In [1]:
```
#
# Exercises by prof. Iacopo Masi
#
```

In [2]:
```
from evaluation import evaluate, show_tests

"""
Esercitazione-Python-A
Benvenuti!
"""

"""
# Esercitazione Python
# Autovalutazione su interi, float, stringhe, istruzioni condiziona
# iterare, liste, dizionari

# Premessa
Questa esercitazione serve per autovalutarvi. E' molto importante cl
la svolgiate da parte vostra per farvi capire che esame e il corso
molto **pratico** ed è necessario che vi allenate continuamente a
casa. Vi consente di capire quale è il vostro livello
(autovalutazione). In base al livello potete decidere se correre ai
ripari (e iniziare a fare più pratica su concetti di base) oppure
passare a concetti più avanzati.

Di seguito vengono riportati diversi piccoli problemi che dovete
risolvere implementando piccole funzionalità. I codici non sono mai
molto lunghi da scrivere quindi se vi trovate a scrivere paginate d
codice c'e' qualcosa che non va. L'esercitazione è divisa
in sezione dove ogni sezione affronta (piu o meno) argomento di bas
A differenza della lezione, c'e' da fare uno sforzo per capire come
**Quindi cerchiamo di passare dalla teoria a farvi programmare nell
pratica perchè anche semplicemente copiare i comandi mentre io li
presento non vi insegna a programmare**

Gli esercizi sono (tendenzialmente) in ordine crescente di difficol
"""
None
```

In [3]:

```
"""
# Parte 1: Interi e float e stringhe
"""



# %%01_int_to_str
"""
### Es. 1 - Banale
Scrivere una funzione che prende in ingresso un intero e rende una
stringa dell'intero

| input | output |
|-------+--------|
|     4 | '4'    |
|     7 | '7'    |
|     3 | '3'    |
|    10 | '10'   |
|   109 | '109   |
"""


def int_to_str(i):
    pass
    # inserisci qui il tuo codice



evaluate(int_to_str)
show_tests(int_to_str)
```

```
> Valutiamo la funzione int_to_str
int_to_str is not implemented
> Mostriamo i test per int_to_str
==================================================
> 0)CHIAMATA: int_to_str(4,)
> 0)RISULTATO ATTESO: '4'

==================================================
> 1)CHIAMATA: int_to_str(7,)
> 1)RISULTATO ATTESO: '7'

==================================================
> 2)CHIAMATA: int_to_str(3,)
> 2)RISULTATO ATTESO: '3'

==================================================
> 3)CHIAMATA: int_to_str(10,)
> 3)RISULTATO ATTESO: '10'

==================================================
> 4)CHIAMATA: int_to_str(109,)
> 4)RISULTATO ATTESO: '109'
```

In [4]:

```
"""
### Es. 2 - Banale
Scrivere una funzione che prende in ingresso un intero e rende una
stringa dell'intero se l'intero è positivo oppure uguale a zero;
altrimenti rende la versione floating point dell'intero ma al contra

| input | output |
|-------+--------|
|     4 |    '4' |
|    -7 |    7.0 |
|     3 |    '3' |
|   -10 |   10.0 |
|  -109 |  109.0 |
"""


def int_to_str_flt(i):
    pass
    # inserisci qui il tuo codice


evaluate(int_to_str_flt)
show_tests(int_to_str_flt)
```

```
> Valutiamo la funzione int_to_str_flt
int_to_str_flt is not implemented
> Mostriamo i test per int_to_str_flt
==================================================
> 0)CHIAMATA: int_to_str_flt(4,)
> 0)RISULTATO ATTESO: '4'

==================================================
> 1)CHIAMATA: int_to_str_flt(-7,)
> 1)RISULTATO ATTESO: 7.0

==================================================
> 2)CHIAMATA: int_to_str_flt(3,)
> 2)RISULTATO ATTESO: '3'

==================================================
> 3)CHIAMATA: int_to_str_flt(-10,)
> 3)RISULTATO ATTESO: 10.0

==================================================
> 4)CHIAMATA: int_to_str_flt(-109,)
> 4)RISULTATO ATTESO: 109.0
```

In [5]:

```
"""
### Es. 3 - Facile
Scrivere una funzione che prende in ingresso un numero interi posit
e torni il numero di cifre del numero (considerando il numero in ba
10). Vedi tabella sottostante per esempi.

|  input | output |
|--------+--------|
|      4 |      1 |
|      7 |      1 |
|      4 |      1 |
|     10 |      2 |
|     23 |      2 |
|     45 |      2 |
|     99 |      2 |
|      0 |      1 |
|    100 |      3 |
|    222 |      3 |
|    999 |      3 |
| 999999 |      6 |
"""

from math import log10
def num_digits(intero):
    pass
    # inserisci qui il tuo codice



evaluate(num_digits)
show_tests(num_digits)
```

```
> Valutiamo la funzione num_digits
num_digits is not implemented
> Mostriamo i test per num_digits
==================================================
> 0)CHIAMATA: num_digits(5,)
> 0)RISULTATO ATTESO: 1

==================================================
> 1)CHIAMATA: num_digits(7,)
> 1)RISULTATO ATTESO: 1

==================================================
> 2)CHIAMATA: num_digits(0,)
> 2)RISULTATO ATTESO: 1

==================================================
> 3)CHIAMATA: num_digits(10,)
> 3)RISULTATO ATTESO: 2

==================================================
> 4)CHIAMATA: num_digits(22,)
> 4)RISULTATO ATTESO: 2

==================================================
> 5)CHIAMATA: num_digits(67,)
> 5)RISULTATO ATTESO: 2

==================================================
> 6)CHIAMATA: num_digits(99,)
> 6)RISULTATO ATTESO: 2

==================================================
> 7)CHIAMATA: num_digits(77,)
> 7)RISULTATO ATTESO: 2

==================================================
> 8)CHIAMATA: num_digits(1000,)
> 8)RISULTATO ATTESO: 4

==================================================
> 9)CHIAMATA: num_digits(10000,)
> 9)RISULTATO ATTESO: 5

==================================================
> 10)CHIAMATA: num_digits(999999,)
> 10)RISULTATO ATTESO: 6
```

In [6]:

```
"""
### Es. 4 - Banale
Modificare la funzione precedente in maniera tale che funzioni anch
con numeri negativi (se non funziona gia' come prima).  Nota da den
la funzione `num_digits_net` si puo richiamare la funzione vecchia
controllare se funziona
"""

def num_digits_neg(v):
    pass
    # inserisci qui il tuo codice


evaluate(num_digits_neg)
show_tests(num_digits_neg)
```

```
> Valutiamo la funzione num_digits_neg
num_digits_neg is not implemented
> Mostriamo i test per num_digits_neg
==================================================
> 0)CHIAMATA: num_digits_neg(5,)
> 0)RISULTATO ATTESO: 1

==================================================
> 1)CHIAMATA: num_digits_neg(-7,)
> 1)RISULTATO ATTESO: 1

==================================================
> 2)CHIAMATA: num_digits_neg(0,)
> 2)RISULTATO ATTESO: 1

==================================================
> 3)CHIAMATA: num_digits_neg(-10,)
> 3)RISULTATO ATTESO: 2

==================================================
> 4)CHIAMATA: num_digits_neg(-22,)
> 4)RISULTATO ATTESO: 2

==================================================
> 5)CHIAMATA: num_digits_neg(67,)
> 5)RISULTATO ATTESO: 2

==================================================
> 6)CHIAMATA: num_digits_neg(99,)
> 6)RISULTATO ATTESO: 2

==================================================
> 7)CHIAMATA: num_digits_neg(77,)
> 7)RISULTATO ATTESO: 2

==================================================
> 8)CHIAMATA: num_digits_neg(1000,)
> 8)RISULTATO ATTESO: 4

==================================================
> 9)CHIAMATA: num_digits_neg(10000,)
> 9)RISULTATO ATTESO: 5

==================================================
> 10)CHIAMATA: num_digits_neg(-999999,)
> 10)RISULTATO ATTESO: 6
```

In [7]:

```
"""
### Es. 5 - Banale
Implementare la stessa funzione ma questa volta l' input può essere
sia un intero anche negativo che una stringa Potete riusare le
funzionalità precedenti se volete

| input     | output |
|-----------+--------|
| '4'       |      1 |
| -7        |      1 |
| '0002'    |      1 |
| 10        |      2 |
| '0999999' |      6 |
"""


def num_digits_neg_str_mix(v):
    pass
    # inserisci qui il tuo codice


evaluate(num_digits_neg_str_mix)
show_tests(num_digits_neg_str_mix)
```

```
> Valutiamo la funzione num_digits_neg_str_mix
num_digits_neg_str_mix is not implemented
> Mostriamo i test per num_digits_neg_str_mix
==================================================
> 0)CHIAMATA: num_digits_neg_str_mix('5',)
> 0)RISULTATO ATTESO: 1

==================================================
> 1)CHIAMATA: num_digits_neg_str_mix('-07',)
> 1)RISULTATO ATTESO: 1

==================================================
> 2)CHIAMATA: num_digits_neg_str_mix(0,)
> 2)RISULTATO ATTESO: 1

==================================================
> 3)CHIAMATA: num_digits_neg_str_mix('-00010',)
> 3)RISULTATO ATTESO: 2

==================================================
> 4)CHIAMATA: num_digits_neg_str_mix(-22,)
> 4)RISULTATO ATTESO: 2

==================================================
> 5)CHIAMATA: num_digits_neg_str_mix('0000067',)
> 5)RISULTATO ATTESO: 2

==================================================
> 6)CHIAMATA: num_digits_neg_str_mix('99',)
> 6)RISULTATO ATTESO: 2

==================================================
> 7)CHIAMATA: num_digits_neg_str_mix('00077',)
> 7)RISULTATO ATTESO: 2

==================================================
> 8)CHIAMATA: num_digits_neg_str_mix(-1000,)
> 8)RISULTATO ATTESO: 4
```

In [8]:

```
"""
###Es. 6 - Facile
Sia data una stringa in ingresso alla funzione e un intero anche
negativo. La funzione deve ritornare il numero di volte che questo
intero è nella stringa.

Ad esempio dato gli input:

'p-1pp-1', -1

la funzione deve rendere 2 in quanto -1 è contenuto 2 volte nella s

| input          | output |
|----------------+--------|
| 'pippo', 1     |      0 |
| 'pippo1', 2    |      0 |
| 'pippo2', 2    |      1 |
| 'p1pp0', 0     |      1 |
| 'p1pp1', 1     |      2 |
| 'p-1pp-1', -1  |      2 |
"""


def check_int_str(stringa, intero):
    pass
    # inserisci qui il tuo codice


evaluate(check_int_str)
show_tests(check_int_str)
```

```
> Valutiamo la funzione check_int_str
check_int_str is not implemented
> Mostriamo i test per check_int_str
==================================================
> 0)CHIAMATA: check_int_str('pippo', 1)
> 0)RISULTATO ATTESO: 0

==================================================
> 1)CHIAMATA: check_int_str('pippo2', 2)
> 1)RISULTATO ATTESO: 1

==================================================
> 2)CHIAMATA: check_int_str('p1pp0', 0)
> 2)RISULTATO ATTESO: 1

==================================================
> 3)CHIAMATA: check_int_str('p1pp0', 0)
> 3)RISULTATO ATTESO: 1

==================================================
> 4)CHIAMATA: check_int_str('p1pp1', 1)
> 4)RISULTATO ATTESO: 2

==================================================
> 5)CHIAMATA: check_int_str('-1-1-1-1', -1)
> 5)RISULTATO ATTESO: 4

==================================================
> 6)CHIAMATA: check_int_str('sup3rcalifragilistic3ppalippa', 3)
> 6)RISULTATO ATTESO: 2

==================================================
> 7)CHIAMATA: check_int_str('2121211212121121', 2)
> 7)RISULTATO ATTESO: 7
```

In [9]:

```
"""
###Es. 7 - Facile/Medio
Scrivere una funzione che prende in ingresso una stringa S (di soli
numeri) e una stringa T che contiene sia numeri che caratteri. La
funzione renda una lista dove ogni elemento contiene il conteggio d
ogni **SINGOLA** cifra di S rispetto alla stringa T.
Il conteggio inserito nella lista è allineato alla lista in uscita.

Dati:

S = '1234'
T = 'p1p2p335o'

si deve rendere [1,1,2,0]

in quanto
- '1' è contenuto 1 volta in T
- '2' è contenuto 1 volta in T
- '3' è contenuto 2 volte in T.
- '4' è contenuto 0 volte in T.
"""


def check_S_in_T(S, T):
    pass
    # inserisci qui il tuo codice


evaluate(check_S_in_T)
show_tests(check_S_in_T)
```

```
> Valutiamo la funzione check_S_in_T
check_S_in_T is not implemented
> Mostriamo i test per check_S_in_T
==================================================
> 0)CHIAMATA: check_S_in_T('42130', 'dc6486c479e84c94efce')
> 0)RISULTATO ATTESO: [4, 0, 0, 0, 0]

==================================================
> 1)CHIAMATA: check_S_in_T('42130', '4bea7169ef7d4c80b6da')
> 1)RISULTATO ATTESO: [2, 0, 1, 0, 1]

==================================================
> 2)CHIAMATA: check_S_in_T('42130', '07d35d393fc7158e18b8')
> 2)RISULTATO ATTESO: [0, 0, 2, 3, 1]

==================================================
> 3)CHIAMATA: check_S_in_T('42130', 'd8f9979694329a71ceed')
> 3)RISULTATO ATTESO: [1, 1, 1, 1, 0]

==================================================
> 4)CHIAMATA: check_S_in_T('42130', 'ee86b4cde9f97afec197')
> 4)RISULTATO ATTESO: [1, 0, 1, 0, 0]

==================================================
> 5)CHIAMATA: check_S_in_T('42130', 'ad3b13c5d12b09e8a0a9')
> 5)RISULTATO ATTESO: [0, 1, 2, 2, 2]

==================================================
> 6)CHIAMATA: check_S_in_T('42130', '965f00f213ce06143a52')
> 6)RISULTATO ATTESO: [1, 2, 2, 2, 3]

==================================================
> 7)CHIAMATA: check_S_in_T('42130', '801f35bde2af0ad54972')
> 7)RISULTATO ATTESO: [1, 2, 1, 1, 2]

==================================================
> 8)CHIAMATA: check_S_in_T('42130', '769845d480b5043f545f')
> 8)RISULTATO ATTESO: [4, 0, 0, 1, 2]

==================================================
> 9)CHIAMATA: check_S_in_T('30216', 'a9b66a0353a6e966b6a0')
> 9)RISULTATO ATTESO: [2, 2, 0, 0, 6]
```

In [10]:

```
"""
### Es. 8  Medio
Data una stringa "query" in input e un'altra stringa "corpo", e'
necessario trovare tutte le volte che "query" è contenuta nella
stringa "corpo"

query = 'pippo'
corpo = 'pipppippopipipipipippppppippo'

output = 2

in quanto pippo è contenuta 2 volte in corpo infatti se marchiamo
PIPPO in corpo appare in:
corpo = 'pippPIPPOpipipipipippppPIPPO'
             1                    2

'abababababababa'
 === === === ===
   --- --- ---
'aba'


NB: NON SI PUO' USARE IL METODO count() delle stringhe
"""


def count_sub_string(query, corpo):
    pass
    # inserisci qui il tuo codice



evaluate(count_sub_string)
show_tests(count_sub_string)
```

```
> Valutiamo la funzione count_sub_string
count_sub_string is not implemented
> Mostriamo i test per count_sub_string
==================================================
> 0)CHIAMATA: count_sub_string('z', 'abcdefgabcda')
> 0)RISULTATO ATTESO: 0

==================================================
> 1)CHIAMATA: count_sub_string('a', 'abcdefgabcda')
> 1)RISULTATO ATTESO: 3

==================================================
> 2)CHIAMATA: count_sub_string('abc', 'abcdefgabcd')
> 2)RISULTATO ATTESO: 2

==================================================
> 3)CHIAMATA: count_sub_string('pippo', 'pipppippopipipipipipppppp
ippo')
> 3)RISULTATO ATTESO: 2

==================================================
> 4)CHIAMATA: count_sub_string('ape', 'aperitivoapeapeapeape')
> 4)RISULTATO ATTESO: 5

==================================================
> 5)CHIAMATA: count_sub_string('$', '$$$$$$$$ $')
> 5)RISULTATO ATTESO: 9

==================================================
> 6)CHIAMATA: count_sub_string('$$', '$$$$$$$$ $')
> 6)RISULTATO ATTESO: 7

==================================================
> 7)CHIAMATA: count_sub_string('pap', 'papapapapapapapapapap')
> 7)RISULTATO ATTESO: 10

==================================================
> 8)CHIAMATA: count_sub_string('xxxxxx', 'xx')
> 8)RISULTATO ATTESO: 0
```

In [11]:

```
"""
### Es. 09 Medio
Dato l'esercizio precedente, modifica la funzione al fine di
restituire una lista di tuple. La lista è cosi fatta.  Ciascuna
elemento della lista è una tupla. La tupla contiene gli indici di
inizio e terminazione della sotto stringhe `query` trovata nella
stringa `corpo`.

Se in ingresso abbiamo:

query = 'pippo'
corpo = 'pipppippopipipipipippppppippo'
         012345678

deve rendere:
output = [(4, 8), (24, 28)]

in quanto:
- corpo[4] vale 'p' (p iniziale di pippo)
- corpo[8] vale 'o' (o finale di pippo)

pippo si trova dall'indice 4 all'indice 8 compreso.

Inoltre si trova anche a:
- corpo[24] vale 'p' (p iniziale di pippo)
- corpo[28] vale 'o' (o finale di pippo)
"""


def count_sub_string_idx(query, corpo):
    pass
    # inserisci qui il tuo codice


evaluate(count_sub_string_idx)
show_tests(count_sub_string_idx)
```

```
> Valutiamo la funzione count_sub_string_idx
count_sub_string_idx is not implemented
> Mostriamo i test per count_sub_string_idx
==================================================
> 0)CHIAMATA: count_sub_string_idx('z', 'abcdefgabcda')
> 0)RISULTATO ATTESO: []

==================================================
> 1)CHIAMATA: count_sub_string_idx('a', 'abcdefgabcda')
> 1)RISULTATO ATTESO: [(0, 0), (7, 7), (11, 11)]

==================================================
> 2)CHIAMATA: count_sub_string_idx('abc', 'abcdefgabcd')
> 2)RISULTATO ATTESO: [(0, 2), (7, 9)]

==================================================
> 3)CHIAMATA: count_sub_string_idx('pippo', 'pipppippopipipipipipp
ppppippo')
> 3)RISULTATO ATTESO: [(4, 8), (24, 28)]

==================================================
> 4)CHIAMATA: count_sub_string_idx('ape', 'aperitivoapeapeapeape')
> 4)RISULTATO ATTESO: [(0, 2), (9, 11), (12, 14), (15, 17), (18, 2
0)]

==================================================
> 5)CHIAMATA: count_sub_string_idx('$', '$$$$$$$$ $')
> 5)RISULTATO ATTESO: [(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5,
5), (6, 6), (7, 7), (9, 9)]

==================================================
> 6)CHIAMATA: count_sub_string_idx('$$', '$$$$$$$$ $')
> 6)RISULTATO ATTESO: [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5,
6), (6, 7)]

==================================================
> 7)CHIAMATA: count_sub_string_idx('pap', 'papapapapapapapapapap')
> 7)RISULTATO ATTESO: [(0, 2), (2, 4), (4, 6), (6, 8), (8, 10), (1
0, 12), (12, 14), (14, 16), (16, 18), (18, 20)]

==================================================
> 8)CHIAMATA: count_sub_string_idx('xxxxxx', 'xx')
> 8)RISULTATO ATTESO: []
```

In [12]:

```
"""
### Es. 10 - Facile
Vengono date due stringhe in ingresso ed è necessario controllare se
la prima stringa `S` è stata scritta al rovescio rispetto alla strin
`T`.

T = 'pippo'
S = 'oppip'

allora S è stata scritta al rovescio.

Mentre

```python
T = 'pippo'
S = 'opppp'
```
non lo è.

Scrivere una funzione check_reverse(S, T) che prende in ingresso due
stringhe e ritorni True se sono a rovescio, altrimenti False.
"""

def check_reverse(S, T):
    pass
    # inserisci qui il tuo codice

evaluate(check_reverse)
show_tests(check_reverse)
```

```
> Valutiamo la funzione check_reverse
check_reverse is not implemented
> Mostriamo i test per check_reverse
==================================================
> 0)CHIAMATA: check_reverse('oppip', 'pippo')
> 0)RISULTATO ATTESO: True

==================================================
> 1)CHIAMATA: check_reverse('opppp', 'pippo')
> 1)RISULTATO ATTESO: False

==================================================
> 2)CHIAMATA: check_reverse('ppip', 'pippo')
> 2)RISULTATO ATTESO: False

==================================================
> 3)CHIAMATA: check_reverse('pippo', 'oppip')
> 3)RISULTATO ATTESO: True

==================================================
> 4)CHIAMATA: check_reverse('AzAz', 'zAzA')
> 4)RISULTATO ATTESO: True

==================================================
> 5)CHIAMATA: check_reverse('**', '**')
> 5)RISULTATO ATTESO: True

==================================================
> 6)CHIAMATA: check_reverse('oTTag', 'gatto')
> 6)RISULTATO ATTESO: False
```

In [13]:
```
"""
# Parte 2: Liste, Tuple, Ordinamento, Dizionari
"""
```

Out[13]:
```
'\n# Parte 2: Liste, Tuple, Ordinamento, Dizionari\n'
```

In [14]:
```
"""
###Es. 11 Banale
Data una lista in ingresso L ritornare una tupla che contiene
1. al primo elemento la somma di tutti i valori nella lista
2. al secondo elemento il massimo di tutti i valori
3. al terzo elemento il minimo
"""


def sum_max_min(L):
    pass
    # inserisci qui il tuo codice


evaluate(sum_max_min)
show_tests(sum_max_min)
```

```
> Valutiamo la funzione sum_max_min
sum_max_min is not implemented
> Mostriamo i test per sum_max_min
==================================================
> 0)CHIAMATA: sum_max_min([1, 2, 3],)
> 0)RISULTATO ATTESO: (6, 3, 1)

==================================================
> 1)CHIAMATA: sum_max_min([3, 2, 1],)
> 1)RISULTATO ATTESO: (6, 3, 1)

==================================================
> 2)CHIAMATA: sum_max_min([-1, -2, 3],)
> 2)RISULTATO ATTESO: (0, 3, -2)
```

In [15]:

```
"""
### Es. 12  Banale
Dalla funzione sottostante che prende in ingresso una
lista `L` sommare tutti i valori positivi

L = [10, 5, -5, -20, -30, -1, 20]

restituire 35.
"""


def sum_positive(L):
    pass
    # inserisci qui il tuo codice


evaluate(sum_positive)
show_tests(sum_positive)
```

```
> Valutiamo la funzione sum_positive
sum_positive is not implemented
> Mostriamo i test per sum_positive
==================================================
> 0)CHIAMATA: sum_positive([10, 5, -5, -20, -30, -1, 20],)
> 0)RISULTATO ATTESO: 35

==================================================
> 1)CHIAMATA: sum_positive([-1, 3, -4, 5, -7, 31],)
> 1)RISULTATO ATTESO: 39
```

In [16]:

```
"""
### Es. 13 Medio
Data la funzione sottostante che prende in ingresso una lista `L` è
necessario restituire una lista che contiene nella PRIMA parte della
lista i valori multipli di 2 della lista L, nella SECONDA parte del
lista valori multipli di 5 della lista L. L'ordine di inserimento
segue come sono messi i valori nella lista originale L.

L = [10, 5, -5, -20, -30, -1, 20]

rende

 [10, -20, -30, 20,     10, 5, -5, -20, -30, 20]
 #  multipli di 2    |  multipli di 5
"""


def list_multipli(L):
    pass
    # inserisci qui il tuo codice


evaluate(list_multipli)
show_tests(list_multipli)
```

```
> Valutiamo la funzione list_multipli
list_multipli is not implemented
> Mostriamo i test per list_multipli
==================================================
> 0)CHIAMATA: list_multipli([10, 5, -5, -20, -30, -1, 20],)
> 0)RISULTATO ATTESO: [10, -20, -30, 20, 10, 5, -5, -20, -30, 20]

==================================================
> 1)CHIAMATA: list_multipli([-11, -9, 9, -26, 15, -48, 29, -6, -4
2, 13, 49, 13, -31, 44, 41, 16, 23, -11, 1, -31],)
> 1)RISULTATO ATTESO: [-26, -48, -6, -42, 44, 16, 15]

==================================================
> 2)CHIAMATA: list_multipli([-22, -5, 31, -9, 37, -4, 41, 46, 24,
-35, 18, -28, 15, 34, -36, 10, -27, 42, -9, -2],)
> 2)RISULTATO ATTESO: [-22, -4, 46, 24, 18, -28, 34, -36, 10, 42,
-2, -5, -35, 15, 10]

==================================================
> 3)CHIAMATA: list_multipli([-28, 18, -36, -9, 4, -44, -44, 27, 47
, 19, 5, -35, -39, -30, 6, 33, 11, -16, -13, -19],)
> 3)RISULTATO ATTESO: [-28, 18, -36, 4, -44, -44, -30, 6, -16, 5,
-35, -30]

==================================================
> 4)CHIAMATA: list_multipli([13, 37, 26, 22, -40, 41, 35, 19, 46,
3, 46, 23, 3, -22, -46, -36, 26, -17, -50, 35],)
> 4)RISULTATO ATTESO: [26, 22, -40, 46, 46, -22, -46, -36, 26, -5
0, -40, 35, -50, 35]

==================================================
> 5)CHIAMATA: list_multipli([-50, 4, 23, -45, -4, -37, 30, 16, 49,
7, -13, -4, -15, -34, 46, 34, -7, 10, 38, -7],)
> 5)RISULTATO ATTESO: [-50, 4, -4, 30, 16, -4, -34, 46, 34, 10, 38
, -50, -45, 30, -15, 10]
```

In [17]:

```
"""
### Es. 14 - Facile
Data la funzione sottostante che prende in ingresso una lista L e'
necessario creare e rendere come output un'altra lista che contiene
all' elemento i-esimo la somma accumulata degli elementi della lista
precedenti all'elemento i-esimo, con questo ultimo compreso nella s

L = [-9, 42, 6]

rende

output = [-9, 33, 39]
"""


def cumulative_sum(L):
    pass
    # inserisci qui il tuo codice



evaluate(cumulative_sum)
show_tests(cumulative_sum)
```

```
> Valutiamo la funzione cumulative_sum
cumulative_sum is not implemented
> Mostriamo i test per cumulative_sum
==================================================
> 0)CHIAMATA: cumulative_sum([-9, 42, 6],)
> 0)RISULTATO ATTESO: [-9, 33, 39]

==================================================
> 1)CHIAMATA: cumulative_sum([0, 1, 3],)
> 1)RISULTATO ATTESO: [0, 1, 4]

==================================================
> 2)CHIAMATA: cumulative_sum([3, 2, 1],)
> 2)RISULTATO ATTESO: [3, 5, 6]

==================================================
> 3)CHIAMATA: cumulative_sum([-9, 9, -5, 5],)
> 3)RISULTATO ATTESO: [-9, 0, -5, 0]
```

In [18]:

```
"""
###Es. 15 - Medio/Difficile
Completare la funzione sottostante che data una lista in ingresso `L`
modifica la suddetta lista L in maniera tale da non avere più il
valore minimo e il massimo. La funzione modifica la lista L **senza
creare una altra lista (si modifica quella in input IN-PLACE)**.
La funzione rende il minimo e il massimo. La lista L NON puo' esser
vuota.

NB: si assume che il minimo e il massimo NON abbiano duplicati.
Ossia, il minimo e massimo sono unici.

L = [0, 5, 42, -1, 5, 3, 23]

modifica L in maniera tale che dopo sia:

L = [0, 5, 5, 3, 23]

e rende -1, 42.

Altro esempio:

L = [0, 4]

rende 0, 4 e L = []
"""

def get_list_except_min_max(L):
    pass
    # inserisci qui il tuo codice


evaluate(get_list_except_min_max)
show_tests(get_list_except_min_max)
```

```
> Valutiamo la funzione get_list_except_min_max
get_list_except_min_max is not implemented
> Mostriamo i test per get_list_except_min_max
==================================================
> 0)CHIAMATA: get_list_except_min_max([81, 23, 94, 48, 15, 20, 54,
33, 32, 42, 98, 66, 40, 57, 79, 6, 35, 77, 11, 39, 36, 84, 62, 26,
38, 19, 67, 99, 63, 51, 49, 90, 7, 58, 55, 9, 25, 73, 34],)
> 0)ARGOMENTI MODIFICATI ATTESI: ([81, 23, 94, 48, 15, 20, 54, 33,
32, 42, 98, 66, 40, 57, 79, 35, 77, 11, 39, 36, 84, 62, 26, 38, 19
, 67, 63, 51, 49, 90, 7, 58, 55, 9, 25, 73, 34],)
> 0)RISULTATO ATTESO: (6, 99)

==================================================
> 1)CHIAMATA: get_list_except_min_max([52, -16, -48, 2, -19],)
> 1)ARGOMENTI MODIFICATI ATTESI: ([-16, 2, -19],)
> 1)RISULTATO ATTESO: (-48, 52)

==================================================
> 2)CHIAMATA: get_list_except_min_max([94, -29, 79, -57],)
> 2)ARGOMENTI MODIFICATI ATTESI: ([-29, 79],)
> 2)RISULTATO ATTESO: (-57, 94)

==================================================
> 3)CHIAMATA: get_list_except_min_max([-44, 51, 24],)
> 3)ARGOMENTI MODIFICATI ATTESI: ([24],)
> 3)RISULTATO ATTESO: (-44, 51)

==================================================
> 4)CHIAMATA: get_list_except_min_max([98, 7, 41, 20],)
> 4)ARGOMENTI MODIFICATI ATTESI: ([41, 20],)
> 4)RISULTATO ATTESO: (7, 98)

==================================================
> 5)CHIAMATA: get_list_except_min_max([-38, 46, 80],)
> 5)ARGOMENTI MODIFICATI ATTESI: ([46],)
> 5)RISULTATO ATTESO: (-38, 80)

==================================================
> 6)CHIAMATA: get_list_except_min_max([-6, -80, -13, 22],)
> 6)ARGOMENTI MODIFICATI ATTESI: ([-6, -13],)
> 6)RISULTATO ATTESO: (-80, 22)

==================================================
> 7)CHIAMATA: get_list_except_min_max([0, 24],)
> 7)ARGOMENTI MODIFICATI ATTESI: ([],)
> 7)RISULTATO ATTESO: (0, 24)

==================================================
> 8)CHIAMATA: get_list_except_min_max([99, -100],)
> 8)ARGOMENTI MODIFICATI ATTESI: ([],)
> 8)RISULTATO ATTESO: (-100, 99)
```

In [19]:

```
"""
### Es. 16 - Medio/Difficile
Data una lista `L` in ingresso alla funzione modificare la lista
L in-place (SENZA CREARNE UNA COPIA) in maniera da eliminare il min
 e il massimo e restituire il numero totale di elementi eliminati.

1) La lista in ingresso puo' essere vuota.
2) Il massimo e minimo valore possono NON essere unici.
3) il minimo e il massimo possono coincidere.

1)
L = [5, 5, 5, 5, 5]

L si modifica in []
rende 5.

2)
L = [-11, 13, -11,  13, -11]

L si modifica in []
rende 5

3)
L = [-5, 2, -5, 10, -11, -11, 10, 0, -11, 2]

L si modifica in [-5, 2, -5, 0, 2]
rende 5

4)
L = []

si modifica L in []
rende 0
"""

def get_list_except_min_max_general(L):
    pass
    # inserisci qui il tuo codice


evaluate(get_list_except_min_max_general)
show_tests(get_list_except_min_max_general)
```

```
> Valutiamo la funzione get_list_except_min_max_general
get_list_except_min_max_general is not implemented
> Mostriamo i test per get_list_except_min_max_general
==================================================
> 0)CHIAMATA: get_list_except_min_max_general([5, 5, 5, 5, 5],)
> 0)ARGOMENTI MODIFICATI ATTESI: ([],)
> 0)RISULTATO ATTESO: 5

==================================================
> 1)CHIAMATA: get_list_except_min_max_general([-11, 13, -11, 13,
-11],)
> 1)ARGOMENTI MODIFICATI ATTESI: ([],)
> 1)RISULTATO ATTESO: 5

==================================================
> 2)CHIAMATA: get_list_except_min_max_general([-5, 2, -5, 10, -11,
-11, 10, 0, -11, 2],)
> 2)ARGOMENTI MODIFICATI ATTESI: ([-5, 2, -5, 0, 2],)
> 2)RISULTATO ATTESO: 5

==================================================
> 3)CHIAMATA: get_list_except_min_max_general([],)
> 3)ARGOMENTI MODIFICATI ATTESI: ([],)
> 3)RISULTATO ATTESO: 0
```

In [20]:
```
### Es. 17 - Difficile
'''
Sono date in ingresso tre liste di lunghezza uguale di nome
 `L`, `src`, e  `dst`.
E' necessario creare una nuova lista `out` in uscita con le seguent
proprietà:
- dato l'elemento i-esimo nella lista `L`,
    la nuova lista deve contenere il valore preso alla posizione i-
    delle lista `src`. Questo valore deve essere scritto nella posi
    dall'elemento i-esimo della lista `dst`.

- Nota bene: i valori di dst fanno si che alcuni elementi della lis
non vengano scritti. In questo caso il loro valore intero deve esse

L =   [4, 2, 1, -5]
src = [1, 2, 3,  0]     # indici di quali valori prendere da L
dst = [2, 4, 6,  8]     # indici di dove mettere i valori nel risul

deve rendere:

out = [None, None, 2, None, 1, None, -5, None, 4]

'''

def reshape_by_index(L, src, dst):
    pass
    # inserisci qui il tuo codice



evaluate(reshape_by_index)
show_tests(reshape_by_index)
```

```
> Valutiamo la funzione reshape_by_index
reshape_by_index is not implemented
> Mostriamo i test per reshape_by_index
==================================================
> 0)CHIAMATA: reshape_by_index([4, 2, 1, -5], [1, 2, 3, 0], [2, 4,
6, 8])
> 0)RISULTATO ATTESO: [None, None, 2, None, 1, None, -5, None, 4]

==================================================
> 1)CHIAMATA: reshape_by_index([4, 2, 1, -5], [3, 2, 1, 0], [0, 1,
2, 3])
> 1)RISULTATO ATTESO: [-5, 1, 2, 4]

==================================================
> 2)CHIAMATA: reshape_by_index([0, 1, 7, -5], [1, 0, 2, 3], [3, 8,
20, 21])
> 2)RISULTATO ATTESO: [None, None, None, 1, None, None, None, None
, 0, None, None, None, None, None, None, None, None, None, None, N
one, 7, -5]
```

In [21]:

```
"""
###Es. 18 Medio
E' fornita in ingresso una lista L di tuple ed è necessario tornare
come valore di ritorno una tupla di liste. Le tuple possono avere
lunghezza variabile. E' necessario tornare come output una tupla di
liste. Ciascuna lista dentro la tupla di ritorno contiene i valori a
contrario rispetto alle tuple iniziali.  Le liste nella tupla da re
contengono valori il cui prodotto e' multiplo di 2.

L = [(2, -2, 4 ), (2, -2, 1 ), (0, 5, 4 ), (1,3), (1, 1, 5, 11)]

rende

out = ([4, -2, 2], [1, -2, 2], [4, 5, 0])
"""

def list_to_tuple(L):
    pass
    # inserisci qui il tuo codice

evaluate(list_to_tuple)
show_tests(list_to_tuple)
```

```
> Valutiamo la funzione list_to_tuple
list_to_tuple is not implemented
> Mostriamo i test per list_to_tuple
==================================================
> 0)CHIAMATA: list_to_tuple([(2, -2, 4), (2, -2, 1), (0, 5, 4),
(1, 3), (1, 1, 5, 11)],)
> 0)RISULTATO ATTESO: ([4, -2, 2], [1, -2, 2], [4, 5, 0])

==================================================
> 1)CHIAMATA: list_to_tuple([(1, 1), (1, 5)],)
> 1)RISULTATO ATTESO: ()

==================================================
> 2)CHIAMATA: list_to_tuple([(1, 1, 2), (1, 0), (1, 2, 4, 8, 16, 3
2)],)
> 2)RISULTATO ATTESO: ([2, 1, 1], [0, 1], [32, 16, 8, 4, 2, 1])
```

In [22]:
```
"""
###Es. 19 - Medio/Difficile
Dati in ingresso una tupla T di stringhe restituire una tupla S che
la tupla suddetta T in ordine crescente in base alla lunghezza dell
parole e in ordine decrescente in base all'ordine lessicografico.

T = ('aaaaa', 'aaa', 'zzzzz', 'zzz')

S = ('zzz', 'aaa', 'zzzzz', 'aaaaa')
"""

def sort_by_str(T):
    pass
    # inserisci qui il tuo codice


evaluate(sort_by_str)
show_tests(sort_by_str)
```

```
> Valutiamo la funzione sort_by_str
sort_by_str is not implemented
> Mostriamo i test per sort_by_str
==================================================
> 0)CHIAMATA: sort_by_str(('aaaaa', 'aaa', 'zzzzz', 'zzz'),)
> 0)RISULTATO ATTESO: ('zzz', 'aaa', 'zzzzz', 'aaaaa')

==================================================
> 1)CHIAMATA: sort_by_str(('pipppo', 'pipppo', 'pipo', 'p', 'q', '
r', 'A', 'a', 'ab', 'AA', 'bb', 'p1ppp0'),)
> 1)RISULTATO ATTESO: ('r', 'q', 'p', 'a', 'A', 'bb', 'ab', 'AA',
'pipo', 'pipppo', 'pipppo', 'p1ppp0')
```

In [23]:
```
"""
###Es. 20  - Difficile

Data in ingresso una lista con i nomi dei mesi restituire un'altra
lista ordinata in base all'ordine dei mesi nel calendario.

NB: i nomi dei mesi possono avere l'iniziale sia in UPPER case che
lower case. Quindi 'Gennaio' e' valido cosi come 'gennaio'.

a)
L = ['Settembre', 'luglio', 'gennaio', 'Maggio']

deve rendere

['gennaio', 'Maggio', 'luglio', 'Settembre']

in quanto gennaio viene prima di tutti, poi segue
Maggio poi luglio e infine Settembre secondo il calendario.
"""

def sort_by_month(L):
    pass
    # inserisci qui il tuo codice


evaluate(sort_by_month)
show_tests(sort_by_month)
```

```
> Valutiamo la funzione sort_by_month
sort_by_month is not implemented
> Mostriamo i test per sort_by_month
==================================================
> 0)CHIAMATA: sort_by_month(['Settembre', 'luglio', 'gennaio', 'Ma
ggio'],)
> 0)RISULTATO ATTESO: ['gennaio', 'Maggio', 'luglio', 'Settembre']

==================================================
> 1)CHIAMATA: sort_by_month(['Settembre', 'Dicembre', 'agosto', 'G
iugno', 'luglio', 'gennaio', 'Maggio'],)
> 1)RISULTATO ATTESO: ['gennaio', 'Maggio', 'Giugno', 'luglio', 'a
gosto', 'Settembre', 'Dicembre']

==================================================
> 2)CHIAMATA: sort_by_month(['novembre', 'febbraio'],)
> 2)RISULTATO ATTESO: ['febbraio', 'novembre']

==================================================
> 3)CHIAMATA: sort_by_month(['dicembre', 'novembre', 'ottobre', 's
ettembre', 'agosto', 'luglio', 'giugno', 'maggio', 'aprile', 'marz
o', 'febbraio', 'gennaio'],)
> 3)RISULTATO ATTESO: ['gennaio', 'febbraio', 'marzo', 'aprile', '
maggio', 'giugno', 'luglio', 'agosto', 'settembre', 'ottobre', 'no
vembre', 'dicembre']
```

In [24]:

```
"""
###Es. 21 Difficile

Data in ingresso una lista di interi `L` è necessario generare una s
che visualizza a video il conteggio di ogni singolo intero sottoforr
ISTOGRAMMA.

L = [1, 1, 1, 1, 4, 4, 4, 5, 5, 10, 10]

e' necessario che la funzione torni una stringa uguale a

'1\t****\n2\t\n3\t\n4\t***\n5\t**\n6\t\n7\t\n8\t\n9\t\n10\t**\n'

che se stampata mostra l'ISTOGRAMMA degli interi dove:
- il numero di volte (frequenza) che in intero appare in L e' rappr
di asterischi uguali alla frequenza.
- ISTOGRAMMA ha su ciascuna riga il seguente formato:
   valore \t  * per quante volte e' presente e infine \n (accapo)"
- se un numero non e' presente non va stampato nessun asterisco.

Ad esempio la stringa in uscita dalla funzione int_to_hist quando p
ingressso L e':



1   ****
2
3
4   ***
5   **
6
7
8
9
10  **


"""

def int_to_hist(values):
    pass
    # inserisci qui il tuo codice


evaluate(int_to_hist)
show_tests(int_to_hist)
```

```
> Valutiamo la funzione int_to_hist
int_to_hist is not implemented
> Mostriamo i test per int_to_hist
==================================================
> 0)CHIAMATA: int_to_hist([1, 1, 1, 1, 4, 4, 4, 5, 5, 10, 10],)
> 0)RISULTATO ATTESO: '1\t****\n2\t\n3\t\n4\t***\n5\t**\n6\t\n7\t
\n8\t\n9\t\n10\t**\n'

==================================================
> 1)CHIAMATA: int_to_hist([4, 3, -1, -3, 0, -1, 3, -2, 0, 1, 4, 0,
-2, 3, 1, -3, 5, 5, 3, 4],)
> 1)RISULTATO ATTESO: '-3\t**\n-2\t**\n-1\t**\n0\t***\n1\t**\n2\t
\n3\t****\n4\t***\n5\t**\n'

==================================================
> 2)CHIAMATA: int_to_hist([6, 4, -1, -4, 0, -2, 5, -3, 0, 1, 7, 0,
-4, 4, 2, -4, 7, 8, 5, 7, -3, 4, 7, 3, 0, -7, -1, 2, 7, 8, 0, 6,
-4, 5, 1, -8, 4, -2, 6, 3, -8, 0, 6, -4, -3, 6, -5, 1, -4, 8, 5,
-1, -7, -3, 0, 7, -7, 1, 4, 1, 5, 1, 8, 2, 1, -1, 2, -2, 1, -4,
-5, -5, 2, 3, 0, -7, 4, 6, 7, 6, 7, 7, 1, -2, 3, -4, 5, 6, 7, 2, 8
, 1, -1, 3, 8, 7, 5, -7, 2, 0],)
> 2)RISULTATO ATTESO: '-8\t**\n-7\t*****\n-6\t\n-5\t***\n-4\t*****
***\n-3\t****\n-2\t****\n-1\t*****\n0\t*********\n1\t**********\n2
\t*******\n3\t*****\n4\t******\n5\t*******\n6\t********\n7\t******
*****\n8\t******\n'
```

In [25]:

```
"""
###Es. 22

Data in ingresso due stringhe S e T la funzione deve controllare se
due stringhe sono una l'anagramma dell'altra. Ritorna True se lo so
altrimenti torna False.

Data una strinag S, un anagramma si crea permutando (spostando di p
 i caratteri della stringa S ma senza aggiungere ne rimuovere carat

S = 'roma'
T = 'amor'

la coppia e' un anagramma perche' 'roma' puo' essere generata da
'amor' semplicemente spostando alcuni caratteri.

S = 'a gentleman'
T = 'elegant man'

anche questa coppia lo e'.

S = 'fiore'
T = 'eroii'

NON lo e'.
"""

def anagramma(S : str, T: str) -> bool:
    pass
    # inserisci qui il tuo codice


evaluate(anagramma)
show_tests(anagramma)
```

```
> Valutiamo la funzione anagramma
anagramma is not implemented
> Mostriamo i test per anagramma
==================================================
> 0)CHIAMATA: anagramma('roma', 'amor')
> 0)RISULTATO ATTESO: True

==================================================
> 1)CHIAMATA: anagramma('elegant man', 'a gentleman')
> 1)RISULTATO ATTESO: True

==================================================
> 2)CHIAMATA: anagramma('fiore', 'eroii')
> 2)RISULTATO ATTESO: False
```