# Fondamenti di Informatica

# Ing. Gestionale

# a.a. 2022/2023

# Esercitazione del 06/04/2023

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 (P4.12)

Scrivete un programma che legga una parola e visualizzi tutte le sue sottostringhe, ordinate per lunghezza crescente. Se, ad esempio, l'utente fornisce la stringa "rum", il programma deve visualizzare:

r
u
m
ru
um
rum

*Commenti alle soluzioni:*

Assumendo che la variabile *parola* contenga la stringa "rum"

parola[0:1] == "r"

parola[1:2] == "u"

parola[2:3] == "m"

parola[0:2] == "ru"

parola[1:3] == "um"

parola[0:3] == "rum"

in generale, per *i >= 0* e *lunghezza_sottostringa >= 1*

*parola[i:i+lunghezza_sottostringa]* è la sottostringa dall'indice *i* all'indice *i + lunghezza_sottostringa* escluso.

Siccome l'indice *i + lunghezza_sottostringa* viene escluso, deve essere *i + lunghezza_sottostringa* <= len(parola).

La condizione di cui sopra si può scrivere anche come

i <= len(parola) - lunghezza_sottostringa

ovvero, sommando 1 (solo) al membro destro,

i < len(parola) - lunghezza_sottostringa + 1

cosa che ci permette di formulare il ciclo col costrutto for in range(…).

Si noti che se *lunghezza_sottostringa > len(parola)*, viene fuori che *i* deve essere negativo, cosa che non vogliamo.

**NOTA:**

In realtà, lo slicing è molto robusto rispetto all'uso di indici fuori dall'intervallo consentito.

Per esempio, se parola == "rum" allora

parola[1:5] == "um"

il "problema" è che in questi casi la differenza degli indici non indica più la lunghezza della sottostringa, es. 5 – 1 == 4

## Esercizio 2

Scrivere un programma che legga un intero non negativo e lo stampi separando le cifre a gruppi da tre usando la virgola alla maniera anglosassone. Ad esempio,

*1234* diventa *1,234*

*2096* diventa *2,096*

*1095604 diventa 1,095,604*

*Commenti alle soluzioni:*

Le due soluzioni differiscono per il costrutto usato per inserire gli zeri per formare il gruppo di 3 cifre da aggiungere.

## Esercizi da svolgere a casa

## Esercizio 3 (P4.1)

Scrivete programmi che, usando cicli, calcolino quanto segue:

a. La somma di tutti i numeri pari compresi tra 2 e 100 (estremi inclusi);
b. La somma di tutti i numeri compresi tra 1 e 100 (estremi inclusi) che siano quadrati perfetti;
c. Tutte le potenze di 2, da 20 a 220;
d. La somma di tutti i numeri dispari compresi tra a e b (estremi inclusi), dove a e b sono valori acquisiti in ingresso;
e. La somma di tutte le cifre dispari di un numero acquisito in ingresso (se, ad esempio, il numero è 32677, la somma da calcolare è 3 + 7 + 7 = 17);

## Esercizio 4

Scrivere un programma che legga in input una stringa $s_1$ ed una stringa $s_2$ e che quindi stampi tutte le posizioni in cui $s_2$ compare come sottostringa di $s_1$.

## Esercizio 5

Scrivere un programma che legga in input una stringa $s_1$ e stampi una stringa $s_2$ nella quale siano stati rimossi caratteri duplicati adiacenti.

Per esempio, la stringa *"aabbcad"* diventa *"abcad"*

## Esercizio 6

Scrivere un programma che legga in input una stringa $s_1$ e stampi una stringa $s_2$ nella quale siano state inserite delle virgole ogni tre caratteri a partire dalla fine della stringa

Per esempio, la stringa *"12345679"* diventa *"12,345,679"*

# Esercizi per casa della esercitazione precedente

Alcune osservazioni sparse:

*x == y* è equivalente a *y == x*

*x < y* è equivalente a *y > x*

*x <= y* è equivalente a *y >= x*

*x > y* è equivalente a *y < x*

*x >= y* è equivalente a *y <= x*

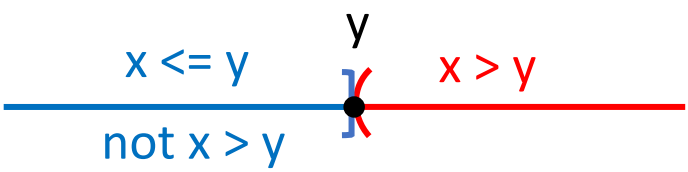
*x < y* è equivalente a *not x >= y*

*x <= y* è equivalente a *not x > y*

*x > y* è equivalente *not x <= y*

*x >= y* è equivalente *not x < y*

Ecco un esempio:




Si consideri il seguente *if*

if x > 3:

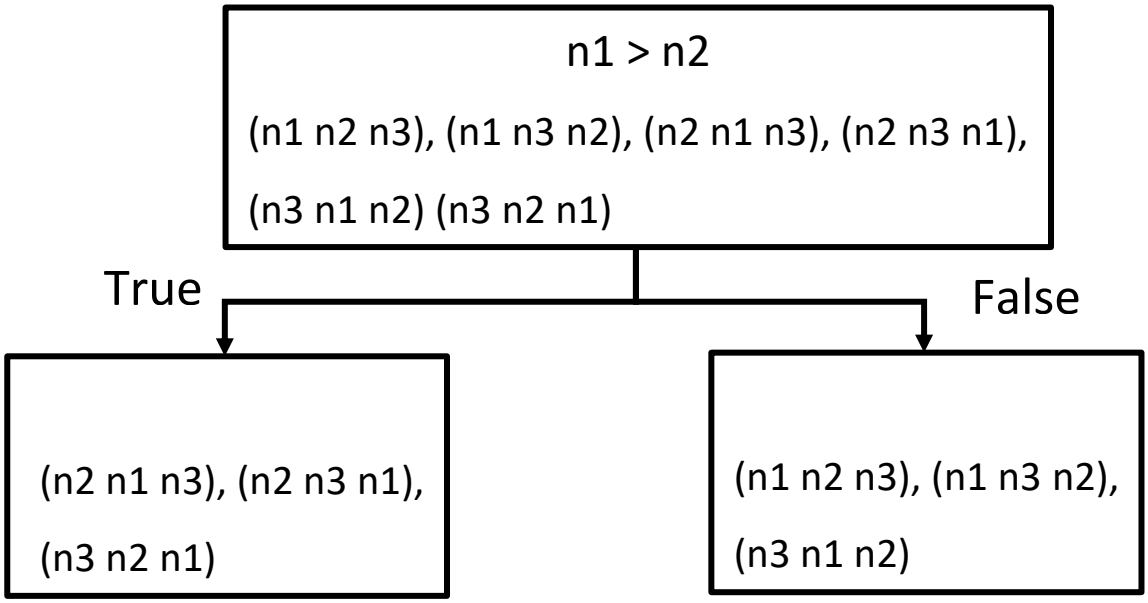
*qui sappiamo che x è maggiore di 3*

else:

*qui sappiamo che x è minore o uguale 3*

Alcune osservazioni circa il programma che ordina 3 variabili:

- Devono esserci 3! = 6 stampe diverse per coprire tutti i possibili modi in cui le variabili possono essere riordinate. In effetti, le 6 permutazioni di 3 numeri distinti rappresentano un ottimo insieme di casi di test.
- All'inizio non sappiamo quale tra le 6 possibili permutazioni dobbiamo scegliere. Ogni *if* ci permette di focalizzarci su un sottoinsieme.
  Considerando l'esempio dell'ordinamento *non decrescente*, supponiamo che nel primo *if* usiamo la condizione *n1 > n2*. A seconda che la condizione sia *True* ("ramo *if*") o *False* ("ramo *else*") possiamo considerare un sottoinsieme delle possibili permutazioni. Nel primo caso, essendo n1 > n2, possiamo escludere tutte le permutazioni in cui n1 "viene prima" di n2, che invece considereremo nell'altro caso (n1 <= n2).




Continueremo così finché in ogni ramo non ci siamo ristretti ad una sola permutazione.