# Esercizi di Fondamenti di Informatica

Andrea Gussoni
andrea1.gussoni at polimi.it

Politecnico di Milano

November 07, 2019

# Table of Contents

Section 1

# Ripasso Esercitazione Precedente

# Ripasso

Problemi con esercizi della precedente esercitazione?

- Tipi base
- Costrutti di controllo decisionale
- Costrutto di controllo ciclico `while`

# Ripasso

Problemi con esercizi della precedente esercitazione?

- Tipi base
- Costrutti di controllo decisionale
- Costrutto di controllo ciclico `while`

# Ripasso

Problemi con esercizi della precedente esercitazione?

- Tipi base
- Costrutti di controllo decisionale
- Costrutto di controllo ciclico `while`

Section 2

# Esercizi Vari

# While e For

Ideare un programma che esegua la moltiplicazione tra due numeri, non avendo però a disposizione l'operatore $*$ (moltiplicazione) di Python.

# While e For

Possibile soluzione:

```
moltiplicando = int(input("Inserire il moltiplicando"))
moltiplicatore = int(input("Inserire il moltiplicatore"))

accumulatore = 0

i = 0
while i < moltiplicatore:
    accumulatore = accumulatore + moltiplicando
    i = i + 1

print("Risultato", accumulatore)
```

# While e For

Il costrutto `for` non è altro che *zucchero sintattico* sul costrutto while:

- Il costrutto `for` permette di iterare su una sequenza di valori, ed eseguire il codice contenuto nel corpo del costrutto per uguale numero di volte.
- Possiamo utilizzare il costrutto `for` al posto di un ciclo `while` che presenta:
  - Una inizializzazione della variabile di induzione.
  - Un incremento(diminuzione) costante della stessa ad ogni iterazione del ciclo.
  - Un controllo, nella condizione del `while`, del valore della variabile di induzione rispetto ad un massimo(minimo).
- Si utilizza generalmente la funzione `range(min,max)` per ottenere una sequenza di valori sui quali l'iterazione è effettuata.

# While e For

Soluzione con il costrutto for:

```
moltiplicando = int(input("Inserire il moltiplicando"))
moltiplicatore = int(input("Inserire il moltiplicatore"))

accumulatore = 0

for i in range(0, moltiplicatore, 1):
    accumulatore = accumulatore + moltiplicando

print("Risultato", accumulatore)
```

# Massimo

Ideare un programma che prende in input una serie di numeri interi, terminati dal valore speciale -1. Il programma stamperà quindi il massimo tra i valori immessi.

# Massimo

Possibile soluzione:

```
massimo = 0

valore = int(input("Inserire un valore intero positivo,
                    -1 per terminare"))

while valore != -1:
    if valore > massimo:
        massimo = valore

    valore = int(input("Inserire un valore intero
                        positivo, -1 per terminare"))

print("massimo:", massimo)
```

# Minimo, Massimo, Media

Ideare un programma che prende in input una serie di numeri interi, terminati dal valore speciale -1. Il programma stamperà quindi il massimo e il minimo, la media aritmetica e il numero dei valori immessi.

# Minimo, Massimo, Media I

Possibile soluzione:

```
massimo = 0
minimo = 92233720368547758O7 # Intero massimo rappresentabile
accumulatore = 0
cardinalita = 0

valore = int(input("Inserire un valore intero positivo,
                    -1 per terminare"))

while valore != -1:
    if valore > massimo:
        massimo = valore
    if valore < minimo:
        minimo = valore
```

# Minimo, Massimo, Media II

```
    accumulatore = accumulatore + valore
    cardinalita = cardinalita + 1

    valore = int(input("Inserire un valore intero
                        positivo, -1 per terminare"))

if cardinalita != 0:
    media = accumulatore/cardinalita
    print("massimo:", massimo, "minimo:", minimo,
          "media:", media, "numero:", cardinalita)
else:
    print("Nessun valore immesso")
```

# Tabella delle Potenze

Ideare un programma che accetti come input un intero n, e che stampi la tabella delle potenze dei numeri da 1 a 10 fino alla potenza indicata da n.

# Tabella delle Potenze

Esempio output:

```
   1          2          3
   x          x          x
 ------------------------------
   1          1          1
   2          4          8
   3          9         27
   4         16         64
   5         25        125
   6         36        216
   7         49        343
   8         64        512
   9         81        729
  10        100       1000
```

# Tabella delle Potenze I

Possibile soluzione:

```
x_max = int(input("Intero massimo"))
power_max = 10

for i in range(1, x_max + 1):
    print("%10d" % i, end="")

print()

for i in range(1, x_max + 1):
    print("%10s" % "x ", end="")

print("\n", "     ", "—" * 35)
```

# Tabella delle Potenze II

```
for x in range(1, power_max + 1):
    for n in range(1, x_max + 1):
        print("%10.0d" % x ** n, end="")

    print()
```

Section 3

# Stringhe come Container

# Stringhe come Container

Successivamente vedremo i *container* del linguaggio *Python*:

- I container non sono altro che oggetti speciali che permettono di immagazzinare un numero arbitrario di altri oggetti.
- Solitamente i container permettono di accedere ed iterare facilmente sulla collezione di oggetti che contengono.
- Possiamo considerare le stringhe come il primo esempio di container.
- Le stringhe infatti non sono altro che un container che contiene un numero arbitrario di caratteri.

# Stringa Inversa

Ideare un programma che prenda in input una stringa, e stampi in output la stringa letta al contrario (da destra a sinistra).

# Cheat Sheet Operazioni tra Stringhe

Come visto la volta precedente, se definiamo le stringhe A = "Hello", B = "World", abbiamo a disposizione i seguenti operatori:

- Concatenazione: `A + B` $\rightarrow$ `HelloWorld`
- Concatenazione ripetuta: `A * 2` $\rightarrow$ `HelloHello`
- Accesso al singolo carattere: `A[1]` $\rightarrow$ e
- Lunghezza stringa: `len(B)` $\rightarrow$ 5

# Stringa Inversa

Python offre un ulteriore modo molto comodo per accedere a sequenze di caratteri appartenenti ad una stringa:

- Accedere ad un intervallo specificato di caratteri: `stringa[4:9]` (dal quinto al decimo carattere).
- Omettendo uno degli estremi, automaticamente si usano l'inizio e il termine della stringa: `stringa[:9]` e `stringa[5:]`.

# Stringa Inversa

Possibile soluzione:

```
stringa = input("Immettere la stringa")
inversa = ""

lunghezza = len(stringa)

i = lunghezza - 1
while i >= 0:
    inversa = inversa + stringa[i]
    i = i -1

print(inversa)
```

Section 4

# Funzioni

# Funzioni

- Una funzione è una sequenza di istruzioni ben specifiche etichettate con un nome.
- Il vantaggio delle funzioni è che permettono di riutilizzare una porzione di codice in molti punti senza dover duplicare il codice stesso.
- Il modo più comune di visualizzare una funzione è pensare ad essa come ad una *black box*:

# Funzioni

- Una funzione può accettare uno o più parametri, che possiamo pensare come delle variabili che saranno poi disponibili all'interno del corpo della funzione stessa.
- Una funzione inoltre ha generalmente un valore di ritorno. Questo valore di ritorno è in sostanza un valore che rappresenta l'esito della computazione della funzione.
- Abbiamo già usato diverse funzioni, e.g., la funzione `len` per le stringhe.
- In quel caso, il parametro era la stringa stessa, e il valore di ritorno la lunghezza della stringa.
- Attenzione alla nomenclatura: differenza tra parametro (argomento formale) e argomento (parametro attuale).

# Bubble Sort

- L'algoritmo di *bubble sort* è un semplice algoritmo per effettuare l'ordinamento.
- Cominceremo ideando un programma che eseguira l'ordinamento lessicografico di una stringa.
- Ovvero ordinare i caratteri in una stringa secondo l'ordine alfabetico.
- La stringa *zaog* deve diventare la stringa *agoz*.

# Bubble Sort

Possibile soluzione utilizzando il range slicing:

```
stringa = input("Inserire una stringa")

dim = len(stringa)
modifica = True

while modifica:
    modifica = False
    for elemento in range(0, dim - 1):
        if stringa[elemento] > stringa[elemento + 1]:
            ordinata = stringa[0:elemento] + stringa[elemento + 1]
                       + stringa[elemento] + stringa[elemento + 2:dim]
            modifica = True
            stringa = ordinata

print("La stringa ordinata:", stringa)
```

# Bubble Sort

- E se dovessimo ordinare una serie di stringhe in ordine lessicografico? Come fare per riutilizzare il codice che abbiamo scritto?
- Possiamo incapsulare il codice in una funzione!

# Bubble Sort I

Possiamo creare una funzione dedicata che effettua l'ordinamento:

```
def ordina(stringa):
    dim = len(stringa)
    modifica = True
    while modifica:
        modifica = False
        for elemento in range(0, dim - 1):
            if stringa[elemento] > stringa[elemento + 1]:
                ordinata = stringa[0:elemento]
                           + stringa[elemento + 1]
                           + stringa[elemento]
                           + stringa[elemento + 2:dim]
                modifica = True
                stringa = ordinata
    return stringa
```

# Bubble Sort II

```
stringa = input("Inserire una stringa")
ordinata = ordina(stringa)
print("La stringa ordinata:", ordinata)
```