Passaggio parametri, *args, **kwargs

Advanced Computer Programming

Prof. Luigi De Simone

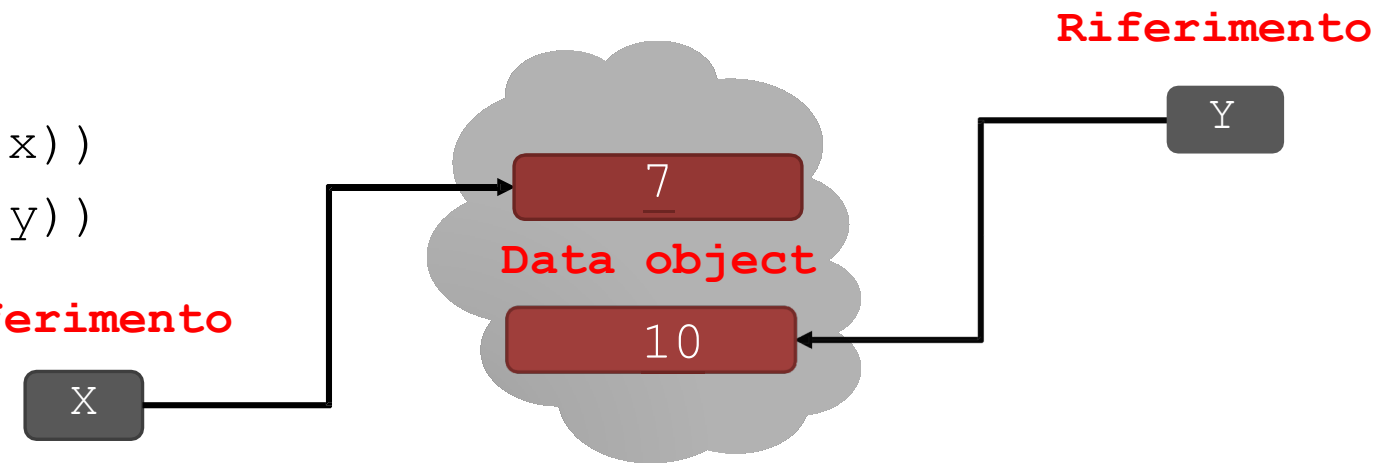
Riferimenti

- **Tutto in Python è un oggetto**
- Possiamo interagire con gli oggetti attraverso dei *riferimenti*
  - Quando dichiariamo una variabile, la variabile non conterrà effettivamente il valore, ma piuttosto la sua posizione in memoria
  - E' possibile utilizzare la funzione ***id(…)*** per conoscere il riferimento
- Ad esempio:

```
x = 7
y = 10
print("x=", x, "ref_x=", id(x))
print("y=", y, "ref_y=", id(y))
```

Output:

```
x= 7 ref_x= 4308052464
y= 10 ref_y= 4308052560
```



Passaggio di parametri

- Tutti i parametri in Python sono **passati per riferimento**
- Quando passiamo un parametro ad una funzione, il riferimento contenuto nel parametro attuale è copiato all'interno del parametro formale
- Utilizzando tale riferimento, la funzione avrà accesso direttamente all'oggetto in memoria
- Ad esempio:

```
def fun(x):
    print("Inside fun: x=", x, "ref_x=", id(x))

x = 7
fun(x)
print("Outside fun: x=", x, "ref_x=", id(x))
```

Output:

```
Inside fun: x= 7 ref_x= 4308052464
Outside fun: x= 7 ref_x= 4308052464
```

Passaggio di parametri: oggetti mutabili

- Nel caso in cui il parametro sia un **oggetto mutabile** (ad esempio una lista), ogni modifica al parametro all'interno della funzione, si riflette anche sul chiamante
- Ad esempio:

```
def fun(mylist):
    mylist.append([1,2,3,4])
    print ("Inside fun: val=", mylist, "ref=", id(mylist))


mylist = [10,20,30]
fun( mylist )
print ("Outside fun: val=", mylist, "ref=", id(mylist))
```

Output:

```
Inside fun: val= [10, 20, 30, [1, 2, 3, 4]] ref= 4353638528
Outside fun: val= [10, 20, 30, [1, 2, 3, 4]] ref= 4353638528
```

Passaggio di parametri: oggetti mutabili

- Da notare che, nel caso in cui il riferimento venga sovrascritto all'interno della funzione, il parametro diventa locale alla funzione.
- Pertanto, ogni modifica al parametro all'interno della funzione, non si riflette sul chiamante
- Ad esempio:

```
def fun(mylist):
    mylist = [1,2,3,4]
    print ("Inside fun: val=", mylist, "ref=", id(mylist))


mylist = [10,20,30]
fun( mylist )
print ("Outside fun: val=", mylist, "ref=", id(mylist))
```

Output:

```
Inside fun: val= [1, 2, 3, 4] ref= 4353638464
Outside fun: val= [10, 20, 30] ref= 4352610752
```

Passaggio di parametri: oggetti immutabili

- Nel caso in cui il parametro sia un **oggetto immutabile**, anche se la funzione dispone del riferimento all'oggetto, non gli sarà possibile apportare modifiche
- Viene infatti creato un nuovo oggetto in memoria
  Pertanto, ogni operazione sul parametro non avrà effetto sul chiamante

```
def fun(x):
    print ("Inside fun before: val=", x, "ref=", id(x))
    x**=2
    print ("Inside fun after: val=", x, "ref=", id(x))

x = 3
fun( x )
print ("Outside fun: val=", x, "ref=", id(x))
```

Output:

```
Inside fun before: val= 3 ref= 4308052336
Inside fun after: val= 9 ref= 4308052528
Outside fun: val= 3 ref= 4308052336
```

Argomenti a lunghezza variabile: *args

- La sintassi ***args*** può essere utilizzata nella definizione delle funzioni Python per passare alla funzione un numero variabile di argomenti
- La funzione riceverà una tupla di argomenti (*args* nell'esempio), la quale può essere iterata per recuperare tutti gli argomenti

```
def fun(*args):
    for arg in args:
        print(arg)

fun("first", "second", "third")
```

Output:

```
first
second
third
```

```
def fun(x, *args):
    print(x)
    for arg in args:
        print(arg)

fun(100, "first", "second", "third")
```

Output:

```
100
first
second
third
```

Keyword arguments

- I **keyword arguments** permettono al chiamante di una funzione di passare alla funzione gli argomenti basandosi sul nome, e non sulla posizione
- La sintassi da utilizzare è del tipo he `key = value`

```
def fun(name, surname, country):
    print(name, surname, country)

fun(surname="Rossi", country="Italy", name="Mario")
```

Output:

```
Mario Rossi Italy
```

Keyword arguments: **kwargs

- La sintassi *****kwargs*** può essere utilizzata nella definizione delle funzioni Python per passare alla funzione un numero variabile di keyword arguments
- La funzione riceverà un dizionario di argomenti (*kwargs* nell'esempio), il quale può essere iterato per recuperare tutti gli argomenti

```
def fun(**kwargs):
    for key, value in kwargs.items():
            print(key, "=", value)

fun(day=8, month="March", year=2024)
```

Output:

```
day = 8
month = March
year = 2024
```

```
def fun(x, **kwargs):
    print(x)
    for key, value in kwargs.items():
            print(key, "=", value)

fun("Date:", day=8, month="March", year=2024)
```

Output:

```
Date:
day = 8
month = March
year = 2024
```

Argomenti di default

- Python permette di definire dei valori di **default** per gli argomenti di una funzione
- I valori di default di un argomento sono utilizzati quando nella chiamata alla funzione non è specificato un valore per quell'argomento
- Questo permette di invocare una funzione con meno parametri rispetto alla quantità presente nella sua definizione

```
def fun(username, password="changeme", days=30):
    print(username, password, days)

fun("admin", "admin", 120)
fun("user", "password")
fun("operator")
```

Output:

```
admin admin 120
user password 30
operator changeme 30
```

Argomenti di default

- Gli argomenti con parametri di default devono essere specificati dopo quelli obbligatori

```
def fun(username, password="changeme", days=30, email):
    print(username, password, days, email)

fun("admin", "admin", 120, "admin@email.com")
```

Output:

```
SyntaxError: non-default argument follows default argument
```

```
def fun(username, email, password="changeme", days=30):
    print(username, password, days, email)

fun("admin", "admin", 120, "admin@email.com")
```

Output:

```
admin admin 120 admin@email.com
```

Unpacking argument lists

- La sintassi ***args** utilizzata per i *variable-length arguments* può essere anche utilizzata per operazioni di *unpacking degli argomenti*:
  - Gli argomenti sono già inseriti in una lista o tupla ma devono essere utilizzati in una funzione che richiede argomenti utilizzati in maniera separata

```
>>> list(range(3, 6))    #normal call with separate arguments
[3, 4, 5]
>>> args = [3, 6]
>>> list(range(*args))   #call with arguments unpacked from a list
[3, 4, 5]
```