

Cicli for

Informatica@DSS 2025/2026

Massimo Lauria <massimo.lauria@uniroma1.it>
<https://massimolauria.net/informatica2025/>

I cicli for

Significato del ciclo `for`

Ripeti il seguente codice per ogni elemento nella lista

<code>for x in lista:</code>	1
<code>istruzione1</code>	2
<code>istruzione2</code>	3
<code>istruzione3</code>	4
<code>...</code>	5
	6

Se la lista contiene n elementi, il codice contenuto all'interno del ciclo viene eseguito n volte, con `x` che assume uno per uno i valori nella sequenza.

Tre elementi del ciclo for

for x in lista:	1
istruzione1	2
istruzione2	3
istruzione3	4
...	5
	6

- ▶ la lista di valori su cui si cicla
- ▶ il nome `x` associato ai valori della lista
- ▶ il codice che viene eseguito ad ogni ripetizione

Esempio

```
seq = [1, 15, 2+5, "casa"+"gatto", -6]
```

1

2

```
for x in seq:
```

3

```
    print(x)
```

4

1

15

7

casagatto

-6

Esempio 2

```
L = [1, 2, 3, 4, 5]
```

1

2

```
for a in L:
```

3

```
    print(a*a)
```

4

1

4

9

16

25

Variabile del ciclo

Un ciclo `for` crea una variable che viene associata ai valori nella lista, il cui nome è scelto dal programmatore.

Se L contiene $v_0, v_1, v_2, \dots, v_{n-1}$ allora il codice nel ciclo `for` viene eseguito n volte con

- ▶ `x` settato a v_0 ;
- ▶ `x` settato a v_1 ;
- ▶ ...
- ▶ `x` settato a v_{n-1}

E poi il ciclo termina.

Come usare il ciclo for?

Non c'è molto altro da dire sul *come funziona* il ciclo for.
Ma naturalmente non è ovvio come usarlo.

Il limite è la vostra fantasia, ma ora vedremo degli esempi
tipici che possono chiarirne la funzionalità.

Stampa di una lista

Abbiamo già visto la semplice scansione degli elementi di una lista

```
seq = [1, 5, 2+3, "casa"+"gatto", -6]
```

1

2

```
for x in seq:
```

3

```
    print(x)
```

4

```
1
5
5
casagatto
-6
```

Conteggio degli elementi

Fa la stessa cosa della funzione `len`

```
L = [1, 5, 2+3, "casa"+"gatto", -6] 1
                                     2
contatore=0                           3
for elemento in L:                     4
    contatore = contatore + 1          5
                                     6
print(contatore)                       7
print(len(L))                          8
```

```
5
5
```

Conteggio di elementi maggiori di zero

Contiamo **solo** gli elementi che soddisfano una certa condizione (testata con un **if**).

```
L = [1, 5, -3, 7, 0, -6] 1
                             2
contatore = 0              3
for elemento in L:         4
    if elemento > 0:       5
        contatore = contatore + 1 6
                             7
print(contatore)           8
```

3

Trasformazione/Filtraggio di dati

Costruiamo un **nuova** lista con i dati tradotti o filtrati.

Ripassiamo due nozioni utili:

- ▶ L'espressione `[]` indica una lista vuota
- ▶ Data una lista `T`, `T.append(expr)` aggiunge alla lista `T` il valore dell'espressione `expr`

```
X = [] 1
print(X) 2
X.append("gatto") 3
print(X) 4
X.append(12) 5
print(X) 6
```

```
[]
['gatto']
['gatto', 12]
```

Eliminazione dei valori negativi

1. creiamo una lista vuota
2. scansione della lista vecchia
3. inseriamo in quella i valori non negativi

```
Input = [1, 5, -3, 7, 0, -6]      1
Output = []                       2
for val in Input:                 3
    if val >= 0:                   4
        Output.append(val)        5
print(Output)                     6
                                  7
```

```
[1, 5, 7, 0]
```

Trasformazione in maiuscolo

Se `expr` è una espressione di tipo stringa, `expr.upper()` è la stessa stringa, ma tradotta in maiuscole.

```
L = [ 'casa', 'Gianni', 'Falò', 'Covid-19', '12345' ]      1
R = []                                                       2
                                                            3
for val in L:                                                4
    R.append(val.upper())                                    5
                                                            6
print(R)                                                     7
```

```
['CASA', 'GIANNI', 'FALÒ', 'COVID-19', '12345']
```

Calcolo dei quadrati

Calcoliamo i quadrati dei primi 10 numeri

L = [0,1,2,3,4,5,6,7,8,9]	1
R = []	2
for x in L:	3
R.append(x*x)	4
	5
print(R)	6

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Indici di posizione

La lista canonica dei primi n numeri

`range(n)` parte da 0 e arriva a $n - 1$

```
X = range(10)      1
for i in X:         2
    print(i, end=' ') 3
                    4

print()             5
                    6

Y = range(15)       7
for i in Y:         8
    print(i, end=' ') 9
```

```
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

Le posizioni degli elementi di una lista

In una lista L di lunghezza $\text{len}(L)$

- ▶ la posizione 0 contiene il primo dei valori;
- ▶ la posizione $\text{len}(L)-1$ contiene l'ultimo

<code>L = ['miao', 32, 'casa', 'gatto', -7, 1.2]</code>	1
<code>n = len(L)</code>	2
<code>print(L[0])</code>	3
<code>print(L[n-1])</code>	4

```
miao
1.2
```

Le posizioni della lista e range

L'espressione `range(len(L))` calcola la sequenza delle posizioni della lista `L`.

```
L = ['miao', 32, 'casa', 'gatto', -7, 1.2]      1
posizioni = range(len(L))                      2
                                                3
for i in posizioni:                             4
    print("Alla posizione",i,"ci sta l'elemento", L[i]) 5
```

```
Alla posizione 0 ci sta l'elemento miao
Alla posizione 1 ci sta l'elemento 32
Alla posizione 2 ci sta l'elemento casa
Alla posizione 3 ci sta l'elemento gatto
Alla posizione 4 ci sta l'elemento -7
Alla posizione 5 ci sta l'elemento 1.2
```

Due modi per scorrere i valori di una lista

Il fatto che `range(len(L))` enumeri le posizioni della lista permette di scorrere la lista in due modi:

<code>L = ['miao', 32, 'casa', 'gatto', -7, 1.2]</code>	1
<code>for x in L:</code>	2
<code>print(x)</code>	3
<code>for i in range(len(L)):</code>	4
<code>print(L[i])</code>	5

```
miao
32
casa
gatto
-7
1.2
miao
32
casa
gatto
-7
1.2
```

Calcolo del minimo

Calcolo del minimo di una lista I

- ▶ minimo di [] non definito
- ▶ mantengo il minimo visto fino a quel momento
- ▶ agguirno **se** trovo un "minimo migliore"
- ▶ il minimo deve essere inizializzato!

```
Input = [1, 5, -3, 7, 0, -6, 13, 2, 8]
```

1

2

```
temp=Input[0]  # errore se Input == []
```

3

```
for x in Input:
```

4

```
    if x<temp:
```

5

```
        temp=x
```

6

```
print(temp)
```

7

-6

Calcolo del minino di una lista II

Facciamo la stessa cosa con gli indici

```
Input = [1, 5, -3, 7, 0, -6, 13, 2, 8] 1
2
temp=Input[0] # errore se Input == [] 3
for i in range(len(Input)): 4
    if Input[i]<temp: 5
        temp=Input[i] 6
print(temp) 7
```

-6

Calcolo del minimo di una lista III

Non abbiamo bisogno di visitare di nuovo la prima posizione. Usiamo `range(a,b)` che produce la sequenza `a, a+1, ..., b-1`.

```
Input = [1, 5, -3, 7, 0, -6, 13, 2, 8] 1
2
temp=Input[0]  # errore se Input == [] 3
for i in range(1,len(Input)): 4
    if Input[i]<temp: 5
        temp=Input[i] 6
print(temp) 7
```

-6

Esercizio

Scrivete un programma che calcoli il massimo in una lista.

Ricerca in una lista

Ricerca in una lista I

Dato un valore `target` da cercare: si fa una scansione della lista e ad ogni passo:

- ▶ se ci si trova su un valore uguale a `target` allora **SUCCESSO**, non è necessario proseguire
- ▶ altrimenti, si passa a controllare il successivo

Se si arriva alla fine della scansione, allora **INSUCCESSO**.

Ricerca in una lista II

Per questi esempi usiamo le funzioni

```
def ricerca(target,dati):           1
    for x in dati:                 2
        if x == target:           3
            return True            4
    return False                    5

print(ricerca("ago",["paglia", "fieno", "paglia", 6
                    "fieno", "ago", "fieno", "paglia"])) 7
print(ricerca("delfino",["cane", "gatto", "volpe", "cavallo"])) 8
                                                                9
```

True
False

Ricerca in una lista III

Variazione: restituiamo la posizione dell'elemento cercato, oppure -1.

```
def ricerca(target,dati):  
    for i in range(len(dati)):  
        if dati[i] == target:  
            return i  
    return -1  
  
print(ricerca("ago",["paglia", "fieno", "paglia",  
                    "fieno", "ago", "fieno", "paglia"]))  
print(ricerca("delfino",["cane", "gatto", "volpe", "cavallo"]))
```

```
4  
-1
```

Verifica di proprietà di una lista

Due semplici tipi di verifiche:

- ▶ **esiste** un elemento con proprietà P
- ▶ **tutti** gli elementi hanno proprietà P

E.g. la sequenza contiene una parola di tre lettere.

E.g. la sequenza contiene solo valori positivi.

Verifica di proprietà di una lista

Due semplici tipi di verifiche:

- ▶ **esiste** un elemento con proprietà P
- ▶ **tutti** gli elementi hanno proprietà P

E.g. la sequenza contiene una parola di tre lettere.

E.g. la sequenza contiene solo valori positivi.

Sono variazioni della ricerca in lista

Verifica dell'esistenza

Si scandisce la lista e ad ogni elemento x :

- ▶ se x ha la proprietà desiderata, **SUCCESSO**
- ▶ altrimenti continuiamo con l'elemento successivo

Se arriviamo alla fine allora **FALLIMENTO**

Verifica dell'esistenza

Si scandisce la lista e ad ogni elemento x :

- ▶ se x ha la proprietà desiderata, **SUCCESSO**
- ▶ altrimenti continuiamo con l'elemento successivo

Se arriviamo alla fine allora **FALLIMENTO**

Ricerca di un elemento con la proprietà

Esempio: contiene un numero pari?

Trovato un numero pari, non serve continuare a scandire la lista. Ma fino a che non si trova, è necessario farlo perché il numero pari potrebbe essere nelle posizioni successive.

```
def haunpari(L):
    for x in L:
        if x % 2 == 0:
            return True
    return False

print( haunpari([3,9,5,4,12,21]) )
print( haunpari([23,-9,15,23,11,21]) )
```

1
2
3
4
5
6
7
8

True
False

Esempio: contiene una parola di 3 lettere?

```
def trelettere(L): 1
    for x in L: 2
        if len(x) == 3: 3
            return True 4
    return False 5
6
print(trelettere(['ciao', 'cartello', 'Venezia', 'burro'])) 7
print(trelettere(['nano', 'cowboy', 'eco', 'finestra'])) 8
```

False

True

Verifica dell'universalità

Si scandisce la lista e ad ogni elemento x :

- ▶ se x **NON** ha la proprietà desiderata, **FALLIMENTO**
- ▶ altrimenti continuiamo con l'elemento successivo

Se arriviamo alla fine allora **SUCCESSO**

Verifica dell'universalità

Si scandisce la lista e ad ogni elemento x :

- ▶ se x **NON** ha la proprietà desiderata, **FALLIMENTO**
- ▶ altrimenti continuiamo con l'elemento successivo

Se arriviamo alla fine allora **SUCCESSO**

*Ricerca di un elemento senza la
proprietà*

Esempio: contiene solo positivi?

Si controlla la negazione della proprietà.

```
L = [1, 5, -3, 7, 0, -6, 13, 2, 8] 1
                                     2
def allpositive(S):                 3
    for x in S:                     4
        if x<=0:                   5
            return False           6
    return True                     7
                                     8
print(allpositive(L))               9
print(allpositive([20,12,1]))      10
```

False

True

Esistenza e Universalità

<code>for x in L:</code>	1
<code>if P(x) è vera:</code>	2
esiste un valore con proprietà P	3
nessun valore ha la proprietà P	4

Notate come negli esempi visti un `return` sia dentro al ciclo e uno dopo fine del ciclo.

1. (dentro il ciclo) un solo dato determina il risultato
2. (alla fine del ciclo) il risultato è determinato dopo aver visto tutti gli elementi.

Verifica dell'ordinamento I

Una lista L è ordinata se due elementi adiacenti sono uno minore o uguale dell'altro.

Dobbiamo verificare per ogni coppia $i, i+1$

- ▶ che $L[i] \leq L[i+1]$

Osservazione: è come verificare una proprietà universale, però la proprietà non è legata ai singoli elementi, ma alle coppie di elementi adiacenti.

Esercizio: i deve scorrere da 0 a $\text{len}(L)-2$. Perché?

Verifica dell'ordinamento II

```
def ordinata(L):
    for i in range(len(L)-1):
        if L[i] > L[i+1]:
            return False
    return True

print(ordinata([4,6,8,10]))
print(ordinata([4,6,8,10,5]))
```

1
2
3
4
5
6
7
8

True
False