Cicli for

Introduzione

Ecco un altro costrutto importantissimo nella programmazione, il ciclo **for**. Si tratta esattamente di un altro modo per scrivere dei cicli, quindi blocchi di codice che vengono eseguiti più volte in base a qualche istruzione. In questo caso c'è sempre una variabile (contatore o elemento) che determina il numero di ripetizioni del ciclo. Funziona esattamente come il ciclo **while** e tutto ciò che si può fare con il ciclo **for** si può fare con un **while** e viceversa, è solo questione di praticità.

La sua funzione principale, cioè la funzione per la quale viene utilizzato più spesso, è l'iterazione su sequenze di dati (liste, tuple, set e dizionari, che vedremo prossima lezione). Infatti, è un ciclo "semplificato" rispetto al **while** che permette di eseguire una serie di istruzioni per ogni elemento della lista.

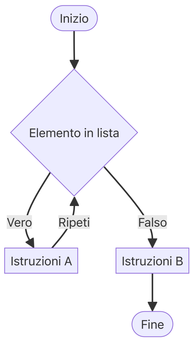
Sintassi:

```
for elemento in lista:

    blocco di codice eseguito se elemento è nella lista

blocco di codice eseguito quando elemento esce fuori dalla lista
```

Diagramma di flusso



Esempio

In [9]:
```
verdura = ["finocchio", "cavolo", "zucca"]
for x in verdura:
  print(x)
```

```
finocchio
cavolo
zucca
```

In [11]:
```
verdura = ["finocchio", "cavolo", "zucca"]
for elemento in verdura:
  print(elemento)
```

```
finocchio
cavolo
zucca
```

In

La parola chiave **in** è utilizzata per ottenere un'espressione booleana che determina la presenza di un elemento.

In [22]:
```
print("zucca" in verdura)
print("carciofo" in verdura)
```

```
True
False
```

Esercizio 1

Sommare tutti gli elementi di una lista data, la lista può avere lunghezza variabile.

In [29]:
```
numeri = [3, 6, 8, 12, 13, 14, 21, 28]
somma = 0
for n in numeri:
    somma += n
print("La somma è: " + str(somma))
```

```
La somma è: 105
```

La funzione range()

Per scorrere un blocco di codice un numero specifico di volte, possiamo utilizzare la funzione range().

La funzione range() restituisce una sequenza di numeri.

Sintassi:

```
range(start, stop, step)
```

Parameter	Description
start	Optional. An integer number specifying at which position to start. Default is 0
stop	Required. An integer number specifying at which position to stop (not included).
step	Optional. An integer number specifying the incrementation. Default is 1

La funzione range() ha come valore iniziale predefinito 0, tuttavia è possibile specificare il valore iniziale aggiungendo un parametro: range(2, 6), che significa valori da 2 a 6 (ma non includendo il 6).

La funzione range() incrementa la sequenza di 1 per impostazione predefinita, tuttavia è possibile specificare il valore di incremento aggiungendo un terzo parametro: range(2, 30, 3).

Quindi:

```
range(2, 30, 3)
```
Ritornerà una sequenza di numeri che vanno da 2 a 29 con passo 3.

> 2 5 8 11 14 17 20 23 26 29

Esempio

In [76]:
```
# Tabellina del 5
for x in range(5, 51, 5):
    print(x)
```

```
5
10
15
20
25
30
35
40
45
50
```

Esercizio 2

Stampare tutti i numeri della lista in posizione dispari.

In [65]:
```
lista = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

for i in range(1, len(lista), 2):
    print(lista[i])
```

```
20
40
60
80
100
```

Break & Continue

Break

Come con il **while**, con la parola chiave **break** possiamo fermare il ciclo prima che finisca di eseguire tutte le iterazioni per tutti gli elementi della lista.

Esempio

In [90]:
```
verdura = ["finocchio", "cavolo", "zucca", "zucchina", "carota"]
for x in verdura:
  print(x)
  if x == "zucca":
    break
```

```
finocchio
cavolo
zucca
```

Continue

Come nel **while**, con la parola chiave **continue** possiamo interrompere l'esecuzione di un'iterazione del ciclo e riprendere dalla successiva.

Esempio

In [88]:
```
verdura = ["finocchio", "cavolo", "zucca", "zucchina", "carota"]
for x in verdura:
    if x == "zucca":
        continue
    print(x)
```

```
finocchio
cavolo
zucchina
carota
```