **importarlo** all'interno del nostro programma, usando il costrutto **import**

```
import math

def main():
    numero = 25
    radice = math.sqrt(numero)
    print(numero)

main()

# >>> 5.0
```

# I moduli

- Per accedere ai contenuti di un modulo, è necessario **importarlo** all'interno del nostro programma, usando il costrutto **import**

```
import math

def main():
    numero = 25
    radice = math.sqrt(numero)
    print(numero)

main()

# >>> 5.0
```

# I moduli

- Una volta importato il modulo, sarà possibile accedere alle sue funzioni ricorrendo alla sintassi **<nome\_modulo>.<nome\_funzione>**

```
import math

def main():
    numero = 25
    radice = math.sqrt(numero)
    print(numero)

main()

# >>> 5.0
```

# I moduli

- Digitando **help(<nome\_modulo>)** è possibile esplorare tutti i contenuti del modulo

```
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module is always available. It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.

    acosh(...)
        acosh(x)

        Return the inverse hyperbolic cosine of x.

    asin(...)
        asin(x)

        Return the arc sine (measured in radians) of x.

    asinh(...)
        asinh(x)
```

# I moduli

- È possibile importare anche solo alcune funzioni di un determinato modulo, usando il costrutto **from <nome\_modulo> import <nome\_funzione>**

```
from math import sqrt

def main():
    numero = 25
    radice = sqrt(numero)
    print(numero)

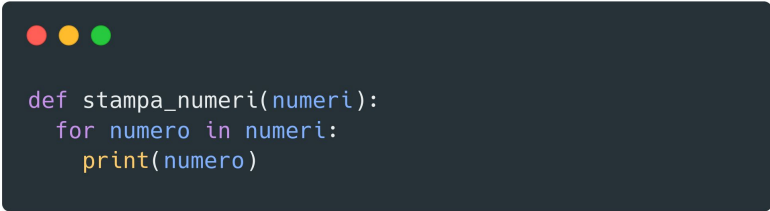
main()

# >>> 5.0
```



# Creare un nuovo modulo

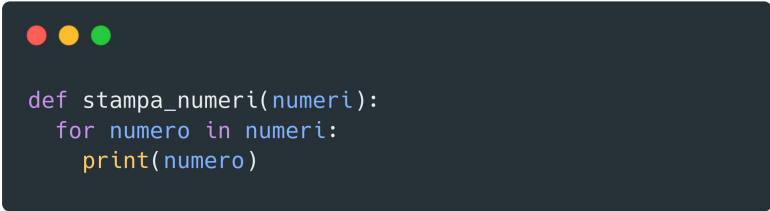
- Per creare un nuovo modulo basta creare un file python (.py), definire una (o più) funzioni e importare il file stesso (usando il *namefile*) all'interno del nostro programma

A dark-themed code editor window with three colored window control buttons (red, yellow, green) in the top-left corner. It contains a Python function definition with a loop.

```
def stampa_numeri(numeri):  
    for numero in numeri:  
        print(numero)
```

# Creare un nuovo modulo

- Per creare un nuovo modulo basta creare un file python (.py), definire una (o più) funzioni e importare il file stesso (usando il *namefile*) all'interno del nostro programma



```
def stampa_numeri(numeri):  
    for numero in numeri:  
        print(numero)
```

operazioni\_liste.py

# Creare un nuovo modulo

- Per creare un nuovo modulo basta creare un file python (.py), definire una (o più) funzioni e importare il file stesso (usando il *namefile*) all'interno del nostro programma

```
def stampa_numeri(numeri):  
    for numero in numeri:  
        print(numero)
```

operazioni\_liste.py

```
import operazioni_liste
```

```
lista = [1, 2, 3, 4]  
operazioni_liste.stampa_numeri(lista)
```

```
# >>> 1
```


```
# >>> 2
```

```
# >>> 3
```

```
# >>> 4
```

# Creare un nuovo modulo

- Per creare un nuovo modulo basta creare un file python (.py), definire una (o più) funzioni e importare il file stesso (usando il *namefile*) all'interno del nostro programma



```
def stampa_numeri(neri):  
    for numero in neri:  
        print(numero)
```

operazioni\_liste.py

```
import operazioni_liste
```

```
lista = [1, 2, 3, 4]  
operazioni_liste.stampa_numeri(lista)
```

```
# >>> 1  
# >>> 2  
# >>> 3  
# >>> 4
```

# Scaricare nuovi moduli

- È possibile scaricare e installare nuovi moduli ricorrendo a:
  - comando pip (pip3) direttamente da terminale;
  - Tramite le impostazioni di Pycharm

```
alessio@alessio:~$ pip install torch
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: torch in ~/.local/lib/python3.6/site-packages (1.7.0)
Requirement already satisfied: dataclasses in /usr/local/lib/python3.6/dist-packages (from torch) (0.6)
Requirement already satisfied: future in ~/.local/lib/python3.6/site-packages (from torch) (0.18.2)
Requirement already satisfied: numpy in ~/.local/lib/python3.6/site-packages (from torch) (1.19.5)
Requirement already satisfied: typing-extensions in ~/.local/lib/python3.6/site-packages (from torch) (3.7.4.3)
```

# Scaricare nuovi moduli

- È possibile scaricare e installare nuovi moduli ricorrendo a:
  - comando pip (pip3) direttamente da terminale;
  - Tramite le impostazioni di Pycharm

```
alessio@alessio:~$ pip install torch
```

```
Defaulting to user installation because normal site-packages is not writeable
```

```
Requirement already satisfied: torch in ./local/lib/python3.6/site-packages (1.7.0)
```

```
Requirement already satisfied: dataclasses in /usr/local/lib/python3.6/dist-packages (from torch) (0.6)
```

```
Requirement already satisfied: future in ./local/lib/python3.6/site-packages (from torch) (0.18.2)
```

```
Requirement already satisfied: numpy in ./local/lib/python3.6/site-packages (from torch) (1.19.5)
```

```
Requirement already satisfied: typing-extensions in ./local/lib/python3.6/site-packages (from torch) (3.7.4.3)
```

# Espressioni regolari

# Espressioni regolari

- Linguaggio standard per caratterizzare stringhe di testo (regular expressions, regex, re)
- Strumento ideale per:
  - ricercare testo;
  - sostituire testo.
- Molti programmi supportano le RE:
  - *Word*;
  - comando *grep* in Unix
  - *Emacs*, *Geany* e altri editor di testo



# Espressioni regolari

- Il pattern di un'espressione regolare si definisce tramite una **RE**
- La **RE** viene verificata se un testo produce come risultato:
  - **True**: il testo contiene una stringa che corrisponde al pattern (*match*)
  - **False**: il testo non contiene una stringa che corrisponde al pattern
- In python, il modulo **re** fornisce le operazioni necessarie per trattare le espressioni regolari

# Espressioni regolari



```
import re

def main():
    stringa = "Hello world!"
    lista_match = re.findall(r'world', stringa)
    for match in lista_match:
        print("Match: ", match)

main()

# >>> Match: world
```

# Espressioni regolari


```
import re

def main():
    stringa = "Hello world!"
    lista_match = re.findall(r'world', stringa)
    for match in lista_match:
        print("Match: ", match)

main()

# >>> Match: world
```

Cerca il pattern  
"world" in  
stringa



# Espressioni regolari

- Un qualsiasi carattere o sequenza di caratteri è una RE
- Le RE sono “*case sensitive*”

RE	Match
r'testo'	'il <b>testo</b> del file'
r'mark up'	'il <b>mark up</b> del testo'
r'Python'	' <b>Python</b> è un linguaggio di programmazione' 'python è un linguaggio di programmazione'

# Espressioni regolari

- Un insieme di caratteri tra parentesi quadre è una RE che definisce una classe di **caratteri disgiunti**

RE	Definizione	Match
<code>r'[st]'</code>	's' o 't'	' <b>s</b> intassi', 'py <b>t</b> hon'
<code>r'[0123456789]'</code>	qualsiasi cifra	' <b>8</b> ore'
<code>r'[Cc]iao'</code>	'Ciao' o 'ciao'	' <b>C</b> iao, sono Alessio', 'Ti ho detto <b>c</b> iao'

# Espressioni regolari

- Utilizzando il trattino (-) è possibile specificare un **intervallo** di caratteri

RE	Definizione	Match
<code>r'[a-z]'</code>	qualsiasi lettera minuscola	' <b>sintassi</b> ', ' <b>python</b> '
<code>r'[0-9]'</code>	qualsiasi cifra	' <b>8</b> ore'
<code>r'[a-zA-Z]'</code>	qualsiasi lettera minuscola o maiuscola	' <b>Ciao sono Alessio</b> '

# Espressioni regolari

- Se vogliamo che un pattern non contenga un determinato carattere possiamo usare **^** all'interno di una classe

RE	Definizione	Match
<code>r'^a-z'</code>	qualsiasi carattere diverso da una lettera minuscola	'Sintassi', ' <b>8</b> ore', 'tempo'
<code>r'^st'</code>	qualsiasi carattere diverso da 's' e 't'	' <b>2 parole</b> ', 'ssssssss'

# Abbreviazioni

RE	Classe di caratteri
<code>r'\d'</code>	<code>r'[0-9]'</code>
<code>r'\w'</code>	<code>r'[a-zA-Z0-9_]'</code>
<code>r'\s'</code>	<code>r'[\t\n]'</code>
<code>r'\D'</code>	<code>r'^[0-9]'</code>
<code>r'\W'</code>	<code>r'^[a-zA-Z0-9_]'</code>
<code>r'\S'</code>	<code>r'^[\t\n]'</code>
<code>\t</code>	tabulazione
<code>\n</code>	ritorno a capo



# Alternativa

- L'operatore `|` esprime la disgiunzione fra due RE (operatore di alternativa)

RE	Definizione	Match
<code>r'cane gatto'</code>	la stringa 'cane' oppure la stringa 'gatto'	'il <b>cane</b> abbaia', 'il <b>gatto</b> miagola'

# Moltiplicatori

- I seguenti simboli sono usati per specificare quante volte deve comparire un carattere:
  - **?** : il carattere precedente è opzionale
  - **\*** : il carattere precedente occorre 0 o  $n$  volte
  - **+** : il carattere precedente occorre 1 o  $n$  volte

RE	Definizione	Match
r'tokens?	l'ultimo carattere 's' è opzionale	'token', 'tokens', 'tokened'
r'ba*	il carattere 'b' seguito da 0 o $n$ 'a'	'b', 'ba', 'baaaaaa', 'c'

# Moltiplicatori

RE	Definizione	Match
<code>r'ba+'</code>	il carattere 'b' seguito da <i>1</i> o <i>n</i> 'a'	'b', 'ba', 'baaaaaaaaaa'
<code>r'[0-9][0-9]*'</code>	un numero infinitamente lungo che deve contenere almeno una cifra	'2', '234', '7654357', 'ciao'

# Moltiplicatori avanzati

- Moltiplicatori avanzati:
  - **{n,m}**: il carattere deve comparire almeno  $n$  e massimo  $m$  volte
  - **{n,}**: il carattere deve comparire almeno  $n$  volte
  - **{n}**: il carattere deve comparire  $n$  volte

RE	Definizione	Match
<code>r'a{2,3}b'</code>	la stringa formata da almeno 2/massimo 3 'a' e poi una 'b'	'aab', 'aaab', 'ab'
<code>r'a{2}b'</code>	la stringa formata da 2 'a' e una 'b'	'ab', 'aab', 'aaab'

# Wildcard

- La RE `r'.'` corrisponde a qualsiasi carattere

RE	Definizione	Match
<code>r'b.s'</code>	qualsiasi stringa di tre caratteri che inizia con 'b' e finisce con 's'	' <b>bas</b> ', ' <b>bbs</b> ', 'baas'
<code>r'b.*s'</code>	qualsiasi stringa che inizia per 'b' e finisce per 's'	' <b>bs</b> ', ' <b>bas</b> ', ' <b>bbs</b> ', ' <b>bisogna prendere l'autobus</b> '

# Raggruppamento e memoria

- Usando le parentesi tonde seguite dai simboli **?:** è possibile raggruppare stringhe di caratteri da moltiplicare

RE	Definizione	Match
<code>r'(?ab)+'</code>	una o più stringhe 'ab'	' <b>ab</b> ', ' <b>abab</b> ', 'acb'

- Le parentesi tonde senza **?:** *memorizzano* la stringa di testo contenuta nelle parentesi:
  - Il contenuto della variabile può essere richiamato con `\<numero>` (`\1` contenuto prima coppia di parentesi, `\2` seconda coppia, ecc.)

# Ancore

- Le ancore specificano dove deve comparire il pattern di testo da cercare:
  - **'^<pattern>'**: il <pattern> deve comparire all'inizio della linea
  - **'<pattern>\$'**: il <pattern> deve comparire alla fine della linea
- Il carattere speciale **\b** è un'ancora che indica il confine di parola

RE	Definizione	Match
r'\bcane\b'	'cane' deve avere a sinistra e a destra un confine di parola	'Il <b>cane</b> è ...', 'Il canestro è...'

# Sostituzione

- La funzione **re.sub()** permette di sostituire il pattern di una stringa con un un nuova stringa

```
import re

def main():
    stringa = "Ciao, sono alessio."
    nuovaStringa = re.sub(r'alessio', r'Alessio', stringa)
    print(nuovaStringa)

main()

# >>> 'Ciao, sono Alessio'
```



# Sostituzione

- La funzione **re.sub()** permette di sostituire il pattern di una stringa con un un nuova stringa

```
import re

def main():
    stringa = "Ciao, sono alessio."
    nuovaStringa = re.sub(r'alessio', r'Alessio', stringa)
    print(nuovaStringa)

main()

# >>> 'Ciao, sono Alessio'
```

# Esercizi

- Formalizzare con le espressioni regolari i pattern per trovare le seguenti stringhe:
  - tutte le vocali minuscole o maiuscole
  - le parole che iniziano per consonante
  - le parole che terminano con un segno di punteggiatura
  - le parole che iniziano per “per” o “pr”
  - il passato prossimo del verbo “vedere”
  - le date dove giorno, mese e anno sono espressi da numeri
  - I numeri formati da una cifra pari e una dispari alternata
  - Le parole in fondo ad una linea di testo che terminano per ”sto” o per “sito”

# Esercizi

- Usando l'operatore di sostituzione:
  - verticalizzare le parole, in modo che ogni parola venga scritta su una riga diversa
  - trasformare tutte le parole che iniziano per vocale e che contengono almeno 4 caratteri con la stringa "WORD"
  - trasformare i numeri invertendo la prima cifra con l'ultima (es. 123456 in 623451)
  - nelle date con il mese scritto in lettere, sostituire spazi con underscore (es. 27 maggio 2021 diventa 27\_maggio\_2021)
  - trasformare tutte le parole del testo che iniziano per vocale invertendo il primo carattere con l'ultimo (es. *esercizio* diventa *osercizie*)
- Altri esercizi su espressioni regolari: <https://www.w3resource.com/python-exercises/re/>

# Elaborazione di file di testo

# Il modulo **sys**

- Il modulo **sys** ci permette di interagire con il sistema operativo su cui stiamo lavorando
- Uno degli utilizzi principali di questo modulo è quello di passare degli *argomenti* (stringhe, file di testo, ecc) al nostro programma direttamente da riga di comando

# Il modulo **sys**

- Il modulo **sys** ci permette di interagire con il sistema operativo su cui stiamo lavorando
- Uno degli utilizzi principali di questo modulo è quello di passare degli *argomenti* (stringhe, file di testo, ecc) al nostro programma direttamente da riga di comando

**python** programma.py argomento\_1 argomento\_2 argomento\_n

# Il modulo sys



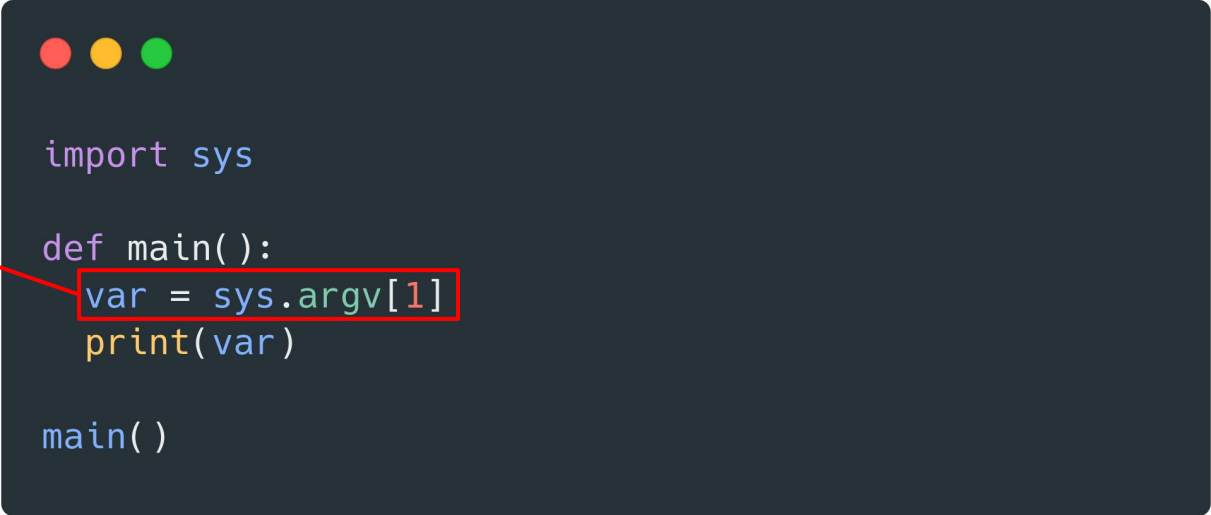
```
import sys

def main():
    var = sys.argv[1]
    print(var)

main()
```

# Il modulo sys

Primo  
argomento

A dark-themed code editor window with three colored window control buttons (red, yellow, green) in the top-left corner. The code is as follows:

```
import sys

def main():
    var = sys.argv[1]
    print(var)

main()
```

The line `var = sys.argv[1]` is highlighted with a red rectangular border. A red arrow points from the text "Primo argomento" to the `1` in `sys.argv[1]`.



# Il modulo sys

Primo  
argomento

```
import sys
```

```
def main():
```

```
    var = sys.argv[1]
```

```
    print(var)
```

```
main()
```

```
alessio@alessio:~/Desktop$ python programma.py argomento
argomento
alessio@alessio:~/Desktop$
```

# Il modulo sys




```
import sys

def main():
    num1 = sys.argv[1]
    num2 = sys.argv[2]
    print(num1, num2)

main()
```

# Il modulo sys



```
import sys

def main():
    num1 = sys.argv[1]
    num2 = sys.argv[2]
    print(num1, num2)

main()
```

```
alessio@alessio:~/Desktop$ python programma.py 10 20
10 20
alessio@alessio:~/Desktop$
```

# Apertura/chiusura di un file

- Per aprire un file in python si utilizza la funzione **open()**
- La funzione **open()** prende in input più argomenti, ma i più importanti sono:
  - **nome del file:** percorso che mi indica la posizione del file
  - **modalità:** ovvero se stiamo aprendo un file in modalità di lettura o di scrittura (**r**, **w** o **a**)
  - **codifica:** la codifica dei caratteri da utilizzare per la lettura del file
- Quando abbiamo finito di eseguire determinate operazione su un file, è sempre consigliato chiuderlo utilizzando il metodo **<nome\_file>.close()**

# Apertura/chiusura di un file



```
file1 = open('file_di_testo.txt', 'r', encoding='utf-8')  
print(file1)
```

```
# >>> <_io.TextIOWrapper name='prova.txt' mode='r' encoding='utf-8'>
```

# Apertura/chiusura di un file

nome del file

```
file1 = open('file_di_testo.txt', 'r', encoding='utf-8')  
print(file1)
```

```
# >>> <_io.TextIOWrapper name='prova.txt' mode='r' encoding='utf-8'>
```

modalità di apertura

codifica dei caratteri

# Apertura/chiusura di un file



```
file1 = open('file_di_testo.txt', 'r', encoding='utf-8')  
print(file1)  
file1.close()  
print(file1)
```

```
# >>> <_io.TextIOWrapper name='prova.txt' mode='r' encoding='utf-8'>  
# >>> <_io.TextIOWrapper name='prova.txt' mode='r' encoding='utf-8'>
```

# Modalità di scrittura

- Utilizzando **w** (*writing*) come modalità di apertura, potremo non solo leggere il contenuto del file ma anche scriverci all'interno
- Nel caso in cui il file non esistesse, l'apertura tramite modalità **w** creerà automaticamente un nuovo file
- **ATTENZIONE:** aprendo un file con la modalità **w** il contenuto precedentemente salvato all'interno del file andrà perso
  - Per poter aggiungere del contenuto, il file dovrà essere aperto con la modalità **a**



# Metodi dei file object

Metodo	Descrizione
<code>file.read()</code>	Legge e restituisce l'intero file in una stringa
<code>file.read(<i>n</i>)</code>	Legge e restituisce <i>n</i> caratteri dei file
<code>file.readline()</code>	Legge e restituisce una riga del file
<code>file.readlines()</code>	Legge e restituisce l'intero file (lista di righe)
<code>file.write(<i>s</i>)</code>	Scrive la stringa <i>s</i> all'interno del file
<code>file.close()</code>	Chiude il file

# Esercizi

- Scrivere un programma che legga un file di testo con la seguente struttura:

```
Pippo 22  
Pluto 34  
Alessio 29  
...
```

e memorizzi il suo contenuto all'interno di un dizionario (es. {"Pippo": 22, "Pluto": 34, "Alessio": 29})

# Esercizi

- Scrivere un programma che legga un file di testo contenente un nome per riga e scriva all'interno di un nuovo file i nomi e la loro rispettiva lunghezza in termini di caratteri. Ad esempio, dato il file:

Alessio  
Pippo  
Pluto  
Ciao

il file di output dovrà presentare la seguente struttura:

Alessio, lunghezza = 7  
Pippo, lunghezza = 5  
Pluto, lunghezza = 5  
Ciao, lunghezza = 4

# Esercizi

- Scrivere un programma che prenda in input un file e che restituisca su un nuovo file i vari paragrafi con i nomi propri anonimizzati (es. “Biden è andato [...]” con “NOME\_PROPRIO è andato [...]”)
  - Si può assumere che le parole a inizio frase non siano nomi propri!
- Scrivere un programma che prenda in input due file di testo e che restituisca in output la *Type/Token Ratio* (TTR) di entrambi. La TTR è così calcolata:

$$TTR = \frac{V}{T_n}$$

dove  $V$  corrisponde al vocabolario (il totale delle parole **diverse** nel testo) e  $T_n$  è il totale delle parole nel testo