Cicli

Ripetere le operazioni

Immagina di dover stampare "Ciao" 1000 volte. Scriveresti 1000 righe di `print("Ciao")`? Ovviamente no! Ecco perché esistono i **cicli** (o loop): ti permettono di **ripetere** un blocco di codice quante volte vuoi senza impazzire.

Python ha due tipi di ciclo:

- `for` — quando sai **quante volte** ripetere ("fallo 10 volte")
- `while` — quando ripeti **finché** una condizione è vera ("continua finché non trovi la risposta")

Il ciclo `for`

Ripetere N volte con `range()`

`range()` genera una sequenza di numeri. È tipo un distributore automatico di numeri:

```
# Ripete 5 volte (da 0 a 4)
for i in range(5):
    print(f"Ripetizione numero {i}")
```

Le varianti di `range()`

Sintassi	Genera	Esempio
`range(n)`	Da 0 a n-1	`range(5)` → 0,1,2,3,4
`range(inizio, fine)`	Da inizio a fine-1	`range(2, 6)` → 2,3,4,5
`range(inizio, fine, passo)`	Con salti	`range(0, 10, 2)` → 0,2,4,6,8

```
# Da 1 a 10
print("Da 1 a 10:")
for i in range(1, 11):
    print(i, end=" ")

print()  # A capo

# Numeri pari da 0 a 20
print("
Numeri pari da 0 a 20:")
for i in range(0, 21, 2):
    print(i, end=" ")

print()

# Conto alla rovescia! 🚀
print("
Conto alla rovescia:")
for i in range(10, 0, -1):
    print(i, end=" ")
print("
Via!")
```

Scorrere i caratteri di una stringa

Il `for` è anche perfetto per analizzare una stringa carattere per carattere:

```
parola = "Python"

for carattere in parola:
    print(carattere)
```

Il ciclo `while`

Il `while` ripete **finché la condizione è vera**. È tipo "continua a mangiare finché hai fame":

```
contatore = 1

while contatore <= 5:
    print(f"Contatore: {contatore}")
    contatore += 1

print("Fine!")
```

!!! danger "ATTENZIONE ai cicli infiniti! ∞"

````
Se la condizione del `while` non diventa MAI falsa, il programma si bloc

```python
# ❌ ERRORE: ciclo infinito!
x = 1
while x > 0:    # x sarà sempre > 0!
    print(x)
    x += 1      # x aumenta, non diminuirà MAI
```
````

Esempio: indovina il numero 🎯

```
import random

numero_segreto = random.randint(1, 20)
tentativi = 0

print("Indovina il numero (tra 1 e 20)!")

while True:
    tentativo = int(input("Il tuo tentativo: "))
    tentativi += 1

    if tentativo == numero_segreto:
        print(f"Bravo! Hai indovinato in {tentativi} tentativi! 🎉")
        break
    elif tentativo < numero_segreto:
        print("Troppo basso! ⬆️")
    else:
        print("Troppo alto! ⬇️")
```

`break` e `continue`

Due istruzioni speciali per controllare il flusso del ciclo. Sono tipo il freno a mano e il tasto "salta" di un lettore MP3:

`break` — Esci subito dal ciclo

```
# Trova il primo multiplo di 7
for i in range(1, 100):
    if i % 7 == 0:
        print(f"Il primo multiplo di 7 è: {i}")
        break  # Trovato! Me ne vado
```

`continue` — Salta questa iterazione e vai alla prossima

```
# Stampa solo i numeri dispari (salta i pari)
for i in range(1, 11):
    if i % 2 == 0:
        continue  # Pari? Nah, salto!
    print(i, end=" ")
```

Cicli annidati

Un ciclo dentro un altro ciclo. È tipo l'Inception dei loop — un sogno dentro un sogno! Utile per lavorare con tabelle e griglie:

```
# Tabellina pitagorica (da 1 a 5)
for i in range(1, 6):
    for j in range(1, 6):
        print(f"{i*j:4}", end="")
    print()  # A capo dopo ogni riga
```

Disegnare un triangolo di asterischi ⭐

```
altezza = 5

for i in range(1, altezza + 1):
    print("*" * i)
```

Accumulatori e contatori

Due pattern fondamentali che userai in continuazione. Sono tipo il pane e burro della programmazione con i cicli:

Contatore: contare quante volte succede qualcosa

```
frase = input("Scrivi una frase: ")

vocali = 0
for c in frase.lower():
    if c in "aeiou":
        vocali += 1

print(f"La frase contiene {vocali} vocali")
```

Accumulatore: sommare valori mano a mano

```
n = int(input("Quanti numeri vuoi sommare? "))
somma = 0

for i in range(n):
    numero = float(input(f"Numero {i+1}: "))
    somma += numero

media = somma / n
print(f"Somma: {somma}")
print(f"Media: {media:.2f}")
```

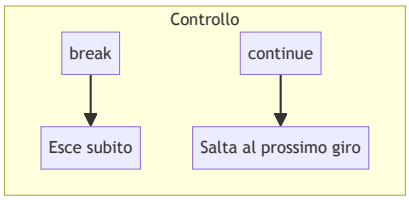
`for` ... `else` (sì, esiste!)

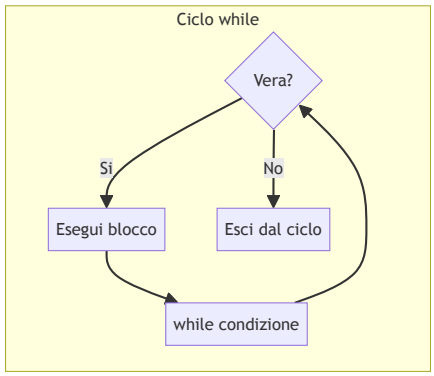
Plot twist: in Python puoi mettere un `else` dopo un ciclo! Il blocco `else` viene eseguito SOLO se il ciclo finisce **senza** `break`. Se fai `break`, l'`else` viene saltato. È tipo: "se non hai trovato niente, allora...":

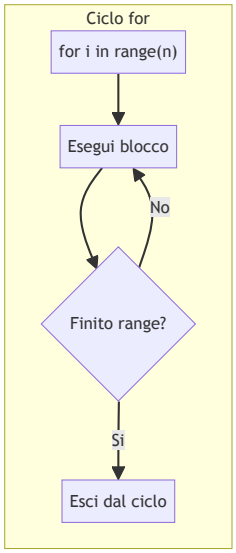
```
numeri = [4, 8, 15, 16, 23, 42]
cerca = int(input("Che numero cerchi? "))

for n in numeri:
    if n == cerca:
        print(f"Trovato: {n}! 🎯")
        break
else:
    print(f"{cerca} non è presente nella lista 😕")
```

Mappa concettuale







Esercizi

Esercizio 1: Somma da 1 a N

Calcola la somma dei numeri da 1 a N (dove N lo sceglie l'utente). Sì, esiste una formula matematica, ma qui devi usare il ciclo!

```
n = int(input("Inserisci N: "))

# Calcola la somma da 1 a N con un ciclo
```

??? success "Soluzione"

```
```pyodide
n = int(input("Inserisci N: "))
somma = 0
for i in range(1, n + 1):
    somma += i
print(f"La somma da 1 a {n} è {somma}")
```
```

Esercizio 2: Fattoriale

Calcola il fattoriale di un numero (es: 5! = 5 × 4 × 3 × 2 × 1 = 120). Attenzione: il fattoriale di 0 è 1!

```
n = int(input("Inserisci un numero: "))

# Calcola il fattoriale
```

??? success "Soluzione"

```
```pyodide
n = int(input("Inserisci un numero: "))
fattoriale = 1
for i in range(1, n + 1):
    fattoriale *= i
print(f"{n}! = {fattoriale}")
```
```

Esercizio 3: Numeri primi

Stampa tutti i numeri primi da 2 a N. Un numero è primo se è divisibile solo per 1 e per se stesso. Hint: ti servirà un ciclo dentro un ciclo! 🤯

```
n = int(input("Inserisci N: "))

# Stampa i numeri primi fino a N
```

??? success "Soluzione"

```
```pyodide
n = int(input("Inserisci N: "))
print(f"Numeri primi da 2 a {n}:")
for num in range(2, n + 1):
    is_primo = True
    for div in range(2, num):
        if num % div == 0:
            is_primo = False
            break
    if is_primo:
        print(num, end=" ")
```
```