

LABORATORIO DI PYTHON

ITERAZIONE IN PYTHON

22 Marzo 2019

CORREZIONE ESERCIZI PER OGGI

Scrivere una funzione che restituisce **True** se una stringa passata come parametro è palindroma (senza considerare gli spazi, la punteggiatura ma facendo distinzione tra maiuscole e miniscole: es “Anna” **non** è palindroma, “ANGOLO BAR, A BOLOGNA!” sì). Usare il ciclo **for**.

ESERCIZIO 1 PER CASA I

```
1 import string
2
3 def normalizza(stringa):
4     """Funzione che restituisce una nuova stringa, ripulita da:
5     spazi e punteggiatura"""
6     stringa_pulita = ''
7     for c in stringa:
8         if c not in string.punctuation and c not in string.
9         whitespace:
10             stringa_pulita = stringa_pulita+c
11     return stringa_pulita
12
13 def palindroma(s):
14     """Funzione che stabilisce se s e' palindroma"""
15     s = normalizza(s)
16     return s == s[::-1]
```

ESERCIZIO 1 PER CASA II

```
15
16 def palindroma2(s):
17     """Funzione che stabilisce se s e' palindroma (senza usare
18     operatore di slicing)"""
19     s = normalizza(s)
20     if len(s) < 2:
21         return True #una stringa lunga 0 o 1 e' palindroma per
22         def
23         for i in range(len(s)//2): #scorro meta' stringa (vado da 0
24             alla meta' arrotondata -1)
25             if s[i] != s[-1-i]: #controllo i caratteri
26                 corrispondenti
27                 return False #mi basta un carattere non uguale per
28                 terminare
29     return True #se sono arrivato fin qui, e' palindroma
```

ESERCIZIO 2 PER CASA

Scrivere una funzione che presi due numeri come parametri della funzione, restituisca il Massimo Comun Divisore (MCD) tra i due numeri. Usare il ciclo **for**.

ESERCIZIO 2 PER CASA

Scrivere una funzione che presi due numeri come parametri della funzione, restituisca il Massimo Comun Divisore (MCD) tra i due numeri. Usare il ciclo **for**.

```
1 def MCD(a,b):
2     if b == 0:
3         return a
4     if a == 0:
5         return b
6     if a<b:
7         minore = a
8     else:
9         minore = b
10    #scorro tutti i numeri dal minore fino a 1
11    for i in range(minore, 0, -1):
12        if (a % i == 0) and (b % i == 0): #se entrambi sono
            divisibili per i...
13            return i #... allora i e' il loro massimo comun
            divisore
```

Scrivere una funzione che preso come parametro un numero n ($n > 2$) restituisce il più piccolo c ($c \geq 2$) tale che $\text{MCD}(n, c) == 1$. Usare il ciclo **for** e la funzione dell'esercizio 2.

Scrivere una funzione che preso come parametro un numero n ($n > 2$) restituisce il più piccolo c ($c \geq 2$) tale che $\text{MCD}(n, c) == 1$. Usare il ciclo **for** e la funzione dell'esercizio 2.

```
1 import Es2
2
3 def coprime(n):
4     if n > 2:
5         for c in range(2, n):
6             if Es2.MCD(n, c) == 1:
7                 return c
```

Scrivere una funzione con un parametro n . Se $n \geq 7$, disegna una “stellina” a n punte. Si tratta di una generalizzazione della versione a 5 punte. NB! Il numero dei lati da “saltare” può essere scelto come un numero **coprime con n** (e dunque si può usare la funzione dell'esercizio 3).

ESERCIZIO 4 PER CASA

Scrivere una funzione con un parametro n . Se $n \geq 7$, disegna una “stellina” a n punte. Si tratta di una generalizzazione della versione a 5 punte. NB! Il numero dei lati da “saltare” può essere scelto come un numero **coprimo con n** (e dunque si può usare la funzione dell’esercizio 3).

```
1 import turtle , Es3
2
3 def stellina(n):
4     if n >= 7:
5         for i in range(n): #ho n lati
6             turtle.forward(100)
7             turtle.right((360/n)*Es3.coprime(n))
```

ITEREAZIONE CON IL WHILE

Costrutto `while`

```
1 while condizione :  
2     istruzioni interne al while  
3     ...  
4     istruzioni interne al while  
5 istruzioni successive al while
```

- La **condizione** (espressione booleana) viene valutata.
- **Solo se** l'espressione booleana vale **True** allora si eseguono le istruzioni all'interno del costrutto di iterazione **while** (notare l'indentazione).
- Finite le istruzioni all'interno del **while**, **si torna nuovamente a testare la condizione.**
- Se è ancora vera, si eseguono nuovamente le istruzioni all'interno del **while**
- Si prosegue così finché la condizione diviene falsa.
- Se la condizione vale **False**, le istruzioni all'interno del **while** **non** sono più eseguite, e si passa alle istruzioni successive.

COSTRUTTO `while`

```
1 while condizione:  
2     istruzioni interne al while
```

- Possiamo pensarlo come “**ripeti finché la condizione rimane vera**”

COSTRUTTO while

```
1 while condizione:  
2     istruzioni interne al while
```

- Possiamo pensarlo come “**ripeti finché la condizione rimane vera**”
- **Attenzione:** bisogna verificare che l'espressione presente nella condizione venga modificata nelle istruzioni interne al while.

COSTRUTTO `while`

```
1 while condizione:  
2     istruzioni interne al while
```

- Possiamo pensarlo come “**ripeti finché la condizione rimane vera**”
- **Attenzione:** bisogna verificare che l'espressione presente nella condizione venga modificata nelle istruzioni interne al `while`.
- **Attenzione:** anche se la forma è visivamente simile, il costrutto `while` è molto diverso semanticamente dal costrutto `if`.

Costrutto `while`

```
1 while condizione:  
2     istruzioni interne al while
```

- Possiamo pensarlo come “**ripeti finché la condizione rimane vera**”
- **Attenzione:** bisogna verificare che l'espressione presente nella condizione venga modificata nelle istruzioni interne al `while`.
- **Attenzione:** anche se la forma è visivamente simile, il costrutto `while` è molto diverso semanticamente dal costrutto `if`.
- Cosa accade?

```
1 a = 3  
2 while a > 0:  
3     print("Numero positivo")
```

Scrivere un programma Python che genera un numero random *compreso tra 1 e 100*.

N.B. Usare la funzione `random.randint(a, b)` del modulo `random` – Return a random integer N such that $a \leq N \leq b$. Alias for `randrange(a, b+1)`.

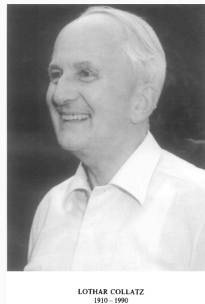
<https://docs.python.org/3.7/library/random.html>

Chiedere all'utente di indovinarlo e rispondere **"troppo grande"** o **"troppo piccolo"**.

Proseguire così finché l'utente non indovina.

La Congettura di Collatz (nella foto, il matematico tedesco) si basa sul seguente algoritmo

1. Si prende un intero positivo n .
2. Se $n = 1$ l'algoritmo termina.
3. Se n è
 pari, lo si divide per due (divisione intera); se
 dispari, lo si moltiplica per 3 e si aggiunge 1.
4. Si riparte da capo con il nuovo n così trovato.



Info. La congettura di Collatz sostiene che l'algoritmo descritto *termini* sempre.

ESERCIZIO (CONGETTURA DI COLLATZ)

Scrivere una funzione `Collatz(n)` che *stampi* (possibilmente sulla stessa riga) la sequenza di Collatz da n a 1. La sequenza di Collatz viene generata a partire dall'algoritmo considerato dalla congettura.

Esempio. `Collatz(29)` stampa:

29 88 44 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

Cosa calcola e restituisce questa funzione?

```
1 def foo(x):  
2     d=()  
3     for i in range(1,x+1):  
4         if x%i ==0:  
5             d=d+(i ,)  
6     return d
```

Cosa calcola e restituisce questa funzione?

```
1 def foo(x):  
2     d=()  
3     for i in range(1,x+1):  
4         if x%i ==0:  
5             d=d+(i ,)  
6     return d
```

Riscrivere la funzione usando il costrutto **while**.

Si scriva una funzione `somma(t)` che, preso come parametro `t`

- se `t` non è una tupla, restituisce `None`
- verifica che `t` contenga solo interi (**usando il ciclo `while`**), altrimenti restituisce `False`.
- restituisce `True` se e solo se la tupla di interi `t` gode della seguente proprietà: ogni elemento, ad eccezione del primo, è maggiore della somma di tutti gli elementi che lo precedono.

N.B. Usare il costrutto `while`.

Scrivere una funzione che prende come parametro una stringa e stampa (anche una per riga) tutte le vocali contenute nella stringa.

Usare il ciclo `while` ed il metodo `lower()`

N.B. `lower()`, chiamato su una stringa (`"Ciao".lower()`) restituisce una stringa con gli stessi caratteri in minuscolo, mantenendone l'ordine (`"ciao"`).

Scrivere un programma di gestione dei voti di uno studente, che presenta il seguente elenco di scelte all'utente che sceglie inserendo il codice associato ad ogni opzione:

- inserire un nuovo voto in una tupla [1];
- stampare tutti i voti [2];
- stampare la media dei voti [3];
- terminare il programma [0].

esempio di interazione con l'utente nella pagina successiva.

```
***Gestione voti studente***
```

- Inserire un voto [1]
- Stampare tutti i voti [2]
- Stampare la media dei voti [3]
- Terminare il programma [0]

```
>>> Scelta: 4
```

```
>>> Scelta non valida
```

```
>>> Scelta: 3
```

```
>>> Nessun voto su cui calcolare la media
```

```
>>> Scelta: 2
```

```
>>> ()
```

```
>>> Scelta: 1
```

```
>>> Inserisci un voto: 30
```

```
>>> Scelta: 1
```

```
>>> Inserisci un voto: 28
```

```
>>> Scelta: 2
```

```
>>> (30, 28)
```

```
>>> Scelta: 3
```

```
>>> 29.0
```

```
>>> Scelta: 0
```

Scrivere una funzione che preso un numero come parametro restituisca **True** se è primo, e **False** altrimenti.

Testare con 32416190071.

N.B. Usare **while** e *NON* **for**.

Scrivere una funzione senza parametri che conta la punteggiatura in una stringa chiesta in input all'utente, stampa **“Il numero di segni di punteggiatura è”** e, anche, restituisce tale numero.

Il modulo **string** ha numerose funzioni e costanti che possono tornare utili, in particolare la costante **string.punctuation** è la stringa contenente segni di punteggiatura.

N.B. Usare **while** e *NON* **for**.

Una sequenza di interi A è ciclica modulo k se

1. per ogni indice i , l'elemento $A[i + 1]$ è il successore modulo k di $A[i]$, cioè $A[i+1] == (A[i]+1)\%k$;
2. $A[0]$ è il successore modulo k dell'ultimo elemento di A .

Si scriva una funzione `ciclo(A,k)` che data una tupla di interi A e un intero k , verifica se A è ciclica modulo k .

Testare con $A = (1,0,1,0)$ e $k = 2$.

N.B. Usare `while` e *NON* `for`.