# Fondamenti di Informatica

# Ing. Gestionale

# a.a. 2022/2023

# Esercitazione del 04/04/2023

Tra parentesi trovate (se esiste) il riferimento all'esercizio di riferimento (con possibili modifiche) dal libro di testo del corso:

Horstmann, C., & Necaise, R. D. (2019). *Concetti di Informatica e Fondamenti di Python* (2ª edizione). Maggioli Editore. ISBN: 9788891635433. http://www.apogeoeducation.com/concetti-di-informatica-e-fondamenti-di-python.html

## Esercizio 1 (P3.1)

Scrivete un programma che acquisisca un numero intero e visualizzi un messaggio diverso nei casi in cui sia negativo, uguale a zero o positivo.

## Esercizio 2

Scrivete un programma che legga in input la targa di una automobile (es., AB123CD), ignorando eventuali spazi all'inizio ed alla fine, e restituisca "targa valida" oppure

"targa non valida", a seconda che l'input rispetti o meno la seguente regola (semplificata rispetto alla realtà):

una targa è formata da 2 lettere maiuscole, 3 cifre decimali, e infine di nuovo 2 lettere maiuscole.

*Commenti alle soluzioni:*

primo approccio (esercizio-02-02.py):

Per validare la prima componente della targa (e analogamente la terza) devo usare una condizione complessa:

```
prima_parte_valida = (prima_parte.isupper() and
                      prima_parte.isalpha()and
                      prima_parte.isascii())
```

*prima_parte.isupper()* verifica che tutti i caratteri della stringa che possono essere maiuscoli lo siano e che la stringa abbia almeno uno di questi caratteri.

Quindi, *"AB".isupper()* è *True*, mentre *"Ab".isupper()* è *False*. Tuttavia, ho anche che *"A1".isupper()* è *True* perché 1 non ha il maiuscolo ed ho anche che *"ÈA".isupper()* è *True*, perché È è un carattere (non-ASCII) maiuscolo.

Pertanto, ho bisogno di mettere in *"and"* ulteriori condizioni:

- *prima_parte.isalpha()* verifica che la stringa sia formata soltanto da caratteri alfabetici e che abbia almeno un carattere. Sto escludendo, quindi, quelle

stringhe (es. "A1") che contengono caratteri non alfabetici.

- prima_parte.isascii() verifica che la stringa sia formata soltanto da caratteri ASCII (https://en.wikipedia.org/wiki/ASCII) e che abbia almeno un carattere. Sto escludendo, quindi, stringhe contenti caratteri come *È* che sono maiuscoli ma non rientrano nel range "A".."Z" che ci interessa

Lo svantaggio di questo primo approccio è la complessità dell'espressione che abbiamo dovuto scrivere dal momento che *isupper()* non fa esattamente ciò che vogliamo.

secondo approccio (esercizio-02-02b-BAD.py):

Iteriamo sui caratteri della stringa in input ed applichiamo un criterio diverso in funzione della posizione cui ci troviamo. Qui, il problema è la condizione per sapere se un carattere è una lettera maiuscola (ed in misura minore anche la condizione sulla parte numerica):

*c in "ABCDEFGHIJKLMNOPQRSTUVWXYZ"*

l'operatore *in* ci permette di verificare se la stringa usata come primo argomento (qui contenuta nella variabile *c*)

occorre nella stringa a destra. Qui, il problema è che il secondo argomento è una stringa molto lunga nella quale è facile commettere un errore (dimenticare una lettera o aggiungerne una sbagliata)

terzo approccio (esercizio-02-02c.py):

Esprimiamo la condizione in maniera più compatta:

*"A" <= c <= "Z"*

Sfruttando il fatto che le stringhe sono confrontabili.

Un fatto importante:

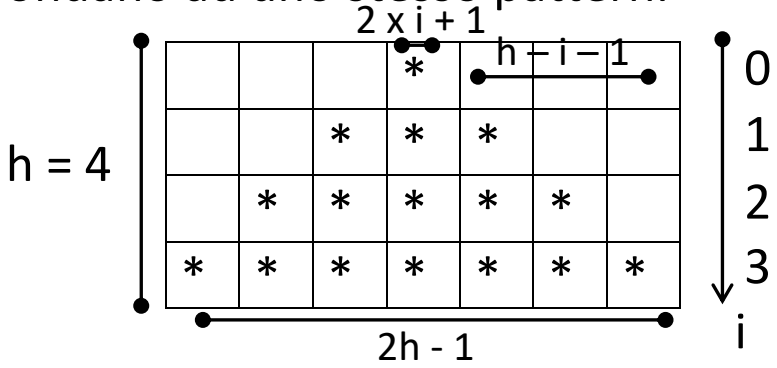
```
while i < len(targa) and targa_valida:
```

Nella condizione ho aggiunto and targa_valida in modo che il ciclo termini nonappena troviamo che la targa non è valida. Altrimenti, senza questa ulteriore condizione, continueremmo a scorrere la stringa e la considereremmo valida basta che l'ultimo carattere sia valido.

## Esercizi per casa della esercitazione precedente

### Esercitazione 03/22: Esercizio 7

Le righe che compongono la immagine di una piramide rispondano ad uno stesso pattern.



Ovviamente, il pattern si sarebbe potuto scrivere in maniera leggermente diversa anche iterando per i da 1 ad h incluso.

## Esercizi da svolgere a casa

### Esercizio 3 (P3.5)

Scrivete un programma che legga tre numeri e visualizzi il messaggio “increasing” se sono in ordine crescente, “decreasing” se sono in ordine decrescente e “neither” se non sono né in ordine crescente né in ordine decrescente. In questo esercizio *crescente* significa *strettamente* crescente, cioè ciascun valore deve essere maggiore del precedente (analogo significato ha il termine *decrescente*): la sequenza 3 4 4, quindi, non va considerata crescente.

### Esercizio 4

Scrivere un programma che legge tre numeri interi e li visualizzi in ordine non decrescente (crescente con possibili numeri uguali):

INPUT: 9, 3, 7

OUTPUT: 3, 7, 9

INPUT: 4, 1, 1

OUTPUT: 1, 1, 4

*Considerazioni circa il testing*:

È possibile ridurre i casi di prova.

Un primo insieme di casi di prova: 27

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 2 |
| 1 | 1 | 3 |
| 1 | 2 | 1 |
| 1 | 2 | 2 |
| 1 | 2 | 3 |
| 1 | 3 | 1 |
| 1 | 3 | 2 |
| 1 | 3 | 3 |
| 2 | 1 | 1 |
| 2 | 1 | 2 |
| 2 | 1 | 3 |
| 2 | 2 | 1 |
| 2 | 2 | 2 |
| 2 | 2 | 3 |
| 2 | 3 | 1 |
| 2 | 3 | 2 |
| 2 | 3 | 3 |
| 3 | 1 | 1 |
| 3 | 1 | 2 |
| 3 | 1 | 3 |
| 3 | 2 | 1 |
| 3 | 2 | 2 |
| 3 | 2 | 3 |
| 3 | 3 | 1 |
| 3 | 3 | 2 |
| 3 | 3 | 3 |

Una selezione più efficiente di casi di prova: 13

| | | | |
|---|---|---|---|
| Tutti uguali | 1 | 1 | 1 |
| Tutti diversi (6 permutazioni) | 1 | 2 | 3 |
| | 1 | 3 | 2 |
| | 2 | 1 | 3 |
| | 2 | 3 | 1 |
| | 3 | 1 | 2 |
| | 3 | 2 | 1 |
| 2 uguali, 1 diverso maggiore (sposto il terzo diverso) | 1 | 1 | 2 |
| | 1 | 2 | 1 |
| | 2 | 1 | 1 |
| 2 uguali, 1 diverso minore (sposto il terzo diverso) | 2 | 2 | 1 |
| | 2 | 1 | 2 |
| | 1 | 2 | 2 |

## Esercizio 5

Scrivere un programma che legge tre numeri interi e ne determini il massimo. PS: non usare la funzione builtin *max*.

## Esercizio 6 (P3.4)

Scrivete un programma che legga tre numeri e visualizzi il messaggio “all the same” se sono tutti uguali, “all different” se sono tutti diversi e “neither” se non sono tutti uguali e non sono tutti diversi.

## Esercizio 7 (P3.20)

Scrivere un programma che chieda all'utente un mese e un giorno e poi visualizzi la stagione di quel determinato mese e giorno.

INFO:

Gennaio - 1 , Febbraio - 2 ,Marzo - 3 = "Inverno"

Aprile - 4 , Maggio - 4 ,Giugno - 6 = "Primavera"

Luglio - 7 , Agosto - 8 , Settembre - 9 = "Estate"

Ottobre - 10 , Novembre - 11, Dicembre -12 = “Autunno"

## Esercizio 8 (P3.20)

Scrivere un programma che converta un numero intero positivo nel corrispondente valore espresso nel sistema di numerazione romano. I numeri romani sono costituiti da sei diverse cifre:

I 1

V 5

X 10

L 50

C 100

D 500

M 100

E i numeri vengono composti seguendo queste regole:

a) Si possono rappresentare soltanto i numeri fino a 3999.
b) Come nel sistema decimale, le migliaia, le centinaia, le decine e le unità vengono espresse separatamente.
c) I numeri da 1 a 9 sono rappresentati, in ordine crescente, da I, II, III, IV, V, VI, VII, VIII, IX; come potete notare, una lettera I che precede la lettera V o X rappresenta un'unità che viene *sottratta* dal valore; inoltre, non si possono avere più di tre I consecutive.
d) Le decine e le centinaia sono gestite allo stesso modo, tranne per il fatto che, al posto delle lettere I, V, X, si usano le lettere X, L, C e, rispettivamente, le lettere C, D e M.
e) Il programma deve acquisire un numero in ingresso, come 1978, e convertirlo secondo la numerazione romana, visualizzando, in questo esempio, MCMLXXVIII.