

Linguistica Computazionale

Laboratorio in Python - II



Alessio Miaschi

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

alessio.miaschi@ilc.cnr.it

<https://alemmiaschi.github.io/>

<http://www.italianlp.it/alessio-miaschi/>

Strutture di controllo

Strutture di controllo

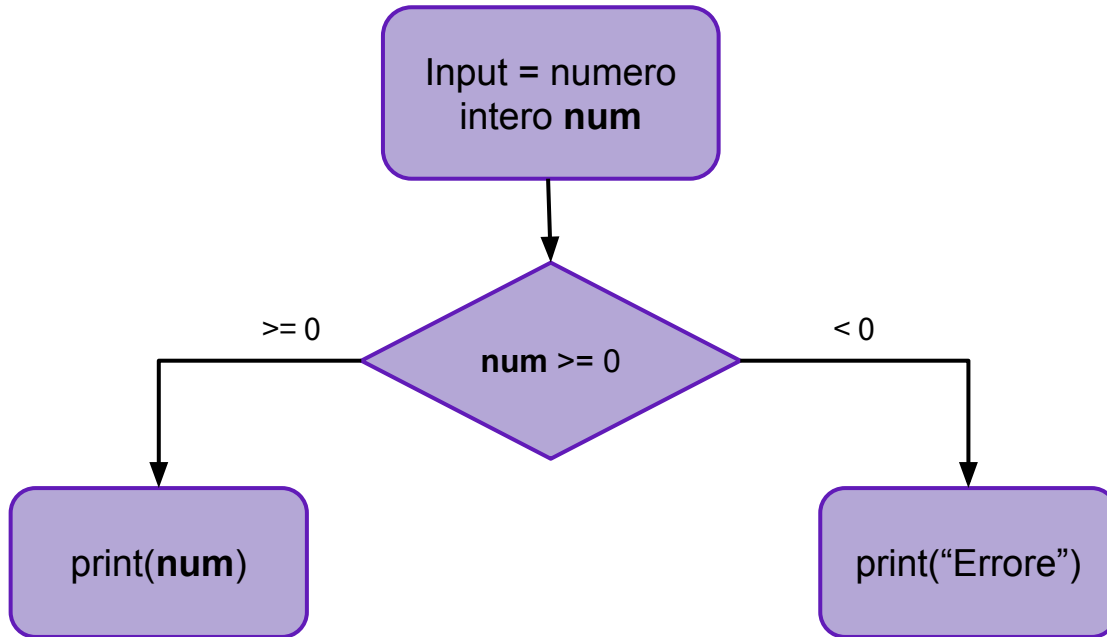
- Ogni programma è strutturato come una sequenza di istruzioni
- Fino ad ora abbiamo visto esecuzioni di tipo **sequenziale**: un comando alla volta
- Come vedremo, ci sono diversi casi in cui è necessario ricorrere a strutture più complesse, alterando l'esecuzione delle nostre istruzioni sulla base di **condizioni**

Strutture di controllo

- In python esistono due principali strutture di controllo per monitorare il flusso del codice
- Strutture condizionali:
 - **if** e **else**
- Strutture di ciclo:
 - **for** e **while**

Condizioni

Condizioni



Condizioni

- In python le condizioni si esprimono attraverso i comandi **if** e **else**:

if condizione:

Istruzioni se la condizione è verificata (**True**)

else:

Istruzioni se la condizione non è verificata (**False**)

Condizioni

- In python le condizioni si esprimono attraverso i comandi **if** e **else**:

if condizione:

Istruzioni se la condizione è verificata (**True**)

else:

Istruzioni se la condizione non è verificata (**False**)

Opzionale!

Condizioni



```
num = 7
if num >= 0:
    print("Il numero è positivo!")
else:
    print("Il numero è negativo!")

# >>> 'Il numero è positivo'
```

Condizioni



```
num = 7
```

```
if num >= 0:  
    print("Il numero è positivo!")  
else:  
    print("Il numero è negativo!")
```

```
# >>> 'Il numero è positivo'
```

Condizioni



```
num = 7
if num >= 0:
    print("Il numero è positivo!")
else:
    print("Il numero è negativo!")

# >>> 'Il numero è positivo'
```

Condizioni



```
num = 7
if num >= 0:
    print("Il numero è positivo!")
else:
    print("Il numero è negativo!")

# >>> 'Il numero è positivo'
```

Condizioni



```
num = -4  
if num >= 0:  
    print("Il numero è positivo!")
```

```
# >>>
```

Condizioni

- Se una condizione risulta essere **non-vuota**, o diversa da zero, python la interpreterà come una condizione **True**



```
num = 10

if num:
    print("E' un numero intero!")
else:
    print("Errore!")

# >>> 'E' un numero intero'
```

Condizioni

- Se una condizione risulta essere **non-vuota**, o diversa da zero, python la interpreterà come una condizione **True**



```
num = 10

if num:
    print("E' un numero intero!")
else:
    print("Errore!")

# >>> 'E' un numero intero'
```



```
stringa = "Ciao"

if stringa:
    print(stringa)
else:
    print("Errore!")

# >>> 'Ciao'
```

Condizioni

- È possibile eseguire blocchi **nidificati**: blocchi all'interno di altri blocchi



```
num = 7
```

```
if num == 0:
```

```
    print("Il numero è uguale a zero")
```

```
else:
```

```
    if num > 0:
```

```
        print("Il numero è maggiore di zero")
```

```
    else:
```

```
        print("Il numero è minore di zero")
```

```
# >>> 'Il numero è maggiore di zero'
```


Condizioni

- Possiamo semplificare la struttura nidificata ricorrendo al comando **elif**

```
num = 7

if num == 0:
    print("Il numero è uguale a zero")
elif num > 0:
    print("Il numero è maggiore di zero")
else:
    print("Il numero è minore di zero")

# >>> 'Il numero è maggiore di zero'
```

Condizioni

- Attenzione all'utilizzo del comando **elif**!

```
num = 10

if isinstance(10, int):
    print("Numero intero")
    if num >= 0:
        print("Numero maggiore o uguale di zero")
    else:
        print("Numero minore di zero")

# >>> 'Numero intero'
# >>> 'Numero maggiore o uguale di zero'
```

```
num = 10

if isinstance(10, int):
    print("Numero intero")
elif num >= 0:
    print("Numero maggiore o uguale di zero")
else:
    print("Numero minore di zero")

# >>> 'Numero intero'
```

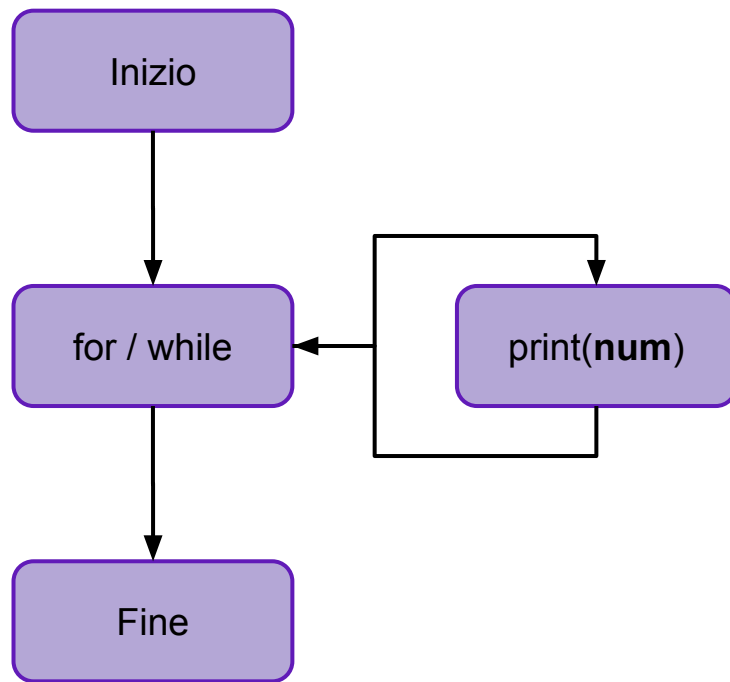
Esercizi

- Chiedere all'utente di inserire due numeri reali **a** e **b** (diversi fra loro) e stampare il maggiore fra i due
- Chiedere all'utente di inserire una stringa e verificare se tale stringa inizia con una vocale
- Data la **lista** di numeri interi [5, 6, 8, 10, 12] e un numero intero **a** (da chiedere in input), verificare se il primo elemento della lista corrisponde ad **a**. In caso negativo, stampare il primo elemento della lista

Cicli for e while

Strutture di ciclo

- I cicli permettono di eseguire blocchi di codice in maniera iterativa
- In python esistono due tipi di cicli (*loop*):
 - **for**: esegue un'iterazione per ogni elemento di un **iterabile**
 - **while**: esegue un'iterazione fintanto che una condizione è vera



Il ciclo for

- Il ciclo for permette di iterare su tutti gli elementi di un **iterabile** ed eseguire un blocco di codice
- Un iterabile è un qualsiasi oggetto python in grado di restituire i suoi elementi uno dopo l'altro:
 - liste, tuple, dizionari, ecc.

Il ciclo for

- Il ciclo for permette di iterare su tutti gli elementi di un **iterabile** ed eseguire un blocco di codice
- Un iterabile è un qualsiasi oggetto python in grado di restituire i suoi elementi uno dopo l'altro:
 - liste, tuple, dizionari, ecc.

for elemento **in** iterabile:
Esegui delle istruzioni

Il ciclo for



```
lista_citta = ["Roma", "Napoli", "Pisa", "Milano"]

for citta in lista_citta:
    print(citta)

# >>> Roma
# >>> Napoli
# >>> Pisa
# >>> Milano
```


Il ciclo for

```
● ● ●  
  
lista_citta = ["Roma", "Napoli", "Pisa", "Milano"]  
  
for citta in lista_citta:  
    print(citta)  
  
# >>> Roma  
# >>> Napoli  
# >>> Pisa  
# >>> Milano
```

Iterabile

Il ciclo for

Indice

```
lista_citta = ["Roma", "Napoli", "Pisa", "Milano"]  
for citta in lista_citta:  
    print(citta)  
  
# >>> Roma  
# >>> Napoli  
# >>> Pisa  
# >>> Milano
```

Iterabile

Il ciclo for

Indice

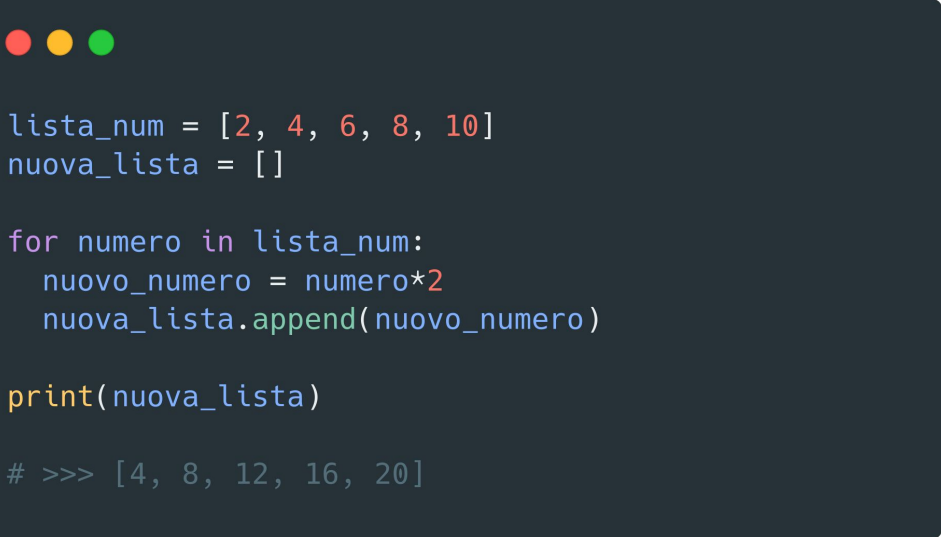
```
lista_citta = ["Roma", "Napoli", "Pisa", "Milano"]  
for citta in lista_citta:  
    print(citta)  
  
# >>> Roma  
# >>> Napoli  
# >>> Pisa  
# >>> Milano
```

Blocco (da
eseguire n
volte)

Iterabile

Il ciclo for

- Iterando ogni elemento di un dato iterabile posso eseguire determinate operazioni (una o più per ogni elemento dell'iterabile)



```
lista_num = [2, 4, 6, 8, 10]
nuova_lista = []

for numero in lista_num:
    nuovo_numero = numero*2
    nuova_lista.append(nuovo_numero)

print(nuova_lista)

# >>> [4, 8, 12, 16, 20]
```

Il ciclo for - la funzione *range()*

- Per eseguire una determinata operazione un numero fisso n di volte posso ricorrere alla funzione *range()*
- *range()* imposta un intervallo di numeri tanto ampio quanto il numero che le passiamo come parametro
- Esempio: *range(10)* == [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Il ciclo for - la funzione *range()*



```
for indice in range(5):  
    print(indice, "Ciao mondo!")
```

```
# >>> 0, 'Ciao mondo!'  
# >>> 1, 'Ciao mondo!'  
# >>> 2, 'Ciao mondo!'  
# >>> 3, 'Ciao mondo!'  
# >>> 4, 'Ciao mondo!'
```

Il ciclo for - iterare su dizionari

- È possibile iterare le coppie *<chiave, valore>* di un dizionario, ricorrendo al metodo *.items()*

```

diz = {"Italia": ['Roma', 'Milano', 'Pisa'],
       "Francia": ['Parigi', 'Lione']}

for k, v in diz.items():
    print("chiave:", k, ", valore: ", v)

# >>> chiave: Italia, valore: ['Roma', 'Milano', 'Pisa']
# >>> chiave: Francia, valore: ['Parigi', 'Lione']
```

Il ciclo for - iterare su dizionari

- È possibile iterare le coppie *<chiave, valore>* di un dizionario, ricorrendo al metodo *.items()*

```
diz = {"Italia": ['Roma', 'Milano', 'Pisa'],
      "Francia": ['Parigi', 'Lione']}

for k, v in diz.items():
    print("chiave:", k, ", valore: ", v)

# >>> chiave: Italia, valore: ['Roma', 'Milano', 'Pisa']
# >>> chiave: Francia, valore: ['Parigi', 'Lione']
```

Doppio indice!

Il ciclo for

- Come per i comandi condizionali, è possibile eseguire blocchi **nidificati**:

```
diz = {"Italia": ["Roma", "Milano", "Pisa"]  
      "Francia": ["Parigi", "Lione"]}
```

```
for k, v in diz.items():  
    for i in v:  
        print(k, i)
```

```
# >>> 'Italia', 'Roma'  
# >>> 'Italia', 'Milano'  
# >>> 'Italia', 'Pisa'  
# >>> 'Francia', 'Parigi'  
# >>> 'Francia', 'Lione'
```

List comprehensions

- La *list comprehension* è un tecnica che ci permette di definire e popolare liste in maniera sintetica


Metodo tradizionale:

```
lista = []  
  
for num in range(5):  
    lista.append(num*2)  
  
print(lista)  
  
# >>> [0, 2, 4, 6, 8]
```

List comprehensions

- La *list comprehension* è un tecnica che ci permette di definire e popolare liste in maniera sintetica

List comprehension:



```
lista = [num*2 for num in range(5)]  
  
print(lista)  
  
# >>> [0, 2, 4, 6, 8]
```

Il ciclo while

- Il ciclo while itera, eseguendo un determinato blocco di codice, finché una condizione è vera (*True*)

while condizione:
Esegui delle istruzioni

Il ciclo while



```
i = 0
```

```
while i < 5:  
    print("Ciao mondo!!")  
    i+=1
```

```
# >>> Ciao mondo!!
```

```
# >>> Ciao mondo!!
```

```
# >>> Ciao mondo!!
```

```
# >>> Ciao mondo!!
```

```
# >>> Ciao mondo!!
```

Il ciclo while

Contatore



```
i = 0
```

```
while i < 5:  
    print("Ciao mondo!!")  
    i+=1
```

```
# >>> Ciao mondo!!  
# >>> Ciao mondo!!  
# >>> Ciao mondo!!  
# >>> Ciao mondo!!  
# >>> Ciao mondo!!
```

Il ciclo while

Contatore

```
i = 0
```

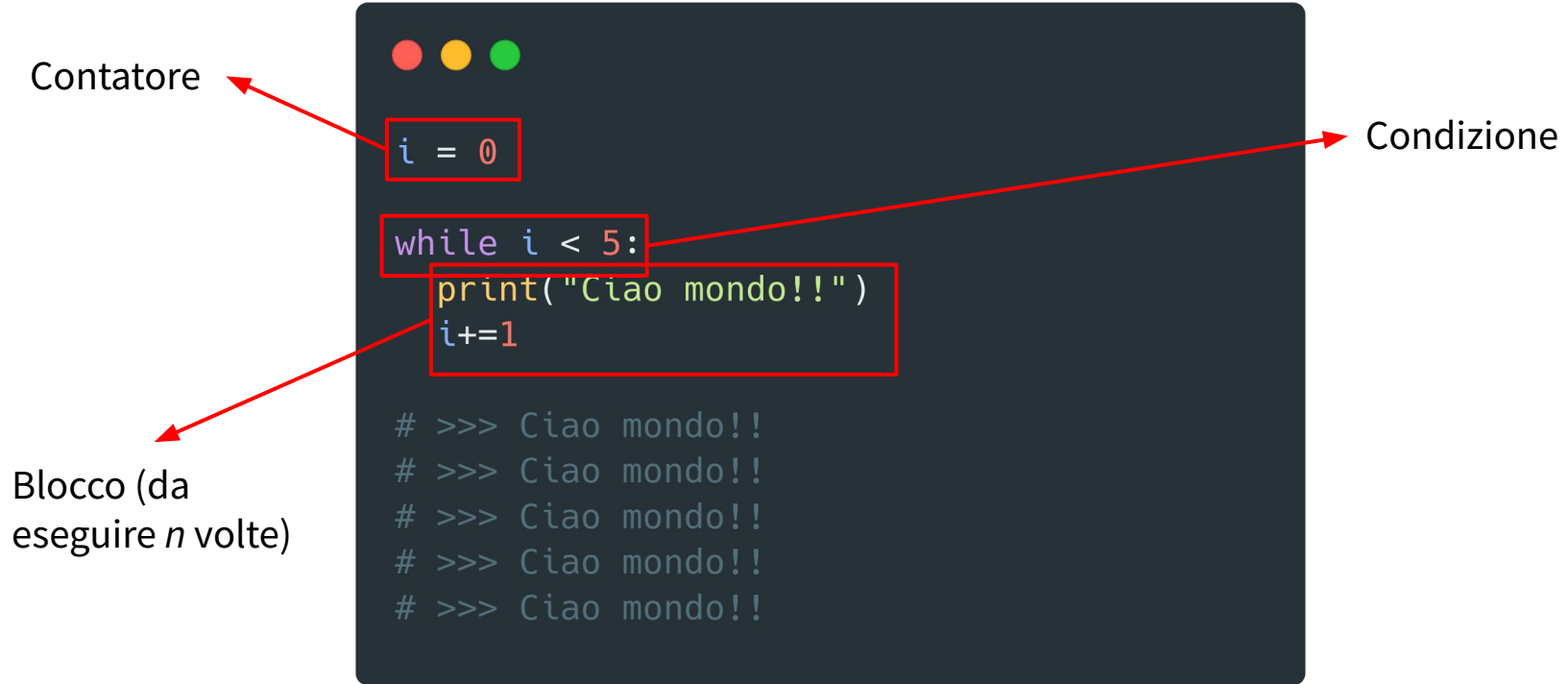
```
while i < 5:
```

```
    print("Ciao mondo!!")  
    i+=1
```

Condizione

```
# >>> Ciao mondo!!  
# >>> Ciao mondo!!  
# >>> Ciao mondo!!  
# >>> Ciao mondo!!  
# >>> Ciao mondo!!
```

Il ciclo while



Il ciclo `while`

- Cosa succederebbe se la condizione di controllo restasse sempre *True*?

Il ciclo while

- Cosa succederebbe se la condizione di controllo restasse sempre *True*?

Infinite loop!



```
num = 10
```

```
while num == 10:  
    print("Il numero è: ", num)
```

```
# >>> Il numero è: 10
```

```
# >>> Il numero è: 10
```

```
# >>> Il numero è: 10
```

```
# >>> Il numero è: 10
```

```
# >>> (...)
```

Break e continue

- Esistono due istruzioni che permettono di alterare la normale esecuzione di un ciclo for/while:
 - **break**;
 - **continue**.
- **break** serve per terminare un ciclo for/while prematuramente
- **continue**, invece, non interrompe l'esecuzione del ciclo, ma fa saltare l'esecuzione del codice dopo l'istruzione e fa ripartire python dalla prima riga del ciclo

Il ciclo while - break



```
controllo = True
contatore = 0

while controllo:
    contatore+=1
    print(contatore)
    if contatore == 5:
        print("Fine del loop!")
        break
```

```
# >>> 1
# >>> 2
# >>> 3
# >>> 4
# >>> 5
# >>> Fine del loop!
```

Il ciclo while - continue



```
contatore = 0

while contatore < 5:
    contatore+=1
    if contatore == 2:
        print("Interruzione del loop!")
        continue
    print(contatore)

# >>> 1
# >>> Interruzione del loop
# >>> 3
# >>> 4
# >>> 5
```

Esercizi

- Scrivere un programma che, data in input una lista di interi, restituisca il totale di numeri pari e dispari presenti al suo interno
- Scrivere un programma che data in input una lista **a** contenente n parole, restituisca in output un'altra lista **b** di interi che rappresentano la lunghezza delle parole contenute in **a**
- Scrivere un programma che, data in input una frase, crei e stampi una nuova frase contenente underscore (“_”) al posto degli spazi (es. “Questo è un esercizio” in “Questo_è_un_esercizio”)

Esercizi

- Scrivere un programma che prende in input una stringa e che stampa la versione inversa della stessa stringa (es, “ciao” diventerà “oaic”)
- Scrivere un programma che, data in input una stringa, riconosce se si tratta di un palindromo
- Scrivere un programma che, data in input una lista di interi, stampi i numeri divisibili per 5. Se scorrendo la lista si trova un numero maggiore o uguale di 150, interrompere il ciclo

Esercizi

- Scrivere un programma che prende in input un dizionario con la seguente struttura: {"Nome dello studente": "lista dei voti", ecc} (es. {"Alessio": [21, 28, 30, 18], "Laura": [21, 20, 30, 30, 28, 25]}). Per ogni studente:
 - verificare se nel suo libretto sono presenti almeno 5 valutazioni
 - In caso positivo, stampare il nome dello studente e la media delle sue valutazioni
 - In caso negativo, stampare "Lo studente *X* non ha raggiunto il numero minimo di esami. Numeri di esami mancanti: *n*", dove *n* corrisponde al numero di esami mancanti per arrivare a 5
- Scrivere un programma che prende in input due liste di interi **a** e **b** e restituisce una nuova lista **c** contenente le somme dei valori delle due liste. Es. se $a = [1, 2, 3, 4, 5]$ e $b = [5, 6, 7, 8, 9]$, allora $c = [6, 8, 10, 12, 14]$. Se le due liste hanno lunghezza diversa, il programma deve stampare il messaggio di errore "Errore! Le due liste hanno lunghezza diversa!" In ogni caso, l'operazione si deve interrompere se la nuova lista **c** raggiunge una lunghezza maggiore di 5 elementi

Le funzioni

Le funzioni

- Le funzioni sono uno strumento che permette di raggruppare porzioni di istruzioni che eseguono un compito specifico
- Lo scopo principale è quello di rendere il codice più ordinato ed evitare inutili ripetizioni
- Le funzioni prendono come input zero o più argomenti, eseguono una serie di istruzioni e restituiscono un risultato

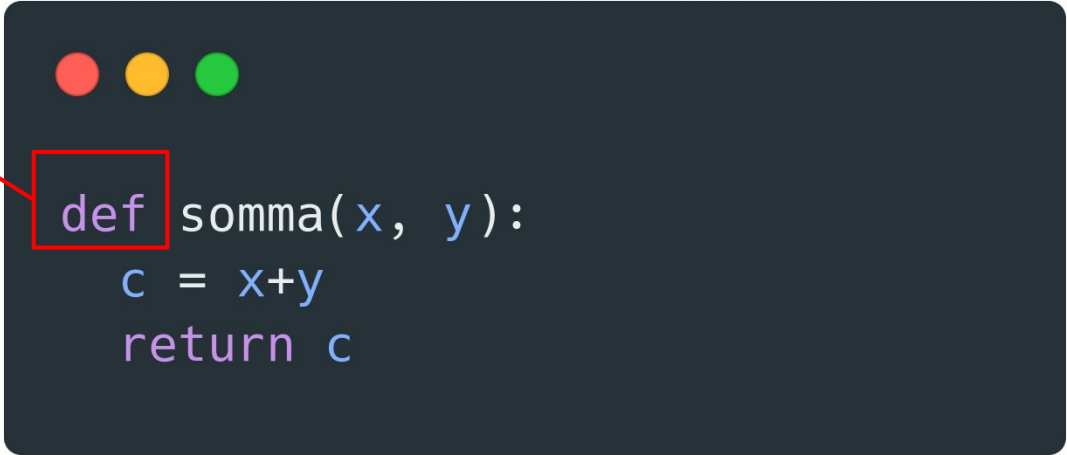
Le funzioni



```
def somma(x, y):  
    c = x+y  
    return c
```

Le funzioni

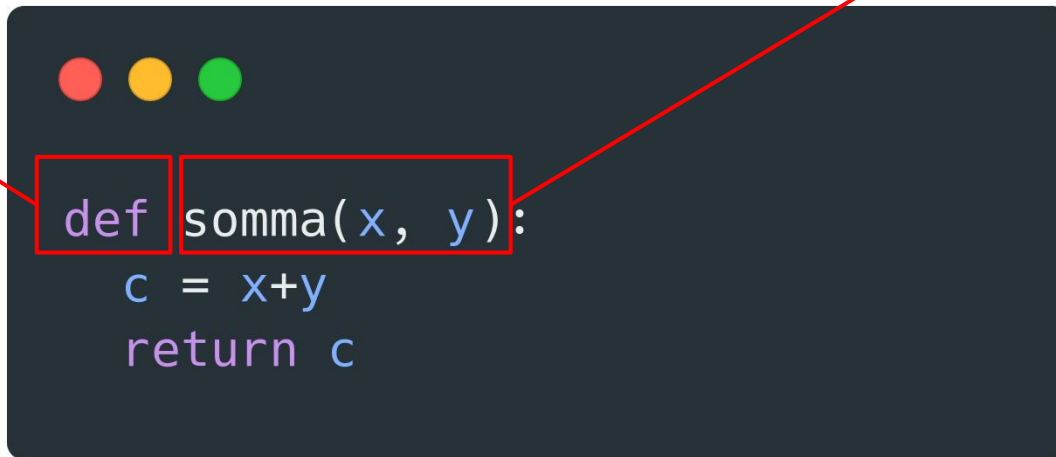
Parola riservata



```
def somma(x, y):  
    c = x+y  
    return c
```

Le funzioni

Parola riservata



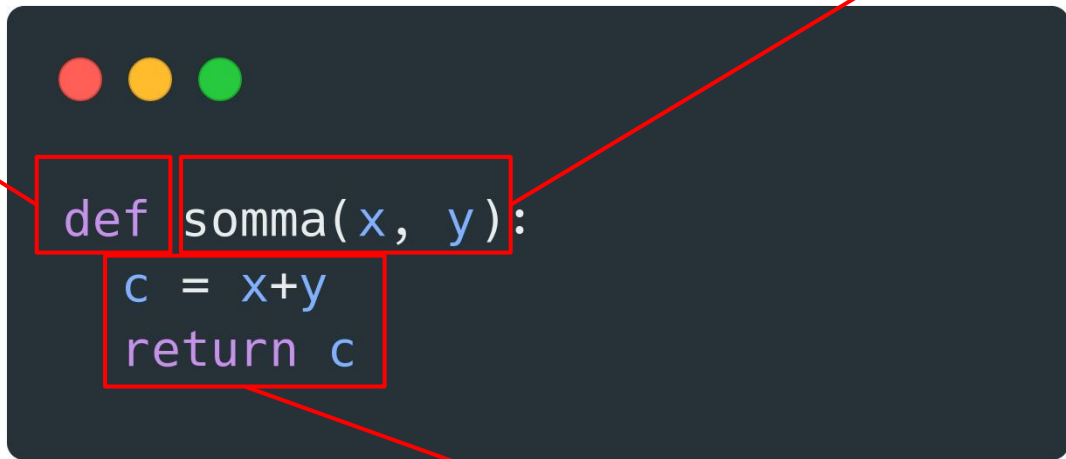
```
def somma(x, y):  
    c = x+y  
    return c
```

Nome della funzione e
parametri

Le funzioni

Parola riservata

Nome della funzione e
parametri



```
def somma(x, y):  
    c = x+y  
    return c
```

Istruzioni

Le funzioni



```
def somma(x, y):  
    c = x+y  
    return c
```

```
a = 5  
b = 7  
c = somma(a, b)  
print(c)
```

```
# >>> 12
```

Le funzioni

Funzione

```
def somma(x, y):  
    c = x+y  
    return c
```

Chiamata

```
a = 5  
b = 7  
c = somma(a, b)  
print(c)
```

```
# >>> 12
```


Le funzioni



```
def somma(x, y):  
    c = x+y  
    return c
```


```
a = 5  
b = 7  
c = somma(a, b)  
print(c)  
d = somma(c, a)  
print(d)
```

```
# >>> 12
```

```
# >>> 17
```

Le funzioni


- È possibile dichiarare funzioni che non restituiscono risultati → senza **return**



```
def stampa(stringa):  
    print(stringa)  
  
stringa = "Ciao mondo"  
stampa(stringa)  
  
# >>> 'Ciao mondo!'
```

Le funzioni

- È possibile dichiarare funzioni senza argomenti



```
def stampa_numeri():  
    lista = [1, 2, 3]  
    for numero in lista:  
        print(numero)
```


```
stampa_numeri()
```

```
# >>> 1
```

```
# >>> 2
```

```
# >>> 3
```

Le funzioni



```
def stampa_lista(lista):  
    for numero in lista:  
        print(numero)  
  
def main():  
    # Stampa la stringa "Hello world"  
    print ("Hello world")  
  
    # Crea due variabili numeriche e stampa la loro somma  
    a = 5  
    b = 6  
    print(a+b)  
  
    # Crea una lista e la passa come argomento ad una funzione  
    lista = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]  
    stampa_lista(lista)  
  
main()
```