Funzioni

Cos'è una funzione?

Una funzione è un **blocco di codice riutilizzabile** che fa un lavoro specifico. Pensala come una **ricetta**: la scrivi una volta e poi la puoi riutilizzare quando vuoi senza riscrivere tutto da capo.

Hai già usato funzioni predefinite senza saperlo: `print()`, `input()`, `len()`, `range()` ... Ora imparerai a creare le **tue**! È tipo passare da consumatore a produttore.

Perché usare le funzioni?

- **Riutilizzo**: scrivi il codice una volta, usalo quante volte vuoi (zero copia-incolla!)
- **Organizzazione**: il programma diventa leggibile e ordinato
- **Manutenzione**: se devi correggere qualcosa, lo fai in UN solo punto

---

Definire una funzione

Si usa la parola chiave `def` (abbreviazione di "define"). Facile, no?

```
# Definizione della funzione
def saluta():
    print("Ciao!")
    print("Benvenuto nel programma")

# Chiamata — qui la funzione viene effettivamente eseguita
saluta()
saluta()  # Puoi chiamarla quante volte vuoi! Gratis!
```

!!! warning "Prima definisci, poi chiami!"

```
La funzione deve essere **definita prima** di essere chiamata. Python le
```

Funzioni con parametri

I parametri sono tipo gli **ingredienti** della ricetta: passi i dati alla funzione e lei fa il suo lavoro con quelli:

```
def saluta(nome):
    print(f"Ciao {nome}!")

saluta("Mario")
saluta("Luigi")
saluta("Peach")
```

Più parametri

```
def presenta(nome, eta, citta):
    print(f"Mi chiamo {nome}, ho {eta} anni e vivo a {citta}")

presenta("Mario", 16, "Roma")
presenta("Luigi", 17, "Milano")
```

Parametri con valore predefinito

Puoi dare un valore di default ai parametri. Se chi chiama la funzione non specifica quel parametro, viene usato il default. Comodo!

```
def saluta(nome, lingua="italiano"):
    if lingua == "italiano":
        print(f"Ciao {nome}!")
    elif lingua == "inglese":
        print(f"Hello {nome}!")
    elif lingua == "spagnolo":
        print(f"Hola {nome}!")

saluta("Mario")                    # Usa il default: italiano
saluta("Mario", "inglese")         # Specifica la lingua
saluta("Mario", lingua="spagnolo") # Parametro nominato (più leggibile!)
```

Il valore di ritorno (`return`)

Una funzione può **restituire un valore** con `return`. È la differenza tra un cameriere che ti DICE cosa c'è nel menu (print) e uno che ti PORTA il piatto (return):

```
def somma(a, b):
    risultato = a + b
    return risultato

# Il valore restituito lo puoi salvare in una variabile
s = somma(5, 3)
print(f"5 + 3 = {s}")

# Oppure usarlo direttamente
print(f"10 + 20 = {somma(10, 20)}")
```

Restituire più valori

Python è furbo: puoi restituire più valori in un colpo solo!

```
def analizza_numeri(lista):
    minimo = min(lista)
    massimo = max(lista)
    media = sum(lista) / len(lista)
    return minimo, massimo, media

numeri = [4, 8, 2, 9, 1, 7]
mi, ma, me = analizza_numeri(numeri)
print(f"Min: {mi}, Max: {ma}, Media: {me:.2f}")
```

!!! tip "return vs print — la confusione più classica!"

```
- `print()` **mostra** un valore sullo schermo, ma non lo "salva" — è ti
- `return` **restituisce** un valore che il programma può usare — è tipo

```python
def area_print(base, altezza):
    print(base * altezza)     # Mostra, ma non puoi riutilizzare il valo

def area_return(base, altezza):
    return base * altezza     # Restituisce, puoi salvarlo e usarlo!
```
```

Scope delle variabili

Le variabili create dentro una funzione **esistono solo lì dentro** (variabili locali). Appena la funzione finisce, puf, spariscono! È tipo Las Vegas: quello che succede nella funzione, resta nella funzione.

```
def mia_funzione():
    x = 10  # Variabile locale — esiste solo qui dentro
    print(f"Dentro la funzione: x = {x}")

mia_funzione()

# print(x)  # ERRORE! x non esiste fuori dalla funzione

# Le variabili esterne si possono LEGGERE ma non MODIFICARE
y = 100

def leggi_y():
    print(f"y vale: {y}")  # Può leggere y, ma non cambiarla

leggi_y()
```

Funzioni come strumenti

Esempio pratico: creiamo un kit di funzioni per la geometria. Tipo una calcolatrice scientifica personalizzata:

```
def area_rettangolo(base, altezza):
    return base * altezza

def area_triangolo(base, altezza):
    return base * altezza / 2

def area_cerchio(raggio):
    return 3.14159 * raggio ** 2

def perimetro_rettangolo(base, altezza):
    return 2 * (base + altezza)

# Uso delle funzioni
print("=== Calcolatrice Geometrica ===")
print(f"Rettangolo 5x3: area={area_rettangolo(5,3)}, perimetro={perimetr
print(f"Triangolo base=6, h=4: area={area_triangolo(6,4)}")
print(f"Cerchio r=5: area={area_cerchio(5):.2f}")
```

Funzioni ricorsive

Una funzione che **chiama se stessa**. Sì, hai letto bene. È tipo l'Inception delle funzioni: una funzione dentro una funzione dentro una funzione... Ma con un punto di stop, altrimenti esplode tutto!

```
def fattoriale(n):
    if n <= 1:       # Caso base: FERMATI qui!
        return 1
    else:
        return n * fattoriale(n - 1)  # Chiama se stessa con n-1

print(f"5! = {fattoriale(5)}")
print(f"10! = {fattoriale(10)}")
```

Come funziona?

Immagina una pila di scatole, una dentro l'altra:

```
fattoriale(5)
= 5 * fattoriale(4)
= 5 * 4 * fattoriale(3)
= 5 * 4 * 3 * fattoriale(2)
= 5 * 4 * 3 * 2 * fattoriale(1)
= 5 * 4 * 3 * 2 * 1
= 120
```

!!! danger "Senza caso base = disastro!"

```
Se dimentichi il caso base (`if n <= 1`), la funzione si chiama all'infi
```

Funzioni lambda

Per funzioni semplici di una sola riga, Python offre le **lambda** — le funzioni usa e getta. Sono tipo i post-it delle funzioni: rapide e per cose brevi.

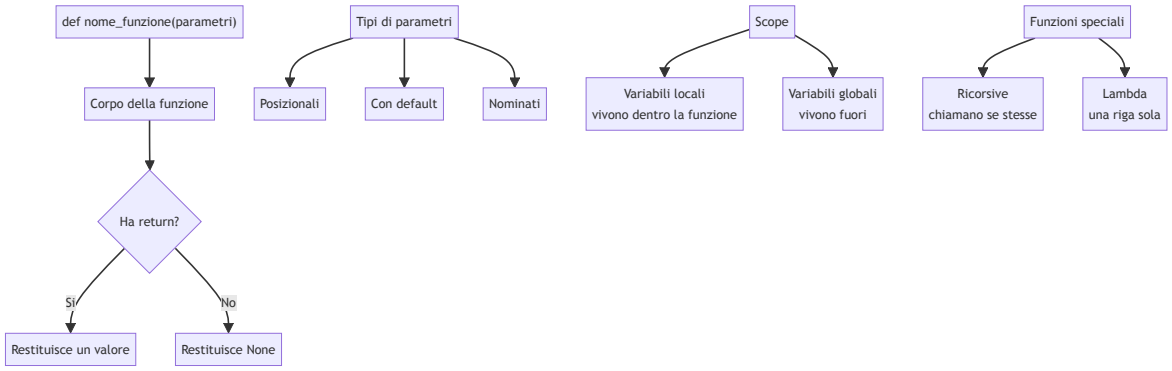
```
# Funzione normale
def quadrato(x):
    return x ** 2

# Equivalente con lambda (una riga!)
quadrato_lambda = lambda x: x ** 2

print(quadrato(5))
print(quadrato_lambda(5))

# Dove brillano: con sort, map, filter
numeri = [3, 1, 4, 1, 5, 9]
numeri.sort(key=lambda x: -x)  # Ordina in modo decrescente
print("Ordinati (desc):", numeri)
```

Mappa concettuale



Esercizi

Esercizio 1: Funzione massimo

Scrivi una funzione `massimo(a, b, c)` che restituisce il massimo tra tre numeri, **senza usare** la funzione `max()`. Devi farlo con gli if!

```
def massimo(a, b, c):
    # Scrivi il codice qui
    pass

# Test
print(massimo(3, 7, 5))   # Deve stampare 7
print(massimo(10, 2, 8))  # Deve stampare 10
```

??? success "Soluzione"

```
```pyodide
def massimo(a, b, c):
    if a >= b and a >= c:
        return a
    elif b >= a and b >= c:
        return b
    else:
        return c

print(massimo(3, 7, 5))
print(massimo(10, 2, 8))
```
```

Esercizio 2: Funzione palindroma

Scrivi una funzione che controlla se una parola è un **palindromo** (si legge uguale da sinistra a destra e viceversa). "anna", "radar", "osso"... ci siamo capiti!

```
def is_palindroma(parola):
    # Scrivi il codice qui
    pass

# Test
print(is_palindroma("anna"))    # True
print(is_palindroma("ciao"))    # False
print(is_palindroma("radar"))   # True
```

??? success "Soluzione"

```pyodide
def is_palindroma(parola):
    return parola == parola[::-1]

print(is_palindroma("anna"))
print(is_palindroma("ciao"))
print(is_palindroma("radar"))
```

Esercizio 3: Fibonacci

Scrivi una funzione che restituisce l'n-esimo numero della sequenza di Fibonacci (1, 1, 2, 3, 5, 8, 13...). Ogni numero è la somma dei due precedenti!

```
def fibonacci(n):
    # Scrivi il codice qui
    pass

# Test: stampa i primi 10 numeri di Fibonacci
for i in range(1, 11):
    print(fibonacci(i), end=" ")
```

??? success "Soluzione"

```pyodide
def fibonacci(n):
    if n <= 2:
        return 1
    a, b = 1, 1
    for _ in range(n - 2):
        a, b = b, a + b
    return b

for i in range(1, 11):
    print(fibonacci(i), end=" ")
```