

# esempi

I seguenti programmi **terminano**? ...*sempre, mai* o *qualche volta*?

```
n = int(input())
while n > 0:
    print(n)
    n = n - 2
```

ok

```
n = int(input())
while n != 0:
    print(n)
    n = n - 2
```

se n è **negativo** loop  
se n è pari OK  
se n è **dispari** loop

```
n = int(input())
while n != 1:
    print(n)
    if n%2 == 0: # n e' pari
        n = n/2
```

n=8 ok 8 4 2 1  
n=7 **loop** 7 7 7 ...  
n=24 **loop** 24 12 6 3 3...

# esercizi

I seguenti programmi **terminano**? ...*sempre, mai* o *qualche volta*?

```
n = int(input())
while n != 1:
    print(n)
    if n%2 == 0: # n e' pari
        n = n/2
    else: # n e' dispari
        n = (n+1)/2
```

si può dimostrare che  
 $\forall n > 0$  termina

```
n = int(input())
while n != 1:
    print(n)
    if n%2 == 0: # n e' pari
        n = n/2
    else: # n e' dispari
        n = n*3+1
```

**$\forall n > 0$  termina ?**

I matematici credono di sì, ma nessuno  
è mai riuscito a **dimostrarlo**!

## Congettura di Collatz

- La congettura è stata verificata **mediante computer** (giorni e giorni, rete di computer) per tutti i valori **fino a  $2^{68}$**  circa  $10^{20}$ . (<http://www.ericr.nl/wondrous/>)
- Questi test non potranno **mai dimostrare la correttezza** della congettura, ma **solo l'eventuale falsità**.

# Funzioni

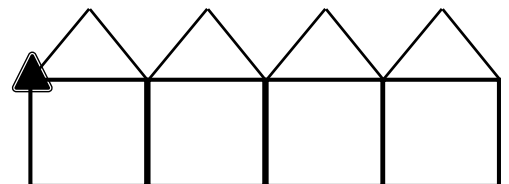
```
# procedura per disegnare una fila di case:
def draw_a_row_of_houses():
    max_area=240
    position=0
    # 50 is the dimension of a house
    while position+50 < max_area:
        draw_house(50)
        print('sposta la penna')
        position=position+50

# procedura per disegnare una casa
def draw_house(dim):
    draw_square(dim)
    draw_triangle(dim)

# procedura per "disegnare" un quadrato
def draw_square(dim):
    print('disegna un quadrato')

# procedura per "disegnare" un triangolo
def draw_triangle(dim):
    print('disegna un triangolo')

# programma che disegna una fila di case:
# invoca l'algoritmo definito sopra
draw_a_row_of_houses()
```



## Decomposizione

dall'algoritmo  
all'implementazione



è ben diverso leggere un programma  
dall'alto in basso oppure seguendo il  
flusso di esecuzione!

# Funzioni

intestazione: **nome** e **lista parametri**

```
1 def incipit():  
2     print('Tanto tempo fa')  
3     print('in una galassia')  
4     print('lontana, lontana ...')  
5  
6 incipit()  
7 print('Non ho capito')  
8 incipit()
```

**corpo:** blocco di istruzioni

**definizione di funzione**

**chiamata di funzione**

- Una **funzione** (*procedura*) è una serie di istruzioni a cui viene assegnato un nome.
- Il corpo della funzione contiene delle istruzioni che **vencono eseguite solo quando (e se) la funzione viene chiamata/invocata** mediante il suo nome

Una funzione deve essere definita prima di essere chiamata (*Esercizio: provare e leggere il messaggio di errore*)

## Funzioni e flusso di controllo

```
1 def incipit():  
2     print('Tanto tempo fa')  
3     print('in una galassia')  
4     print('lontana, lontana ...')  
5  
6 incipit()  
7 print('--Non ho capito')  
8 incipit()
```

**istruzioni eseguite:**  
6, 2,3,4, 7,8 ,2,3,4

Tanto tempo fa  
in una galassia  
lontana, lontana ...  
--Non ho capito  
Tanto tempo fa  
in una galassia  
lontana, lontana ...

- L'esecuzione inizia sempre dalla prima istruzione/comando, procedendo un'istruzione per volta dall'alto verso il basso.
- Quando viene **chiamata una funzione**, il flusso di controllo anziché proseguire con l'istruzione successiva, **salta nel corpo della funzione chiamata**, ne esegue le istruzioni, e infine **riprende il percorso dal punto che aveva lasciato**

# Funzioni e flusso di controllo

```
1 def incipit():
2     print('Tanto tempo fa')
3     print('in una galassia')
4     print('lontana, lontana ...')
5
6 def duplica():
7     incipit()
8     incipit()
9
10 duplica()
11 print('Bye Bye')
```

istruzioni eseguite:

10, 7, 2,3,4, 8, 2,3,4, 11

```
Tanto tempo fa
in una galassia
lontana, lontana ...
Tanto tempo fa
in una galassia
lontana, lontana ...
Bye Bye
```

- L'esecuzione inizia sempre dalla prima istruzione/comando, procedendo un'istruzione per volta dall'alto verso il basso.
- Quando viene **chiamata una funzione**, il flusso di controllo anziché proseguire con l'istruzione successiva, **salta nel corpo della funzione chiamata**, ne esegue le istruzioni, e infine **riprende il percorso dal punto che aveva lasciato**

## Funzioni

- La funzione può avere dei **parametri**, allora quando la si invoca bisogna **passarle degli argomenti**, uno per ogni parametro.
- La funzione, come ultima cosa, può restituire un **valore di ritorno**

```
print('Hello', 3)    chiama la funzione con 2 argomenti,
                    nessun valore di ritorno
```

```
msg = input()       chiama la funzione senza argomenti, ritorna una
                    stringa, usata per inizializzare la variabile msg
```

```
# procedura per "disegnare" un quadrato
def draw_triangle(dim):
    print('disegna triangolo di lato', dim)
```

**definisce** la funzione di nome `draw_triangle` con 1 parametro e nessun valore di ritorno (quindi va chiamata con 1 argomento)

# Funzioni e parametri

```
1 def stampa2volte(s):  
2     print(s)  
3     print(s)  
4  
5  
6 stampa2volte('Tony')  
7 stampa2volte('Bruce')
```

**parametro** (è una variabile che si usa nel corpo della funzione)

**argomento**

```
1 def stampa_coppia(s1,s2):  
2     print(s1,'loves', s2 )  
3  
4 stampa_coppia('Tony','Pepper')  
5 stampa_coppia('Bruce','Betty')
```

**2 parametri -- 2 argomenti**

Tony loves Pepper  
Bruce loves Betty

# Funzioni e parametri

```
def stampa2volte(s):  
    print(s)  
    print(s)
```

**parametro**

```
stampa2volte('pippo')
```

pippo  
pippo

```
stampa2volte('bello'*3)
```

bellobellobello  
bellobellobello

**gli argomenti sono *espressioni*:**

- se ne **valuta** il valore prima della chiamata
- i parametri sono **inizializzati** con questo valore

```
msg = 'Tony'  
stampa2volte(msg)
```

- analogo se si **passa una variabile come argomento**
- la variabile passata **msg** "non esiste" nel corpo della funzione, che usa invece solo **s**

```

# procedura per disegnare una fila di case:
def draw_a_row_of_houses():
    max_area=240
    position=0
    # 50 is the dimension of a house
    while position+50 < max_area:
        draw_house(50)
        print('sposta la penna')
        position=position+50

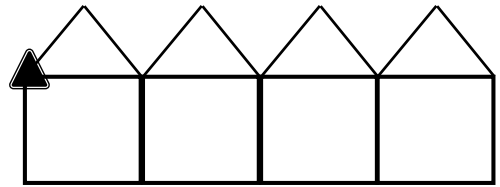
# procedura per disegnare una casa
def draw_house(dim):
    draw_square(dim)
    draw_triangle(dim)

# procedura per "disegnare" un quadrato
def draw_square(dim):
    print('disegna un quadrato')

# procedura per "disegnare" un triangolo
def draw_triangle(dim):
    print('disegna un triangolo')

# programma che disegna una fila di case:
# invoca l'algoritmo definito sopra
draw_a_row_of_houses()

```



invoca la funzione, quindi ne  
esegue il corpo  
passando come  
argomento il valore 50

esegue le istruzioni  
`draw_square(50)`  
`draw_triangle(50)`  
che sono altre due  
chiamate di funzioni

invoca la funzione, quindi  
ne esegue il corpo

## valori di ritorno

```

def calcola_area(raggio):
    a = 3.14 * (raggio**2)
    return a

```

la funzione **termina** e  
restituisce un **valore**

valore:  
1256

```

x = calcola_area(20)
print("area cerchio di raggio 20:", x)

```

```

r = int( input("Che raggio vuoi?"))
y = calcola_area(r)
print("L'area del cerchio di raggio", r, " è ", y)

```

```

r = int( input("Che raggio vuoi?"))
print("L'area del cerchio di raggio", r, " è ", calcola_area(r))

```

una chiamata di funzione  
è un'**espressione** che ha come **valore**  
il valore ritornato dalla funzione

# valori di ritorno

```
def calcola_tassa( reddito, aliquota ):
    # la tassa è una percentuale del reddito
    tassa = reddito * (aliquota/100)
    return tassa
```

la funzione prende  
2 parametri e  
restituisce un **valore**

```
stipendio = 3500
trading = 1000
print("tassa su reddito:", calcola_tassa(stipendio,15))
print("tassa su reddito da capitale:", calcola_tassa(trading,26))
```

valore: 525

valore: 260

# valori di ritorno

```
def scelta_aliquota(totale):
    if totale < 200000 :
        return 15
    else:
        return 25
```

- se il flusso di controllo si ramifica, **tutti i rami devono restituire lo stesso numero di valori**

```
stipendio = 350000
a = scelta_aliquota(stipendio)
t = calcola_tassa(stipendio, a )
```

le variabili **a** e **t** sono inizializzate con il valore restituito dalla chiamata delle funzioni

```
def divisibile(x,y):
    if x%y ==0:
        return True
    else:
        return False
```

```
print("Posso dividere 15 per 4? Risposta: ", divisibile(15,4))
print("Posso dividere 328 per 14? Risposta: ", divisibile(328,14))
print("Posso dividere 777 per 11? Risposta: ", divisibile(777,11))
```

# valori di ritorno

```
def stampa2volte(s):  
    print(s)  
    print(s)  
    return
```

quando non ha nessun valore da ritornare,  
la keyword `return` si può omettere  
(sta per `return None`)

## Esempio

- Funzione che conta **quante volte** una data lettera è presente in una data parola:

```
def conta(lettera, parola):  
    n=0  
    for i in parola:  
        if i == lettera:  
            n = n+1  
    return n
```

```
x = conta('p', 'appalto')  
y = conta('t', 'attrattore')
```

```
n=0  
for i in 'appalto':  
    if i == 'p':
```

```
n=0  
for i in 'attrattore':  
    if i == 't':  
        n=n+1  
return n
```

```
temp = input('inserisci una parola')  
y = conta('p', temp)  
print('la p compare', y, 'volte in', temp)
```



# Esercizio

- Funzione che cerca **se** una data lettera è presente in una data parola:

```
def cerca(lettera, parola):  
    trovato = False  
    for i in parola:  
        if i == lettera:  
            trovato = True  
    return trovato
```

```
x = cerca('p', 'appalto')
```

- confronta la funzione precedente con questa:

```
def cerca(lettera, parola):  
    trovato = False  
    for i in parola:  
        if i == lettera:  
            trovato = True  
        else:  
            trovato = False  
    return trovato
```

# Esercizio

- Funzione che cerca **se** una data lettera è presente in una data parola:


```
def cerca(lettera, parola):  
    trovato = False  
    for i in parola:  
        if i == lettera:  
            trovato = True  
    return trovato
```

```
x = cerca('p', 'appalto')
```

- confronta la funzione precedente con questa, anche tracciando il flusso di controllo generato dalla chiamata delle due diverse funzioni:

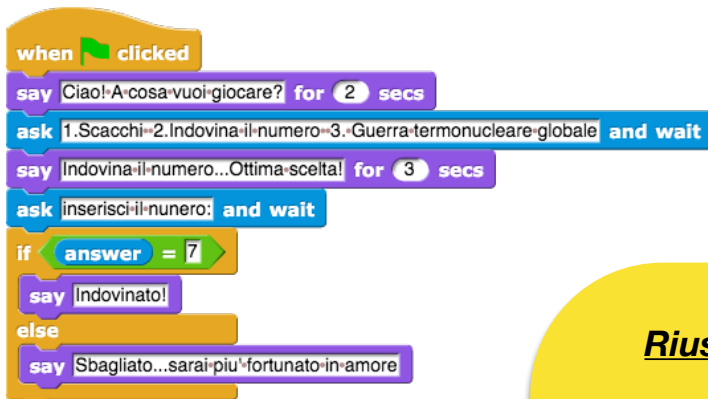
```
def cerca(lettera, parola):  
    trovato = False  
    for i in parola:  
        if i == lettera:  
            return True  
    return False
```

*termina subito la funzione  
senza continuare con le  
istruzioni successive, quindi  
senza completare il ciclo for*



# perché usare le funzioni

- dare un nome ad un gruppo di istruzioni (*functional* or *procedural abstraction*) rende il programma **più facile** da **leggere** e **correggere** (*code readability*)
  - le funzioni rendono il programma più breve, eliminando il codice ripetitivo (*code reuse*).
  - Se in un secondo momento bisogna fare una **modifica** a quel gruppo di istruzioni, basta farla in un posto solo (*manutenibilità e generalizzazione*, es. aggiungo un parametro)
- dividere un programma lungo in funzioni vi permette di correggere le parti una per una, per poi assemblarle in un complesso funzionante (*decomposition, separation of concerns, unit testing*)
- Funzioni ben fatte sono spesso utili per più programmi. Una volta scritta si può riutilizzare (code reuse e **librerie**)



**Riusa il codice** del programma  
“indovina il numero”:

- “impacchetta” il codice in una funzione
- invoca la funzione in questo programma, se l'utente ha scelto il gioco 2.

```
print('Ciao! a cosa vuoi giocare?')
input('1.Scacchi 2.Indovina il numero 3.Guerra termonucleare globale')
print('Hai scelto: Indovina il numero Ottima scelta!')
answer = int(input('inserisci il numero: '))
if answer == 7 :
    print('Indovinato!')
else:
    print('Sbagliato...sarai piu fortunato in amore')
```

# strutture dati e data abstraction

## I dati in uso

“Il programma chiede di inserire una **data** e fa...”

“Il programma saluta l’utente stampando una **citazione** ...”

“Il programma chiede di inserire un **numero di telefono** e fa...”

“Il programma gestisce una **rubrica telefonica** ...”

“Il programma tiene traccia di una **rete di amicizie** ...”

### i dati del problema

serve trovare un modo per **rappresentarli nel programma**  
usando i costrutti del linguaggio di programmazione

**i dati del programma**

come rappresento una *data/citazione/tel...* in un programma python?

# I dati in uso

giorno=5  
mese=11  
anno=2020

“Il programma chiede di inserire una **data** e fa...”

“Nel mezzo del...”

“Il programma saluta l’utente stampando una **citazione** ...”

“Il programma chiede di inserire un **numero di telefono** e fa...”

“Il programma gestisce una **rubrica telefonica** ...”

tel= '04982714'

“Il programma tiene traccia di una **rete di amicizie** ”

## i dati del problema

????

serve trovare un modo per **rappresentarli**  
usando i costrutti del linguaggio di programmazione

```
nome1 = 'Mario'  
tel1 = '003933915'  
nome2 = 'Barbara'  
tel2 = '003947195'  
...
```

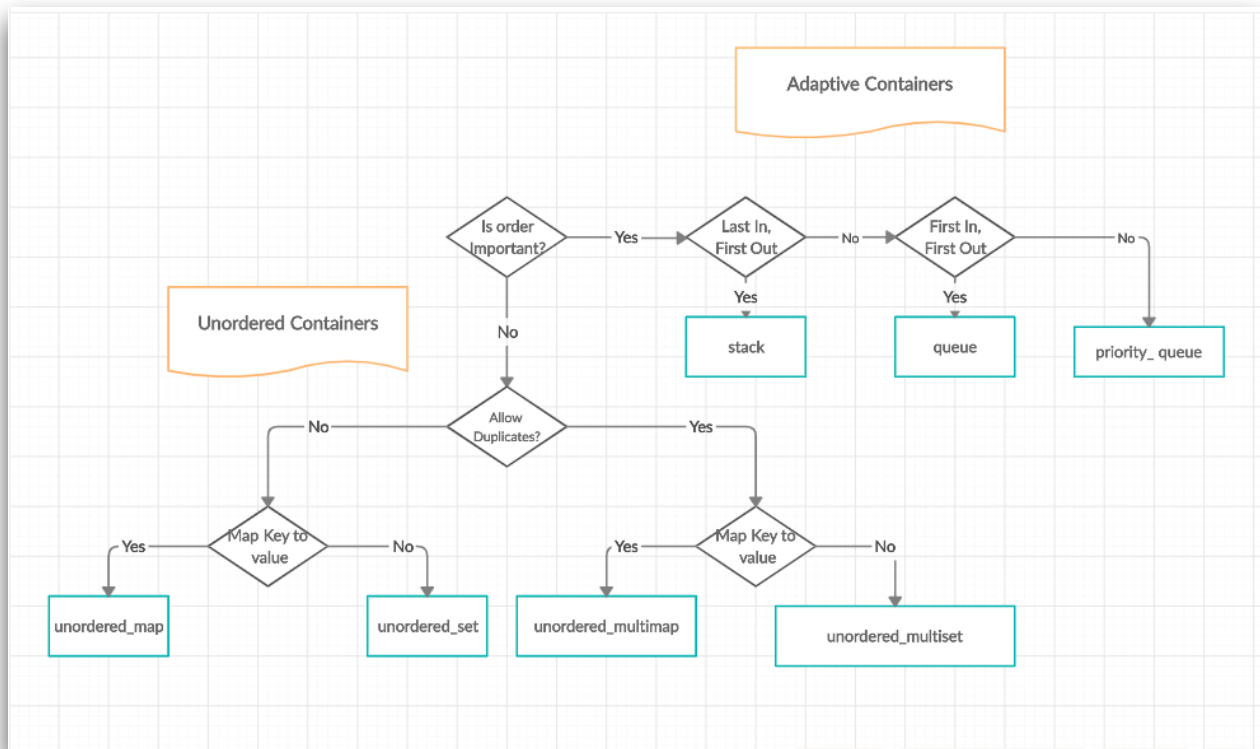
## i dati del programma

come rappresento una *data/citazione/tel...* in un programma python?

# strutture dati

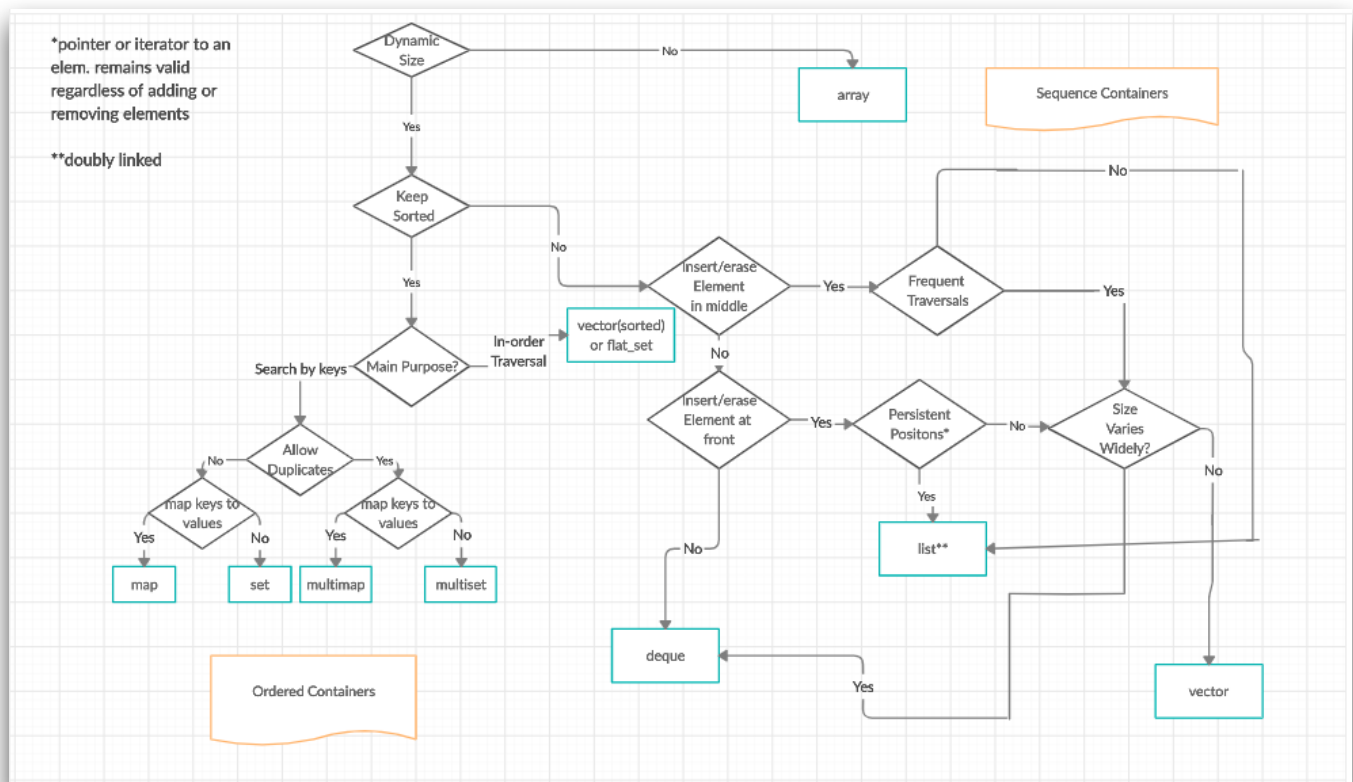
- i dati più semplici sono rappresentati da **valori primitivi** (*interi, decimali, booleani, caratteri, stringhe*), memorizzati in una variabile, o più di una
- i programmi lavorano spesso con dati di natura più complessa, es. *elenchi di persone, tabelle, anagrafe dei clienti, rete di amici, ...*
  - i **dati** che si usano vanno *rappresentati* attraverso i valori e le **strutture dati** offerte dal linguaggio di programmazione.
  - *liste, array, coppie, tuple, dizionari, insiemi, stack, alberi...*
- **Ogni linguaggio** di programmazione offre una **serie di strutture dati**, ciascuna con caratteristiche diverse
  - es. in alcuni casi i *dizionari python* più efficienti dei *vettori C++*
  - es. *Java8* introduce gli *Stream* per **big data applications**

# The C++ Standard Template Library (STL)



<https://www.geeksforgeeks.org/the-c-standard-template-library-stl/>

# The C++ Standard Template Library (STL)



<https://www.geeksforgeeks.org/the-c-standard-template-library-stl/>

# algoritmo e strutture dati

Data la specifica di un problema, **il programmatore deve:**

- individuare i passi che lo risolvono: **algoritmo** risolutivo, e la **sua implementazione** nel linguaggio di programmazione
- scegliere **come rappresentare i dati del problema** usando le **strutture dati più opportune**

**le due cose sono legate:**

a seconda della struttura dati che sceglie, l'implementazione del programma può risultare più o meno **efficiente** (anche drasticamente) ma anche più **leggibile**, e più **manutenibile**

## liste

- Una **lista** è una sequenza di valori, di tipo qualsiasi, detti **elementi** della lista

```
[10,20,30,40] # una lista di interi
```

```
['antipasto','primo','secondo','dessert'] # lista di stringhe
```

```
[10,'pippo',3.14] # lista di valori diversi
```

```
[[1,2],[3,4],[5,6,7]] # lista di liste di interi
```

```
[] # lista vuota
```

```
numeri = [10,200]
```

```
menu = ['antipasto','primo']
```

```
vuota = []
```

```
print(numeri,menu,vuota) #stampa [10,200] ['antipasto','primo'] []
```

# liste

- Data una lista, si può **accedere a ciascun elemento**, per leggerlo oppure modificarlo, tramite il suo **indice**
- Gli elementi hanno indice **da 0** fino alla **lunghezza della lista-1**

```
menu = ['antipasto', 'primo', 'secondo', 'dessert']
print(menu[0])           # stampa antipasto
menu[0] = 'aperitivo'    # sovrascrive l'elemento di indice 0
menu[-1] = 'gelato'      # sovrascrive l'elemento di indice -1
print(menu)              # stampa ['aperitivo', 'primo', 'secondo', 'gelato']

for piatto in menu:      # variabile piatto itera nell'elenco menu
    print(piatto)

x = [4, 55, 20, 12]
quanti_sono = len(x)     # lunghezza lista: 4 (len funzione built-in)

# raddoppio i valori nella lista
for i in range(len(x)):  # i in 0,1,2,3 cioè itera sugli indici
    x[i] = x[i]*2
print(x)                 # stampa [8, 110, 40, 24]
```

## operazioni sulle liste

```
a = [1, 2, 3]
b = [4, 5, 6]
c = a + b  # operazione che concatena le liste
print(c)   # [1, 2, 3, 4, 5, 6]
print(a)   # [1, 2, 3]
```

le liste possono essere passate  
come parametri alle funzioni

```
t = ['d', 'o', 'f']
t.append('q')  # operazione che aggiunge un elemento in coda
print(t)       # ['d', 'o', 'f', 'q']
```

```
t1 = ['b', 'l']
t.extend(t1)   # aggiunge in coda tutti gli elementi di t1
print(t)       # ['d', 'o', 'f', 'q', 'b', 'l']
```

```
t.sort()       # ordina gli elementi di t
print(t)       # ['b', 'd', 'f', 'l', 'o', 'q']
```

```
x = t.pop(1)   # elimina e restituisce elemento di indice 1
t.remove('q')  # elimina elemento
print(t, x)    # ['b', 'f', 'l', 'o'] 'd'
```

# Esercizi

- Scrivere una funzione che prende due parametri: un intero `n` e una lista di interi `list`, e **cerca se** (cioè restituisce `True` se) **l'intero `n` è presente nella lista `list`**

```
# fun sarà chiamata con 2 argomenti:  
# un intero e una lista di interi  
def fun(n,list):  
    ...
```

- Scrivere una funzione che prende due parametri: un intero `n` e una lista **ordinata** di interi `list`, e **cerca se** (cioè restituisce `True` se) **l'intero `n` è presente nella lista `list`**

```
# fun_ord sarà chiamata con 2 argomenti:  
# un intero e una lista già ORDINATA di interi  
def fun_ord(n,list):  
    ...
```

# Esercizi

1. Scrivere un **ALGORITMO** che data una lista di numeri produce in output la lista con quei numeri ordinati in senso crescente
2. Implementare l'algoritmo in un **SOFTWARE** python

**NOTA:** questo esercizio è difficile da svolgere correttamente. Ma è utile provare ad abbozzare un algoritmo, rendersi conto se è corretto almeno in qualche caso, e provare ad implementarlo.



# Esercizio

```
a=int(input('primo numero: '))
b=int(input('secondo numero: '))
c=int(input('terzo numero: '))
d=int(input('quarto numero: '))
```

```
lista=[a,b,c,d]
max=a
for i in lista:
    if a>i:
        max=a
    if b>i:
        max=b
    if c>i:
        max=c
    if d>i:
        max=d
```

```
a=int(input('primo numero: '))
b=int(input('secondo numero: '))
c=int(input('terzo numero: '))
d=int(input('quarto numero: '))
```

```
lista=[a,b,c,d]
max=a
for i in lista:
    if i > max:
        max=i
```

- Confrontare il flusso di controllo di questi due programmi
- Calcolano entrambi correttamente il massimo tra i 4 numeri inseriti?