Esercizi di Fondamenti di Informatica

Andrea Gussoni
andrea1.gussoni at polimi.it

Politecnico di Milano

November 21, 2019

Table of Contents

1 Ripasso Esercitazione Precedente

2 Strutture dati
- Liste

Section 1

Ripasso Esercitazione Precedente

Ripasso

Problemi con esercizi della precedente esercitazione?

- Stampa pattern
- Funzioni

Ripasso

Problemi con esercizi della precedente esercitazione?

- Stampa pattern
- Funzioni

Password Casuale

Ideare un programma che si occupi della generazione di una password casuale. La password dovrà avere i seguenti requisiti:

- Chiedere all'utente da quanti caratteri dovrà essere composta.
- Avere una cifra numerica al suo interno.
- Avere un carattere speciale ($+ - */?!@\#\%\&$) al suo interno.

Password Casuale I

Il codice:

```
from random import randint

def main():
    l = int(input("Lunghezza password"))
    password = genera(l)
    print(password)

def genera(n):
    password = ""
    for i in range (0, n - 2):
        password = password + casuale("abcdefghijklmnopqrstuvwxyz")

    numero = casuale("0123456789")
    password = inserisci(password, numero)
```

Password Casuale II

```
    carattere_speciale = casuale("+-*/?!@#\$\%\&")
    password = inserisci(password, carattere_speciale)

    return password

def casuale(input):
    n = len(input)
    r = randint(0, n - 1)
    return input[r]

def inserisci(stringa, char):
    n = len(stringa)
    r = randint(0, n)
    nuova = ""
```

Password Casuale III

```
    for i in range(r):
        nuova = nuova + stringa[i]

    nuova = nuova + char

    for i in range(r, n):
        nuova = nuova + stringa[i]

    return nuova

main()
```

Section 2

Strutture dati

Funzioni accessorie

- Quando lavoriamo con delle strutture dati, è molto utile avere a disposizione un set di funzioni nella nostra cassetta degli attrezzi che ci permettono agilmente di compiere operazioni comuni sulle nostre strutture dati.
- Implementiamo le seguenti funzioni:
  - Somma di tutti i valori in una lista di interi
  - Ricerca di un elemento in una lista
  - Inserimento di un elemento in una certa posizione

Somma Lista

```
def somma_lista(l):
    accum = 0
    for elem in l:
        accum = accum + elem

    return accum

def main():
    lista = [1, 2, 3, 4, 5]
    print(somma_lista(lista))

main()
```

Ricerca Lista

```
def ricerca_lista(target, l):
    for elem in l:
        if target == elem:
            return True

    return False

def main():
    lista = [1, 3, 5, 7, 9]
    print(ricerca_lista(1, lista))
    print(ricerca_lista(2, lista))

main()
```

Inserisci Lista

```
def inserisci_posizione(elem, pos, l):
    l = l[0:pos] + [elem] + l[pos:]
    return l

def main():
    lista = [1,2,4,5]
    lista = inserisci_posizione(3, 2, lista)
    print(lista)

main()
```

Pari e Dispari

Ideare un programma che prenda in input una seria di interi, terminati dal valore speciale -1, e crei due nuove liste, che contengano rispettivamente tutti i valori pari e tutti i valori dispari presenti nella lista originaria.

Pari e Dispari I

Una possibile soluzione

```
def main():
    lista = []
    pari = []
    dispari = []

    numero = 0
    while numero != -1:
        numero = int(input("Inserisci un intero"))
        if (numero >= 0):
            lista = lista + [numero]

    for elemento in lista:
        if elemento % 2 == 0:
            pari = pari + [elemento]
```

Pari e Dispari II

```
        else:
            dispari = dispari + [elemento]

    print("Numeri inseriti:", lista)
    print("Valori pari:", pari)
    print("Valori dispari:", dispari)

main()
```

Fusione Liste

Ideare un programma che prenda in input due liste ordinate, e le fonda per ottenere un'unica lista che deve essere a sua volta ordinata.

Fusione Liste I

```
def fondi(l1, l2):
    dim1 = len(l1)
    dim2 = len(l2)
    cur1 = 0
    cur2 = 0

    ris = []

    while cur1 < dim1 and cur2 < dim2:
        if l1[cur1] <= l2[cur2]:
            ris = ris + [l1[cur1]]
            cur1 = cur1 + 1
        else:
            ris = ris + [l2[cur2]]
            cur2 = cur2 + 1
```

Fusione Liste II

```
    # Gestione parte rimanente ultima lista
    while (cur1 < dim1 or cur2 < dim2):
        if cur1 < dim1:
            ris = ris + [l1[cur1]]
            cur1 = cur1 + 1
        else:
            ris = ris + [l2[cur2]]
            cur2 = cur2 + 1

    return ris

def main():
    lista1 = [1, 3, 5, 7]
    lista2 = [2, 4, 6, 8]
    lista = fondi(lista1, lista2)
```

Fusione Liste III

```
    print(lista)

main()
```

Insertion Sort

Ideare un programma che, presa in input una lista di numeri interi, ne effettui l'ordinamento. La parte del programma che effettua l'ordinamento dovrà essere incapsulata in una funzione, in modo da poterla riutilizzare facilmente. L'algoritmo di ordinamento usato sara l'algoritmo di *selection sort*.

Insertion Sort I

Una possibile soluzione

```
def ordina(lista):

    for i in range(len(lista)):
        elemento = lista[i]
        pos = i

        while pos > 0 and lista[pos - 1] > elemento:
            lista[pos] = lista[pos - 1]
            pos = pos - 1

        # Sistemiamo l'elemento
        lista[pos] = elemento

    return lista
```

Insertion Sort II

```
def main():
    print(ordina([3,2,1]))

main()
```

Matrice

Ideare un programma che acquisca da tastiera gli elementi di una matrice $n \times n$. Per farlo, prima l'utente inserirà il valore di n, e successivamente inserirà tutti i valori (con una scansione riga per riga). Una volta acquisita la matrice, il programma dovrà stamparla a schermo, verificare che la matrice sia simmetrica ed in questo caso riportare gli elementi sulle diagonali principale e secondaria.

Matrice I

Una possibile soluzione

```
def main():
    n = -1
    while n < 0:
        n = int(input("Inserire il valore di n"))

    matr = []
    for riga in range(0, n):
        lista_riga = []
        for col in range(0, n):
            print("Inserisci il valore in posizione
                  [", riga, "][", col, "]")
            elem = int(input(""))
            lista_riga.append(elem)
        matr.append(lista_riga)
```

Matrice II

```
print("Matrice inserita:")
for riga in matr:
    print(riga)

simmetrica = True
for r in range(0, n):
    for c in range(0, n):
        if matr[r][c] != matr[c][r]:
            simmetrica = False

if simmetrica == True:
    print("Matrice simmetrica")
    print("Diagonale principale:", end="")
    for i in range(0, n):
        print(matr[i][i], end="")
```

Matrice III

```
        print("\nDiagonale secondaria:", end="")
        for i in range(0, n):
            print(matr[i][n-1-i], end="")
    else:
        print("Matrice non simmetrica")

main()
```

Matrice I

Soluzione ottimizzata

```
def main():
    n = -1
    while n < 0:
        n = int(input("Inserire il valore di n"))

    matr = []
    for riga in range(0, n):
        lista_riga = []
        for col in range(0, n):
            print("Inserisci il valore in posizione
                  [", riga, "][", col, "]")
            elem = int(input(""))
            lista_riga = lista_riga + [elem]
        matr = matr + [lista_riga]
```

Matrice II

```
print("La matrice inserita:")
for riga in matr:
    print(riga)

simmetrica = True
r = 0
c = 0
while r < n and simmetrica == True:
    while c < n and c < r and simmetrica == True:
        if matr[r][c] != matr[c][r]:
            simmetrica = False
        c = c + 1
    r = r + 1

if simmetrica == True:
```

Matrice III

```
        print("Matrice simmetrica")
        print("Diagonale principale:", end="")
        for i in range(0, n):
            print(matr[i][i], end="")
        print("\nDiagonale secondaria:", end="")
        for i in range(0, n):
            print(matr[i][n-1-i], end="")
    else:
        print("Matrice non simmetrica")

main()
```

Alfabeto Farfallino

Ideare un programma che presa in unput una lista formata da caratteri, modifichi la stringa per rispecchiare la codifica ad alfabeto farfallino. La suddetta codifica prevede una trasformazione che in corrispondenza di ogni vocale inserisca una "f" e una ripetizione della vocale stessa (la stringa "a" diventerà "afa").

Alfabeto Farfallino I

Una possibile soluzione

```
def farfallino(l):
    pos = 0
    while pos < len(l):
        elem = l[pos]
        if elem == "a"
        or elem == "e"
        or elem == "i"
        or elem == "o"
        or elem == "u":
            l = l[0:pos] + [elem] + ["f"] + [elem] + l[pos + 1:]
            pos = pos + 3
        else:
            pos = pos + 1
    return l
```

Alfabeto Farfallino II

```
def main():
    lista = ["c", "i", "a", "o"]
    print(farfallino(lista))

main()
```

Alfabeto Farfallino I

Utilizzando una funzione aggiuntiva

```
def vocale(lettera):
    vocali = list("aeiou")
    if lettera in vocali:
        return True
    else:
        return False

def farfallino(l):
    pos = 0
    while pos < len(l):
        elem = l[pos]
        if vocale(elem):
            l = l[0:pos] + [elem] + ["f"] + [elem] + l[pos + 1:]
            pos = pos + 3
```

Alfabeto Farfallino II

```
        else:
            pos = pos + 1
    return l

def main():
    lista = ["c", "i", "a", "o"]
    print(farfallino(lista))

main()
```