

# LABORATORIO DI PYTHON

## STRINGHE E TUPLE: I COSTRUTTI IMMUTABILI DI PYTHON

---

22 Marzo 2019

**CORREZIONE ESERCIZI PER OGGI**

Scrivere una funzione che prende tre valori **float**  $a$ ,  $b$ ,  $c$  come parametri e che restituisca la/le soluzione/ i dell'equazione  $ax^2 + bx + c = 0$ , oppure stampi gli opportuni messaggi nei casi in cui l'equazione sia impossibile o indeterminata e restituisca **None** in tali casi (fare attenzione a considerarli tutti!).

## ESERCIZIO 1 PER CASA

```
1 import math
2 def equazione_sec_grado(a,b,c):
3     if a==0 and b==0 and c==0:
4         print("Equazione indet.: a, b, c nulli")
5         return None
6     elif a==0 and b==0:
7         print("Equazione impossibile: a e b nulli")
8         return None
9     elif a==0:
10        return -c/b
11    else:
12        delta = b**2-4*a*c
13        if delta < 0:
14            print("Equazione impossibile: delta < 0")
15            return None
16        else:
17            radice_delta=math.sqrt(delta)
18            x1 = (-b-radice_delta)/(2*a)
19            x2 = (-b+radice_delta)/(2*a)
20            return x1, x2
```

Scrivere una funzione che, presi tre valori come parametri, li stampi in ordine decrescente. Nello stesso file, scrivere un test per la funzione che chieda i 3 valori come **input** e li legga come **float**.

```
1 def decrescente(a,b,c):
2     if a > b:
3         if b > c:
4             print(a, b, c)
5         elif a > c:
6             print(a, c, b)
7         else:
8             print(c, a, b)
9     else: # a <= b
10        if c > b:
11            print(c, b, a)
12        elif c > a:
13            print(b, c, a)
14        else:
15            print(b, a, c)
16
```

```
17 print("Programma che ordina 3 numeri float ")
18 f1 = float(input("Inserisci il primo numero: "))
19 f2 = float(input("Inserisci il secondo numero: "))
20 f3 = float(input("Inserisci il terzo numero: "))
21 print("I tre numeri in ordine decrescente sono: ")
22 decrescente(f1, f2, f3)
```

La tabella seguente riporta in euro le tariffe per il noleggio di uno scooter.

Scooter	24 ore	2 giorni	3 giorni	4 giorni	Ogni giorno extra
Euro	45,00	80,00	120,00	160,00	40,00

Scrivere una funzione che prenda come parametro il numero di giorni di noleggio e ne calcoli il costo totale. La funzione restituisce sempre un float (se `giorno < 1`, restituisce `0.0`). Nello stesso file, scrivere un test per la funzione che chieda in input il numero di giorni e stampi il costo totale di noleggio.



## ESERCIZIO 3 PER CASA

```
1 def noleggio(giorni):
2     if giorni < 1:
3         return 0.0
4     if giorni == 1:
5         return 45.00
6     if giorni == 2:
7         return 80.00
8     if giorni == 3:
9         return 120.00
10    if giorni == 4:
11        return 160.00
12    if giorni > 4:
13        return 160.00+40.00*(giorni-4)
14
15 g = int(input("Inserisci il numero di giorni: "))
16 costo = noleggio(g)
17 print("Il costo per", g, "giorni e' di", costo, "euro")
```

## ESERCIZIO 4 PER CASA

Dato l'anno, calcolare giorno e mese della Pasqua. La funzione prende come parametro  $x$  (l'anno)

	Valore..	...da dividere per	risultati utili	
			Quoziente intero	Resto
1	$x$	100	$b$	$c$
2	$5b + c$	19	-	$a$
3	$3(b + 25)$	4	$r$	$s$
4	$8(b + 11)$	25	$t$	-
5	$19a + r - t$	30	-	$h$
6	$a + 11h$	319	$g$	-
7	$60(5 - s) + c$	4	$j$	$k$
8	$2j - k - h + g$	7	-	$m$
9	$h - g + m + 110$	30	$n$ (mese)	$q$
10	$q + 5 - n$	32	-	$p$ (giorno)

e restituisce  $p$  e  $n$ , che rappresentano giorno e mese della Pasqua nell'anno  $x$ .

## ESERCIZIO 4 PER CASA

```
1 def pasqua(x):
2     b = x // 100
3     c = x % 100
4     due = 5*b+c
5     a = due % 19
6     tre = 3*(b+25)
7     r = tre // 4
8     s = tre % 4
9     quattro = 8*(b+11)
10    t = quattro // 25
11    cinque = 19*a+r-t
12    h = cinque % 30
13    sei = a+11*h
14    g = sei // 319
15    sette = 60*(5-s)+c
16    j = sette // 4
17    k = sette % 4
```

```
18    otto = 2*j-k-h+g
19    m = otto % 7
20    nove = h-g+m+110
21    n = nove // 30
22    q = nove % 30
23    dieci = q+5-n
24    p = dieci % 32
25    return p, n
```

# SEQUENZE

- **Immutabili:** che non possono essere modificate
  - *stringhe* (`str` ' '): sequenze di caratteri
  - *tuple* (`tuple` ( )): sequenze di valori separati da virgole
  - *range*: intervalli
- **Mutabili:** che possono essere modificate una volta create
  - *liste*
  - *dizionari*
  - ...

- Uno, due, tre (doppi) apici?
  - Python usa gli apici singoli `'Andrea'`
  - All'interno di essi posso usare le virgolette:  
`'Andrea disse "Ciao"'`
  - Oppure posso usare le virgolette `"Andrea"`
  - E al loro interno posso usare i singoli apici  
`"Andrea perse l'amo"`
  - Oppure posso usare tre singoli apici: `'''Andrea'''`
  - E al loro interno usare ciò che voglio:  
`'''Andrea disse: "l'amo"'''`
  - Oppure usare tre virgolette `"""Andrea"""`
  - E al loro interno usare ciò che voglio e anche andare a capo:  
`"""Andrea disse: "t'amo"  
Lei rispose "non abbocco"  
"""`

- Abbiamo già visto +, \*
- Possiamo usare gli operatori di confronto (>, <, ==, ...)
- Le stringhe sono sequenze **immutabili**. `s[0]='F'`

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'str' object does not support  
item assignment

Operatore di selezione di singoli caratteri: [ ]

Indici dalla fine: -6 -5 -4 -3 -2 -1

Indici dall'inizio: 0 1 2 3 4 5

```
nome =      +---+---+---+---+---+---+
            | A | n | d | r | e | a |
            +---+---+---+---+---+---+
```



Operatore di *slicing* (selezione di sottostringhe): `[ : ]`

Ha la forma `[inizio:fine]` in cui `inizio` è compreso, `fine` è escluso.

Se `inizio` non è indicato, si parte dall'inizio della stringa.

Se `fine` non è indicato, si arriva fino alla fine della stringa.

Indici dalla fine:   -6   -5   -4   -3   -2   -1

Indici dall'inizio:   0   1   2   3   4   5

```
nome =      +---+---+---+---+---+---+
            | A | n | d | r | e | a |
            +---+---+---+---+---+---+
```

Operatore di *slicing* esteso : `[::]`

Ha la forma `[inizio:fine:passo]`.

Se **passo** è positivo, si va da **inizio** compreso a **fine** escluso, prendendo un elemento ogni **passo**.

Se **passo** è negativo, si va da **fine** compreso a **inizio** escluso (scorrendo dunque la stringa al contrario), prendendo un elemento ogni **passo**.

Indici dalla fine:   -6   -5   -4   -3   -2   -1

Indici dall'inizio:   0   1   2   3   4   5

```

               +---+---+---+---+---+---+
nome =        | A | n | d | r | e | a |
               +---+---+---+---+---+---+
```

- Sequenze di valori (separate da virgole):  
`dati = 'Anna', 'Pannocchia', 'F', 20`
- Le parentesi sono opzionali, ma di solito si usano  

```
>>> dati  
( 'Anna', 'Pannocchia', 'F', 20 )
```
- Le virgole invece sono indispensabili:
  - `("Michael",)` è una tupla (con un solo elemento)
  - mentre `"Michael"` è una stringa!
  - ... o quasi: `()` è una tupla vuota
- Posso usare l'operatore `+` per concatenare tuple.
- Sulle tuple posso usare selezione `[]` e slicing `[:]`, `[::]`
  - Quanto vale `dati[2]`?
  - e `dati[:2]`?

- Posso “impacchettare” una tupla in una singola variabile:  
`dati = ('Anna', 'Pannocchia', 'F', 20)`
- oppure “spacchettare” una tupla in un numero congruo di variabili: `(nome, cognome, sesso, eta) = dati`

- Vi ricordate...?

```
1 def secondiInOreMinSec(secondiTotali):  
2     ore = secondiTotali // 3600  
3     secondiRimanenti = secondiTotali % 3600  
4     minuti = secondiRimanenti // 60  
5     secondi = secondiRimanenti % 60  
6     return ore, minuti, secondi
```

- Restituisce 3 valori: si tratta di una tupla
- `a = secondiInOreMinSec(62)`  
vs.  
`h, m, s = secondiInOreMinSec(62)`

```
1  for <var> in <seq>:  
2      <istruzioni>
```

- **seq**: una sequenza (es. stringa, tupla...)
- **var**: indica il nome della variabile che assume, uno alla volta, in sequenza, tutti i valori presenti in **seq**
- **istruzioni**: sono l'insieme di istruzioni che vengono eseguite **per ogni valore** contenuto nella sequenza

- Cosa stampa questo codice?

```
1 for dato in dati: #dati: la tupla di prima  
2     print(dato)
```

- E questo?

```
1 for lettera in "Andrea":  
2     print(lettera)
```

# ESERCIZI



Scrivere una funzione che prende come parametro una tupla numerica, chiede all'utente di inserire un valore, e restituisce una nuova tupla in cui l'elemento inserito ha sostituito l'elemento di testa della tupla originale. Non serve il **for**.

NB: eval

```
1 def cambia_testa(t):  
2     h = eval(input("Inserisci il nuovo valore di testa: "))  
3     return (h,)+t[1:]
```

Scrivere una funzione che prende come parametro una stringa e la restituisce invertita. Usare il for sulla stringa, non usare l'operatore di slicing. (Dopo: provare a risolverlo anche con l'operatore di slicing esteso).

```
1 def inverti(s):  
2     s_invert = ''  
3     for c in s:  
4         s_invert = c+s_invert  
5     return s_invert
```

```
1 def inverti(s):  
2     s_invert = ''  
3     for c in s:  
4         s_invert = c+s_invert  
5     return s_invert
```

Con l'operatore di slicing:

```
1 def invertiSlice(s):  
2     return s[::-1]
```

Posso usare `in` e `not in` per sapere se un elemento appartiene (o non appartiene) a una sequenza. Qual è il valore di:

- `'a' in 'Andrea'?`
- `3 in (2, 4, 6, 8)?`
- `'p' not in "aeiouAEIOU"?`

NB: per le stringhe, `in` verifica anche le sottostringhe (es. `'nd' in 'Andrea'` è `True`). Non vale per le tuple. Per approfondimenti: <https://docs.python.org/3.4/library/string.html>

Scrivere una funzione che prende una stringa come parametro e rimuove tutti gli spazi bianchi e i segni di interpunzione, e ritorna la nuova stringa così ripulita.

```
1 import string
2 def ripulisci(stringa):
3     stringa_pulita = ''
4     for c in stringa:
5         if c not in string.punctuation and c not in string.
           whitespace:
6             stringa_pulita = stringa_pulita+c
7     return stringa_pulita
```



Scrivere una funzione che prende come parametri una tupla  $t$  e un numero  $n$ , e restituisce una nuova tupla in cui ogni elemento è stato moltiplicato per  $n$ . Esempio:  $t = (4, 2, 5, 3)$ ,  $n = 2$ , la funzione restituisce  $(8, 4, 10, 6)$ .

```
1 def moltiplica_tupla(t, n):
2     t_out = () #la nuova tupla, per ora vuota
3     for e in t:
4         """nella nuova tupla, inserisco come nuovo elemento ogni
5            elemento della tupla presa come parametro, moltiplicato
6            per n"""
7         t_out = t_out + (e*n,)
8     return t_out
```

Chiedere all'utente in input: una stringa, un carattere. Stampare il numero di volte in cui il carattere compare all'interno della stringa.

```
1 stringa = input("Inserisci una stringa: ")
2 carattere = input("Inserisci un carattere: ")
3 n = 0
4 for c in stringa:
5     if c == carattere:
6         n = n + 1
7 print("Il carattere", carattere, "compare", n, "volte")
```

Scrivere una funzione che prende come parametro una tupla di numeri, che rappresentano voti, e ne calcola la media aritmetica. Provare il programma chiedendo all'utente di inserire una tupla (valutando l'input con `eval`)

## ESERCIZIO

```
1 def media(voti):
2     """Calcola la media dei voti presenti nella tupla passata
3     come parametro"""
4     somma = 0
5     for voto in voti:
6         somma = somma + voto
7     if len(voti)>0: #evito divisioni per 0
8         media = somma/len(voti)
9         return media
10    else:
11        print("Nessun voto inserito")
12        return None
13 miei_voti = eval(input("Inserisci i voti separati da virgole: "))
14 print("Media:", media(miei_voti))
```

## ESERCIZI PER CASA

1. Scrivere una funzione che prende come parametri due tuple **t1** e **t2** e un numero intero **n**. La prima tupla è da modificare aggiungendo tutti i valori della seconda tupla alla posizione **n**. La funzione restituisce la tupla aggiornata. Es. se **t1** = (1, 3, 2), **t2** = (9, 7), **n** = 1 restituirà (1, 9, 7, 3, 2).
2. Scrivere un programma che prenda due stringhe in input le stampi in ordine alfabetico, oppure ne stampi solo una delle due se sono identiche.
3. Scrivere una funzione che stampi a video
  - le *consonanti* della stringa passata come parametro
  - il *numero di vocali* della stessa stringa

La funzione **non stampa e non conteggia** spazi, punteggiatura, simboli vari...