



Python

Dalla A alla Z per programmatori

CyberChallenge 2024,
Università degli Studi di Napoli Parthenope



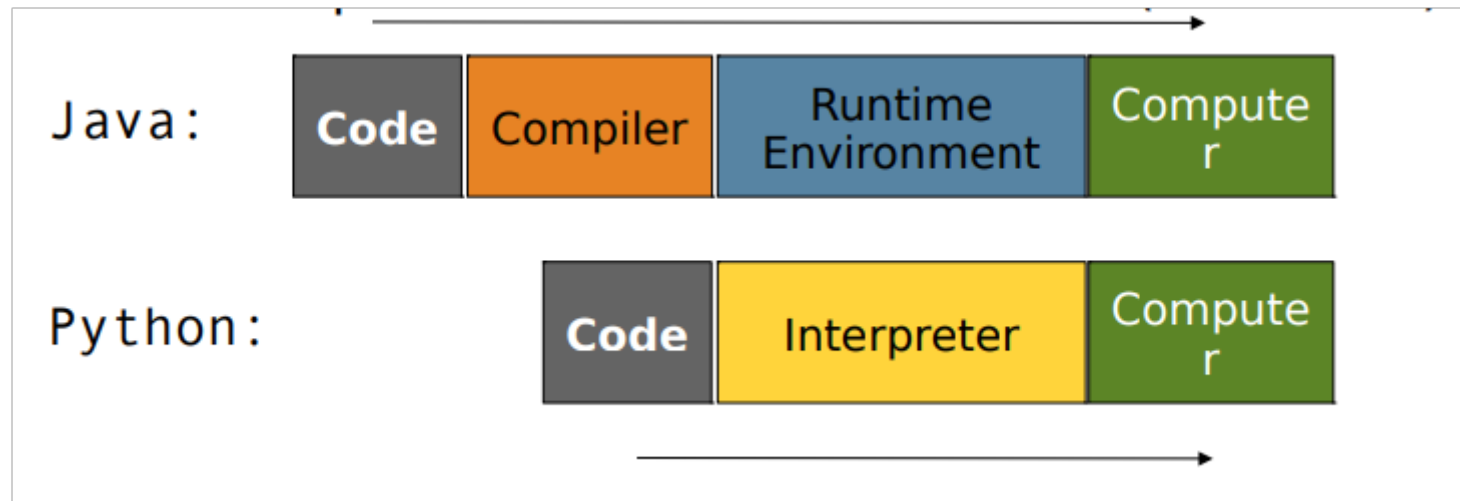
Python!

- Creato nel 1991 da Guido van Rossum. Il nome deriva da Monty Python, gruppo comico inglese.
- Usato da: Google, Yahoo!, Youtube, Molte distribuzioni Linux, Giochi e app. Recentemente diffuso in ambito scientifico (vedi ML).



Linguaggi Interpretati

- interpretato
 - Non compilato come Java, C, C++
 - Codice scritto ed eseguito direttamente da un **interprete**
- E' possibile scrivere comandi direttamente nell'interprete e osservarne il risultato



Vediamo qualche esempio...

- Interprete

```
C:\Users\colui>python3 -c "print('Hello World')"  
Hello World
```

- Shell

```
C:\Users\colui>python3  
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019,  
01:54:44) [MSC v.1916 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more  
information.  
>>> print("Hello World")  
Hello World  
>>>
```

- Script


```
C:\Users\colui>echo print('Hello World') > hello.py  
C:\Users\colui>python3 hello.py  
Hello World
```

M – Multilinee commenti/istruzioni

- Un commento è preceduto da # ma può essere espresso su più linee includendolo tra ''' e ''' (tre apici)
 - Un commento multilinea che segue l'intestazione di una funzione sarà usato come documentazione quando viene invocato il comando help(nomefunzione)
- In python ogni linea contiene una istruzione (quindi non servono delimitatori per le istruzioni, come il ; del C/C++/Java). Una istruzione può continuare su più linee usando il simbolo \ per indicare che non è terminata
 - Più istruzioni possono essere sulla stessa linea separate da ;

```
print('Ciao mondo')  
#questo è un commento  
'''anche questo  
è un commento'''
```

```
print('Ciao \n  
mondo')
```



Primo programma in Python

- Python non ha un metodo *main*. In alcuni altri linguaggi di programmazione una funzione main è necessaria per avviare il programma, ma non in Python.
- Il codice del programma e' semplicemente scritto nel file che verra' eseguito (file sorgente)
 - In Python le righe non finiscono con ;

hello.py

```
1 print("Hello, world!")
```

Variabili

- Una variabile è creata nel momento in cui è usata per la prima volta

```
1 x = "Hello World"
2 print(x)
```

Hello World

- Per conoscere il tipo di una variabile usare `type()`. Tutti i tipi sono oggetti: questo significa che possono avere metodi e attributi

```
1 x = "Hello World"
2 print(type(x))
```

<class 'str'>

- Tipi disponibili

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict

Set Types:	set
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview



Variabili

- **Tipizzazione Dinamica:** In Python, non è necessario dichiarare esplicitamente il tipo di una variabile; il tipo è assegnato dinamicamente in base al valore attribuito:

```
x = 5          # Intero  
y = "Ciao"     # Stringa  
z = 3.14       # Float
```

- **Assegnazione Multipla:** Python consente di assegnare più variabili in una singola riga.

```
a, b, c = 1, 2.5, "Python"
```

- Tipi disponibili

Boolean e altri tipi

- E' possibile esplicitare il tipo del dato mediante casting

```
1 x = "3.8"
2 y = 10 + x
3 print(y)
```

Errore... 10
numerico e x
stringa

```
1 x = "3.8"
2 y = 10 + float(x)
3 print(y)
```

OK! x «castato»
a float

- Operandi

+	Addition (a+b oppure a+=b)
-	Subtraction (come sopra)
*	Multiplication (c.s.)
/	Division (c.s.)
%	Modulus (c.s.)
**	Exponentiation (c.s.)
//	Floor division (c.s.)

==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

&	AND
	OR
^	XOR
~	NOT
<<	Zero fill left shift
>>	Signed right shift

Stringhe

- In Python, una stringa è una sequenza di caratteri racchiusa tra apici singoli '...' o doppi "...".

```
testo1 = "Ciao, mondo!"  
testo2 = 'Python è potente'  
print(testo1)
```

- Puoi unire due stringhe con +:

```
nome = "Mario"  
saluto = "Ciao, " + nome + "  
print(saluto) # Output: Ciao, Mario!
```

- Puoi ripetere una stringa con *:

```
print("Python! " * 3) # Output: Python! Python! Python!
```

Funzioni e Metodi Utili per le Stringhe

- Le stringhe sono indicizzate, puoi accedere ai caratteri con []:

```
testo = "Python"
print(testo[0]) # Output: P
print(testo[-1]) # Output: n (ultimo carattere)
```

```
s = 'le stringhe sono array'
print(s[:10])
```

le stringh

Metodo	Descrizione	Esempio
<code>len()</code>	Lunghezza della stringa	<code>len("Python") → 6</code>
<code>.lower()</code>	Converti in minuscolo	<code>"CIAO".lower() → "ciao"</code>
<code>.upper()</code>	Converti in maiuscolo	<code>"ciao".upper() → "CIAO"</code>
<code>.strip()</code>	Rimuove spazi	<code>" ciao ".strip() → "ciao"</code>
<code>.replace("a", "o")</code>	Sostituisce caratteri	<code>"palla".replace("a", "o") → "pollo"</code>
<code>.split(" ")</code>	Divide in lista	<code>"ciao mondo".split() → ["ciao", "mondo"]</code>

Collezioni di dati: liste e tuple (segue...)

Liste: ordinate
(indici) e modificabili

```
x = ['pippo', 3, True, 12]
print(x)
print(x[1:3])
print(x[:2])
print(x[1:])
print(x[-3:-1])
print(x[0:len(x):2])
x[2] = False
print(x)
```

```
['pippo', 3, True, 12]
[3, True]
['pippo', 3]
[3, True, 12]
[3, True]
['pippo', True]
['pippo', 3, False, 12]
```

*Estremo sinistro incluso e destro escluso

Sono consentiti duplicati!!!

Tuple: ordinate e
non-modificabili

```
x = ("pippo", 3, True, 12)
print(x)
print(x[1])
print(x[1:3])
print(x[:2])
print(x[1:])
print(x[-3:-1])
x[2] = False
print(x)
```

Errore!!!
Non
modificabile

Collezioni di dati: liste e tuple (segue...)

- Per ordinare una lista due soluzioni
 - `list.sort()` #modifica la lista riordinandola
 - `sorted(list)` #ritorna una nuova lista riordinata

Liste come STACK – Principio LIFO

```
>>> stack = [3,4,5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
```

Liste come Code - principio FIFO

```
>>> from collections import deque
>>> queue = deque([3,4,5])
>>> queue.append(6)
>>> queue.append(7)
>>> queue
deque([3, 4, 5, 6, 7])
>>> queue.popleft()
3
>>> queue.popleft()
4
>>> queue
deque([5, 6, 7])
```

*dalla versione 3.7 sono ordinati in base all'ordine di inserimento



2 strutture dati fondamentali: Dizionari e Insiemi (Set)

Dizionari: non ordinate* (no indici), modificabili, accesso per nome

```
x = {'nome': "Luigi",  
     'cognome': "Coppolino",  
     'eta': 42}  
print(x)  
print(x['cognome'])  
x['nome'] = 'Peppe'  
print(x.get('nome'))
```

```
{'nome': 'Luigi', 'cognome': 'Coppolino', 'eta': 42}  
Coppolino  
Peppe
```

Set: non ordinate (no indici), no ripetizioni

```
x = {'luigi', 'marco', 'teo'}; print(x)  
x.add('nico'); print(x)  
x.update(['anna', 'mia']); print(x)  
x.discard('luigi'); print(x)  
y = {'teo', 'lia'}  
x.update(y); print(x)  
x.clear(); print(x)
```

```
{'marco', 'luigi', 'teo'}  
{'marco', 'nico', 'luigi', 'teo'}  
{'anna', 'nico', 'marco', 'mia', 'luigi', 'teo'}  
{'anna', 'nico', 'marco', 'mia', 'teo'}  
  
{'anna', 'nico', 'marco', 'mia', 'lia', 'teo'}  
set()
```



Ciclo for

- Per elencare gli elementi di una Collezione (o una stringa) possiamo usare il costrutto *for*
 - Notare che l'indentazione determina il blocco di codice da iterare

```
x = {'luigi', 'marco', 'secret', 'teo'}  
for i in x:  
    if i == 'secret':  
        continue  
    print(i)
```

teo
luigi
marco

- Per iterare un numero finito di volte utilizzeremo la funzione range()

```
for i in range(10):  
    if i in range(2,8,3):  
        continue  
    print(i)
```

0
1
3
4
6

7
8
9



I - IF

- Si usa nel modo classico

```
a = 10

if a < 18:
    print("minorenne")
elif a < 25:
    print("vota alla camera")
else:
    print("vota a camera e senato")
```

- and, or, not, in, not in possono essere usati per comporre le clausole di condizione

- Oppure nelle forma Short Hand

```
a = 21
print('minorenne') if a < 18 else print('maggiorenne')
print("non vota") if a < 18 else print("camera") if a < 25 else print("camera e senato")
```


Qualche esercizio

1. Filtra numeri da una lista

- Crea una lista con 5 numeri (es. `[5, 12, 3, 18, 7]`).
- Stampa solo i numeri maggiori di 10.
- *Nota:* puoi usare un ciclo `for` e un `if` per controllare i valori.

2. Dizionario film

- Crea un dizionario con informazioni su un film, ad esempio:

python

Copia Modifica

```
film = {  
    "titolo": "Inception",  
    "regista": "Christopher Nolan",  
    "anno": 2010,  
    "genere": "Fantascienza"  
}
```

- Aggiungi una nuova coppia chiave-valore (ad esempio `"valutazione": 8`).
- Stampa il dizionario aggiornato.

3. Ordinare una lista di stringhe

- Crea una lista di 5 nomi di città (es. `["Roma", "Milano", "Napoli", "Torino", "Bologna"]`).
- Ordina la lista in ordine alfabetico.
- Stampa la lista ordinata.
- *Nota:* puoi usare il metodo `sort()` sulle liste (ad esempio `lista.sort()`). Non è una definizione di funzione, ma un metodo già presente in Python.

Qualche esercizio (Soluzioni)

Soluzione 1

```
# Stampa solo i numeri maggiori di 10
numeri = [5, 12, 3, 18, 7]
for num in numeri:
    if num > 10:
        print(num)
```

```
# Crea un dizionario di un film, aggiungi una chiave e stampa
film = {
    "titolo": "Inception",
    "regista": "Christopher Nolan",
    "anno": 2010,
    "genere": "Fantascienza"
}
film["valutazione"] = 8
print(film)
```

Soluzione 3

```
# Ordina una lista di città in ordine alfabetico e stampala
citta = ["Roma", "Milano", "Napoli", "Torino", "Bologna"]
citta.sort()
print(citta)
```

E- Elencare con Comprehension

- Una **Comprehension** è fatta da parentesi quadre seguite da una espressione del genere:
[espressione **for** elemento **in** iterable **if** condizione]
- Le parentesi quadre indicano che il risultato è una lista.
- L'espressione definisce il valore che verrà inserito nella lista.
- Il for cicla sugli elementi di un iterable (ad es. una lista o range()).
- Un if opzionale può filtrare gli elementi

E- Elencare

- Una Comprehension è fatta da parentesi quadre seguite da una espressione del genere:

[espressione **for** elemento **in** iterable **if** condizione]

```
x = [i for i in range(10)]  
print(x)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
x = [i for i in range(10) if i % 2 == 0]  
print(x)
```

```
[0, 2, 4, 6, 8]
```

```
x = [1,2,3,4]  
y = [2,3]  
print([(i,j) for i in x for j in y if i != j])
```

```
[(1, 2), (1, 3), (2, 3), (3, 2),  
(4, 2), (4, 3)]
```



Esercizio

- Inizializzare una lista con i quadrati dei primi 10 numeri naturali, utilizzando due diverse soluzioni: un ciclo for tradizionale e una list comprehension.
- Cose da sapere:
 - `range(n)` --- Genera numeri da 0 a n-1
 - `x**2` calcola il quadrato del numero x

Esercizio Soluzione

- Inizializzare una lista con i quadrati dei primi 10 numeri naturali

```
quadrati = []  
for n in range(11):  
    if n > 0:  
        quadrati.append(n ** 2)  
  
print(quadrati)
```

Soluzione 1

```
quadrati = [i**2 for i in range(11) if i > 0]  
print(quadrati)
```

Soluzione 2

Esercizio 2 (HackerRank)

Let's learn about list comprehensions! You are given three integers x , y and z representing the dimensions of a cuboid along with an integer n . Print a list of all possible coordinates given by (i, j, k) on a 3D grid where the sum of $i + j + k$ is not equal to n . Here, $0 \leq i \leq x$; $0 \leq j \leq y$; $0 \leq k \leq z$. Please use list comprehensions rather than multiple loops, as a learning exercise.

Example

$x = 1$

$y = 1$

$z = 2$

$n = 3$

All permutations of $[i, j, k]$ are:

$[[0, 0, 0], [0, 0, 1], [0, 0, 2], [0, 1, 0], [0, 1, 1], [0, 1, 2], [1, 0, 0], [1, 0, 1], [1, 0, 2], [1, 1, 0], [1, 1, 1], [1, 1, 2]]$.

Print an array of the elements that do not sum to $n = 3$.

$[[0, 0, 0], [0, 0, 1], [0, 0, 2], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 2]]$

Input Format

Four integers x , y , z and n , each on a separate line.

Constraints

Print the list in lexicographic increasing order.



Esercizio 2 (HackerRank)



Sample Input 0

```
1
1
1
2
```

Sample Output 0

```
[[0, 0, 0], [0, 0, 1], [0, 1, 0], [1, 0, 0], [1, 1, 1]]
```


Esercizio 3

Obiettivo: Data una lista di nomi, creare una nuova lista che contenga solo i nomi che iniziano con la lettera "M".

Istruzioni:

1. Crea una lista di nomi, ad esempio `["Marco", "Luca", "Marina", "Anna", "Marta"]`.
2. Usa una list comprehension con una condizione (`if`) per filtrare i nomi che iniziano con "M".
3. Stampa la nuova lista.

Usare funzione: `variabile.startswith(carattere_indicato)`

La funzione `startswith()` controlla se una stringa inizia con la sequenza di caratteri indicata.

```
nomi = ["Marco", "Luca", "Marina", "Anna", "Marta"]
nomi_m = [nome for nome in nomi if nome.startswith("M")]
print(nomi_m)
```

Esercizio 2 (HackerRank): Soluzione 1



```
x = int(input())  
y = int(input())  
z = int(input())  
n = int(input())
```

Generazione della lista di coordinate con list comprehension

[espressione **for** elemento **in** iterable **if** condizione]

```
result = [ [i, j, k] for i in range(x + 1) # i varia da 0 a x.  
            for j in range(y + 1)         # j varia da 0 a y.  
            for k in range(z + 1)         # k varia da 0 a z  
            if i + j + k != n ]           # esclude le combinazioni dove la somma delle coordinate è n.  
  
print(result)
```

Esercizio 2 (HackerRank): Soluzione 2



```
result = []
for i in range(x + 1):
    for j in range(y + 1):
        for k in range(z + 1):
            if i + j + k != n:
                result.append([i, j, k])

print(result)
```

- **Perché usare una List Comprehension?**

Più compatto rispetto ai loop annidati tradizionali.

Più efficiente, in quanto ottimizzato a livello di interprete Python.

Migliore leggibilità quando il concetto è semplice, come in questo caso.

Funzioni

- Sintassi:

```
def name () :  
    statement  
    statement  
    ...  
    statement
```



hello2.py

```
1  # Prints a helpful message.  
2  def hello():  
3      print("Hello, world!")  
4  
5  # main (calls hello twice)  
6  hello()  
7  hello()
```

- Deve essere dichiarata al di sopra del codice “principale” (main)
- Espressioni all’interno delle funzioni DEVONO essere INDENTATI (ad es. ogni riga inizia con una o piu’ tabulazioni)

Funzioni

- Python usa le indentazioni per indicare i blocchi di codice invece di {}
 - Questo rende il codice più leggibile
 - In Java l'indentazione è opzionale. In Python essa è obbligatoria!

```
hello3.py
1  # Prints a helpful message.
2  def hello():
3  ➡➡ print("Hello, world!")
4  ➡➡ print("How are you?")
5
6  # main (calls hello twice)
7  hello()
8  hello()
```

F — Functions: i parametri

- Si definiscono mediante la keyword `def`
- Parametri passati per posizione o per nome

```
def myFunc(a, b):  
    print(a/b)  
  
myFunc(4, 2)      2.0  
myFunc(b=4, a=2) 0.5
```

```
def myFunc(a, b, *c):  
    somma = 0  
    for x in c:  
        somma += x  
    return a / b + somma
```

```
print(myFunc(4, 2, 1))      3.0  
print(myFunc(4, 2, 1, 2, 3)) 8.0
```

- Le funzioni possono ricevere parametri e restituire uno o più valori con `return`.
- si possono ritornare uno o più valori con `return` (separare i valori da ritornare mediante virgola ,)
- **nome* consente di passare una lista di un numero indeterminato di parametri

- ****** per passare un numero di argomenti per nome (dizionario)

```
def myFunc(**c):  
    x=list()  
    for x,y in c.items():  
        print(x + " is " + str(y))  
    print("num1/den1 is {}".format(c.get("num1")/c.get("den1")))  
  
myFunc(num1=4, den1=2, num2=4, den2=2)
```

```
num1 is 4  
den1 is 2  
num2 is 4  
den2 is 2  
num1/den1 is 2.0
```



F – Functions: i parametri

Metodo	Cosa fa	Tipo di dati
<code>def func(a, b)</code>	Parametri posizionali	Valori fissi
<code>def func(a, b=2)</code>	Parametri con default	Se non specificato, usa il valore predefinito
<code>def func(*args)</code>	Accetta argomenti variabili	Tupla
<code>def func(**kwargs)</code>	Accetta argomenti con nome	Dizionario

- Usa `*args` se vuoi **passare più valori** senza sapere quanti.
- Usa `**kwargs` se vuoi **passare valori con chiavi specifiche**.

Ordine di esecuzione del codice in Python

In Python, non è necessario avere una funzione `main()` come in altri linguaggi come C o Java. Tuttavia, Python segue una logica ben definita per determinare quale codice eseguire per primo.

- **Interpreta il codice riga per riga**
- **Esegue il codice globale** :Quando un file Python viene eseguito, tutto il codice che si trova **fuori dalle funzioni** viene eseguito immediatamente.
- **Ignora definizioni di funzioni fino a quando non vengono chiamate**
Quando l'interprete incontra una funzione, **non la esegue immediatamente**, ma la registra per poterla chiamare più tardi.
- **Esegue solo il codice direttamente chiamato**
Se una funzione non viene chiamata esplicitamente, il suo codice non verrà eseguito.

Ordine di esecuzione del codice in Python

```
print("Questo verrà eseguito subito") # Codice globale → eseguito immediatamente

def funzione():
    print("Questa funzione non verrà eseguita automaticamente")

class Classe:
    print("Questa riga dentro la classe verrà eseguita quando la classe viene letta")

funzione() # Ora la funzione viene chiamata ed eseguita
```

```
Questo verrà eseguito subito
Questa riga dentro la classe verrà eseguita quando la classe viene letta
Questa funzione non verrà eseguita automaticamente
```

Se Voglio simulare un «main»?

- Per evitare che il codice venga eseguito se il file è importato come modulo in un altro script, si usa il blocco `if __name__ == "__main__":` :

```
def main():  
    print("Questo è il codice principale")  
  
if __name__ == "__main__":  
    main() # Questa riga viene eseguita solo se il file è eseguito direttamente
```

- Se eseguiamo il file direttamente (python script.py), il codice dentro `if __name__ == "__main__":` verrà eseguito.
- Se il file viene importato in un altro script con « import script », il codice nel blocco `if __name__ == "__main__":` non verrà eseguito automaticamente.

Come funziona `if __name__ == "__main__":`:

- In Python, ogni script ha una variabile speciale chiamata `__name__`.
- Se esegui direttamente il file, `__name__` assume il valore `"__main__"`.
- Se il file viene importato in un altro script, `__name__` assume il nome del file (script in questo caso).

```
File: script.py

python

def funzione():
    print("Questa è una funzione.")

print("Questo viene eseguito sempre.")

if __name__ == "__main__":
    print("Il file è eseguito direttamente.")
    funzione()
```



```
python script.py
```



```
Questo viene eseguito sempre.
Il file è eseguito direttamente.
Questa è una funzione.
```

Come funziona `if __name__ == '__main__':`:

- In Python, ogni script ha una variabile speciale chiamata `__name__`.
- Se esegui direttamente il file, `__name__` assume il valore `"__main__"`.
- Se il file viene importato in un altro script, `__name__` assume il nome del file (script in questo caso).

```
File: main.py  
  
python  
  
import script  
print("Questo è il file main.py")
```



Python main.py



```
Questo viene eseguito sempre.  
Questo è il file main.py
```

- Nota: Il codice dentro `if __name__ == '__main__':` non viene eseguito perché il file è stato importato e non eseguito direttamente.

Come funziona `if __name__ == '__main__':`:

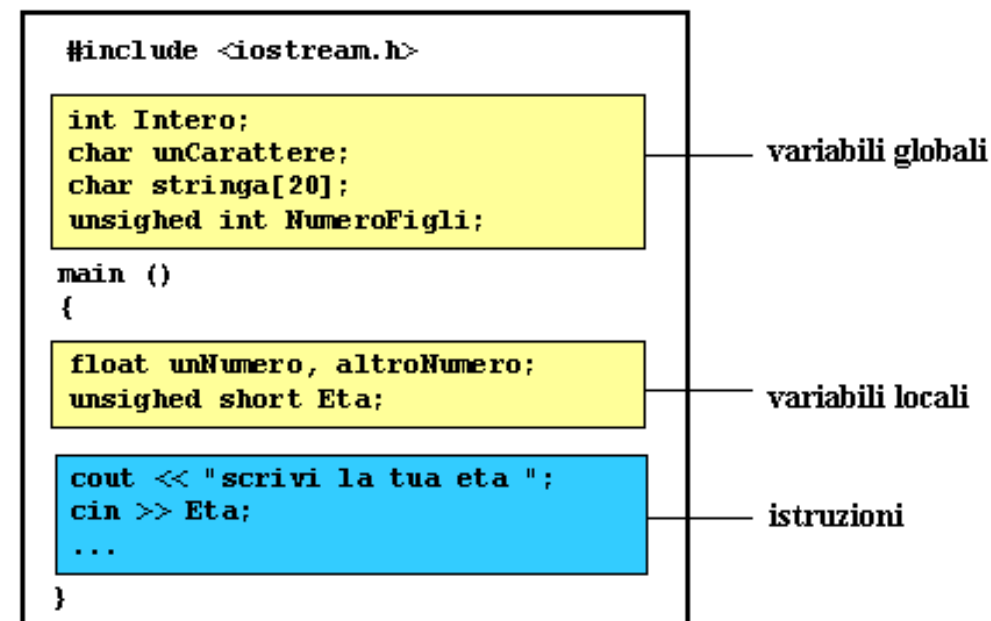
- `if __name__ == '__main__':`: serve per evitare che codice non necessario venga eseguito quando un file è importato come modulo.
- È utile per separare il codice eseguibile dalla definizione di funzioni e classi che possono essere riutilizzate in altri script.

G – Global e regole di visibilità

- In Python, la visibilità delle variabili dipende dal contesto in cui vengono definite. Esistono due tipi principali di variabili:

1. Variabili locali → dichiarate all'interno di una funzione e visibili solo all'interno di essa.

2. Variabili globali → dichiarate fuori da qualsiasi funzione e accessibili da più funzioni.



G – Global e regole di visibilità

```
x = 10 # Variabile globale

def funzione1():
    x = 5 # Questa è una variabile locale (nasconde la globale)
    print("Dentro funzione1, x =", x)

def funzione2():
    global x # Modifica la variabile globale x
    x = 20
    print("Dentro funzione2, x =", x)

# Chiamata delle funzioni
funzione1()
print("Dopo funzione1, x globale =", x)

funzione2()
print("Dopo funzione2, x globale =", x)
```

CHE
RISULTATO TI
ASPETTI?



G – Global e regole di visibilità

```
x = 50 # Variabile globale

def esterna():
    x = 20 # Variabile locale nella funzione esterna
    def interna():
        nonlocal x
        x += 10
        print("Dentro interna, x =", x)

    interna()
    print("Dopo interna in esterna, x =", x)

esterna()
print("Fuori da tutte le funzioni, x =", x)
```

CHE
RISULTATO TI
ASPETTI?



G – Global e regole di visibilità

```
y = 100 # Variabile globale

def modifica():
    global y
    y += 50
    print("Dentro modifica(), y =", y)

def crea_locale():
    y = 30 # Variabile locale (diversa da quella globale)
    print("Dentro crea_locale(), y =", y)

crea_locale()
print("Dopo crea_locale(), y globale =", y)

modifica()
print("Dopo modifica(), y globale =", y)
```

CHE
RISULTATO TI
ASPETTI?

Esercizi con funzioni

Esercizio 1 (Media)

- Scrivi una funzione che accetti una lista di numeri.
- All'interno della funzione, crea un dizionario con le chiavi "massimo", "minimo" e "media".
- Calcola il valore di ogni chiave usando i numeri nella lista (es. massimo = `max(lista_numeri)`).
- Ritorna il dizionario e stampalo fuori dalla funzione.

```
def analizza_numeri(lista_numeri):  
    info = {}  
    info["massimo"] = max(lista_numeri)  
    info["minimo"] = min(lista_numeri)  
    info["media"] = sum(lista_numeri) / len(lista_numeri)  
    return info  
  
numeri = [10, 5, 8, 20, 3]  
risultato = analizza_numeri(numeri)  
print(risultato)
```

H - Handling exceptions

- `try...except...finally` sono usate per gestire le eccezioni
 - Scrivi nel blocco `try` il codice che potrebbe sollevare un'eccezione
 - Nel blocco `except` metti il codice da eseguire se si verifica l'eccezione
 - Il blocco `finally` contiene il codice che deve essere eseguito in ogni caso, sia che si sia verificato un errore sia che non si sia verificato.

```
try:
    x = int(input("Inserisci un numero: "))
    risultato = 10 / x
    print("Risultato:", risultato)
except ValueError:
    print("Errore: devi inserire un numero.")
except ZeroDivisionError:
    print("Errore: divisione per zero.")
finally:
    print("Operazione terminata.")
```

- Se non inserisci un numero, scatta `ValueError`.
- Se dividi per zero, scatta `ZeroDivisionError`.
- Il codice in `finally` si esegue sempre, anche se ci sono errori.



Esercizi di

Esercizi

- S
- e
- I
- N
- f

```
def calcolatrice(a, b, operazione):  
    try:  
        if operazione == '+':  
            return a + b  
        elif operazione == '-':  
            return a - b  
        elif operazione == '*':  
            return a * b  
        elif operazione == '/':  
            return a / b  
        else:  
            return "Operazione non valida"  
    except ZeroDivisionError:  
        return "Errore: divisione per zero"  
  
try:  
    num1 = float(input("Primo numero: "))  
    num2 = float(input("Secondo numero: "))  
    op = input("Operazione (+, -, *, /): ")  
    risultato = calcolatrice(num1, num2, op)  
    print("Risultato:", risultato)  
except ValueError:  
    print("Errore: devi inserire valori numerici validi.")
```

da



Esercizi con funzioni

```
def somma_intervallo(inizio, fine):  
    # Assicuriamoci che inizio sia il minore  
    if inizio > fine:  
        inizio, fine = fine, inizio  
  
    totale = 0  
    for num in range(inizio, fine + 1):  
        totale += num  
    return totale  
  
val1 = 5  
val2 = 2  
print("Somma dei numeri tra", val1, "e", val2, ":", somma_intervallo(val1, val2))
```

Loop and Lambda functions

- Le lambda functions in Python sono funzioni anonime espresse in una sola riga. Si definiscono con la parola chiave lambda, seguita dai parametri e due punti, quindi dall'espressione da calcolare.

```
x = lambda a,b: a + b  
  
print(x(3,2))  
print(x("ciao ", "mamma"))
```

5
ciao mamma

- Uso con map o filter per trasformare o filtrare elementi di una lista

```
numeri = [1, 2, 3, 4]  
# Aumenta ogni numero di 1  
incrementati = list(map(lambda x: x + 1, numeri))  
print(incrementati) # Output: [2, 3, 4, 5]
```

```
numeri = [1, 2, 3, 4, 5, 6]  
# Tieni solo i numeri pari  
pari = list(filter(lambda x: x % 2 == 0, numeri))  
print(pari) # Output: [2, 4, 6]
```

Loop and Lambda functions

- Possono essere usate per generare funzioni parametrizzate

```
def generateScanner(site):  
    return lambda page, param,value : site+'/' +page+'?' +param+'='+value  
  
params = {"id", "username", "pwd"}  
values = {"pippo", "pluto", "paperino"}  
pages = {"admin", "login", "wp-admin"}  
while True:  
    site = input("Inserire l'host di cui eseguire lo scan (#-per uscire): ")  
    if site == '#': break  
    scan = generateScanner(site)  
    for page in pages:  
        for param in params:  
            for value in values:  
                print(scan(page,param,value))
```

```
Inserire l'host di cui eseguire lo scan (#-  
per uscire): www.miosito.it  
www.miosito.it/login?username=paperino  
www.miosito.it/login?username=pippo  
www.miosito.it/login?username=pluto  
www.miosito.it/login?id=paperino  
www.miosito.it/login?id=pippo  
...  
www.miosito.it/wp-admin?pwd=pippo  
www.miosito.it/wp-admin?pwd=pluto  
Inserire l'host di cui eseguire lo scan (#-  
per uscire): #
```



Esercizi



Esercizio 1: Controllo se un numero è pari

- Scrivi una funzione `lambda` che prenda un numero e restituisca `True` se è pari, `False` altrimenti.
- Esempi:

python

Copia Modifica

```
fun(4) → True  
fun(7) → False
```

Soluzione:

python

Copia Modifica

```
pari = lambda x: x % 2 == 0
```

```
# Test
```

```
print(pari(4)) # Output: True
```

```
print(pari(7)) # Output: False
```




Esercizi



Esercizio 2: Raddoppiare ogni numero di una lista

- Scrivi una `lambda` per raddoppiare ogni elemento di una lista.
- Usa `map()` per applicarla a tutta la lista.
- Esempi:

python

[Copia](#) [Modifica](#)

```
fun([1,2,3]) → [2, 4, 6]
```

Soluzione:

python

[Copia](#) [Modifica](#)

```
raddoppia = lambda x: x * 2

lista = [1, 2, 3, 4]
risultato = list(map(raddoppia, lista))
print(risultato) # Output: [2, 4, 6, 8]
```



Esercizi

Esercizio 3: Prendere la prima lettera di ogni parola in una lista

- Scrivi una `lambda` che estragga la prima lettera di ogni parola in una lista.
- Usa `map()` per applicarla a tutte le parole.
- Esempi:

python

Copia

Modifica

```
fun(["gatto", "cane", "elefante"]) → ["g", "c", "e"]
```

Soluzione:

python

Copia

Modifica

```
prima_lettera = lambda parola: parola[0]

parole = ["gatto", "cane", "elefante"]
lettere = list(map(prima_lettera, parole))
print(lettere) # Output: ['g', 'c', 'e']
```

Oggetti e classi

- Python è un linguaggio orientato agli oggetti, il che significa che **tutto** in Python è un **oggetto**, incluso numeri, stringhe e liste. Ma possiamo anche creare i nostri oggetti definendo classi.
- Una classe è un **modello per creare oggetti**. Definisce attributi (variabili) e metodi (funzioni) associati all'oggetto.

costruttore

```
class Persona:
    def __init__(self, nome, cognome):
        self.nome = nome
        self.cognome = cognome
    def presentati(self, hello):
        print(hello+' '+self.nome+" "+self.cognome+', come stai?')

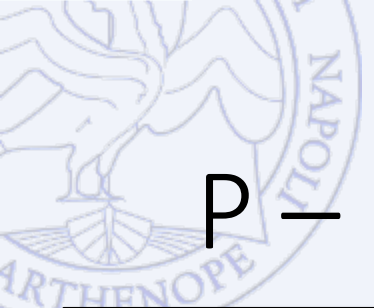
p = Persona('Luigi', 'Coppolino')
p.presentati('Ciao')
p.nome = 'Filippo'
p.presentati('Buongiorno')
```

Gli attributi sono creati nel momento in cui vengono usati

Ogni metodo ha un attributo self che rappresenta l'oggetto su cui è invocato

Ciao Luigi Coppolino, come stai?

Buongiorno Filippo Coppolino, come stai?



P – Parents...ereditarietà

```
class Persona:
    def __init__(self, nome, cognome):
        self.nome = nome
        self.cognome = cognome
    def presentati(self, hello):
        print(hello+' '+self.nome+" "+self.cognome+', come stai?')

class Studente(Persona):
    pass

p = Persona("Luigi", "Coppolino")
s = Studente("Franco", "Franchi")

p.presentati("Buongiorno")
s.presentati("Ciao")
```

pass per dire che il
blocco istruzioni è
vuoto

Uno “Studente” eredita da “Persona” tutti i suoi metodi e attributi

Anche se Studente non ha metodi propri, eredita presentati() da Persona.

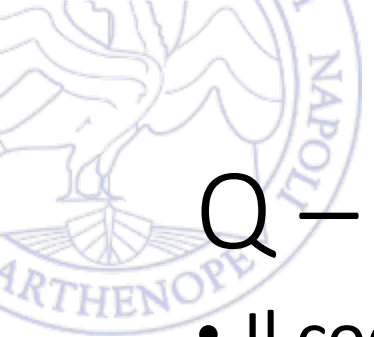
Buongiorno Luigi Coppolino, come stai?
Ciao Franco Franchi, come stai?

```
class Studente(Persona):
    def __init__(self, matricola, nome, cognome):
        self.matricola=matricola
        super().__init__(nome, cognome)
    def presentati(self, hello):
        print(hello+' '+self.nome+" "+self.cognome+', matricola ' + str(self.matricola))

p = Persona("Luigi", "Coppolino")
s = Studente(3274011,"Franco", "Franchi")

p.presentati("Buongiorno")
s.presentati("Ciao")
```

Buongiorno Luigi Coppolino, come stai?
Ciao Franco Franchi, matricola 3274011



Q – Qualità del codice...i Moduli

- Il codice può essere organizzato in moduli

1

Un modulo è semplicemente un file `.py` con funzioni o variabili.

Esempio: Creiamo un file chiamato `matematica.py` con alcune funzioni:

```
python

# matematica.py
def somma(a, b):
    return a + b

def moltiplica(a, b):
    return a * b
```

2

Per usare il modulo in un altro file, possiamo importarlo con `import`.

Esempio (file principale `main.py`):

```
python

import matematica # Importa il modulo

ris1 = matematica.somma(3, 5)
ris2 = matematica.moltiplica(4, 6)

print(ris1) # Output: 8
print(ris2) # Output: 24
```

3

Se vogliamo importare solo una funzione specifica, possiamo usare `from ... import`.

```
python

from matematica import somma

print(somma(10, 20)) # Output: 30
```

♦ **Vantaggio:** Non serve scrivere `matematica.somma()`, basta `somma()` direttamente.

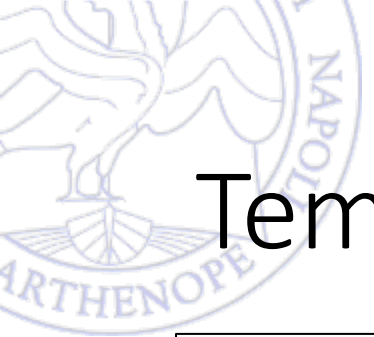
4

Possiamo abbreviare il nome del modulo con `as`:

```
python

import matematica as m

print(m.somma(2, 3)) # Output: 5
```



Tempo

```
import datetime

x = datetime.datetime.now()
print(x)
print(x.year)
print(x.strftime("%A"))
```

2020-03-04 01:08:15.804576
2020
Wednesday

%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17

%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00

Utente, gestire le interazioni

- La funzione `input()` permette di ricevere dati dall'utente come stringa

```
nome = input("Inserisci il tuo nome: ")  
print(f"Ciao, {nome}!")
```

♦ Tutti gli input sono stringhe, quindi se serve un numero, dobbiamo convertirlo.

Se vogliamo inserire un numero, dobbiamo convertire l'input con `int()` o `float()`.

python

Copia Modifica

```
eta = int(input("Inserisci la tua età: "))  
print(f"L'anno prossimo avrai {eta + 1} anni.")
```

⚠ **Attenzione:** Se l'utente inserisce un valore non numerico, il programma darà errore.

Gestione dei file

- **apertura file** `open(nome file, "w/a/x/r")`

La funzione `open(nomefile, modalità)` permette di aprire un file con diverse modalità:

- "w" (write): scrive un nuovo file (se esiste, lo sovrascrive).
- "a" (append): aggiunge contenuto a un file esistente, senza sovrascriverlo.
- "r" (read): legge il contenuto di un file.
- "x" (exclusive creation): crea un file ma genera errore se esiste già.

```
#Apriamo un file e scriviamo
f = open("./prova.txt", 'w')
f.write("ciao \npippo \nbaudo ")
f.close()
-----
#Aggiungiamo ad un file
esistente
f = open("./prova.txt", 'a')
f.write("addio")
f.close()
-----
#Leggiamo il contenuto
f = open("./prova.txt", 'r')
print(f.readline())
for i in f:
    print(i)
f.close()
```

```
ciao
pippo
baudo addio
```




Per esercitarsi...

- <https://www.programmareinpython.it/esercizi-python/>
- https://www.w3schools.com/python/python_exercises.asp
- <https://www.hackerrank.com/domains/python>
- <https://edabit.com/challenges/python3>

Provare codice online:

- <https://www.online-python.com/>



References

- Espressioni Regolari: <https://developers.google.com/edu/python/regular-expressions>
- Esercizi Espressioni Regolari: <https://www.w3resource.com/python-exercises/re/>
- Approfondire python:
<https://www.python.org/>
<https://www.w3schools.com/Python/default.asp>
<https://www.programmareinpython.it/video-corso-python-base/>
<https://www.codecademy.com/learn/learn-python-3>
- Esercizi Python: <https://www.hackerrank.com/domains/python>