Laboratorio 10

Programmazione - CdS Matematica

Michele Donini, Marta Gatto, Mirko Polato

19 gennaio 2016

Esercizio 1

Date le seguenti istruzioni, dire senza eseguire, quale sarà il valore contenuto nella variabile `t`.

```
x = 1
y = 2
t = (x, y*x, x*x, [y,3])
x = x * t[2]
t[3][1] = x
t[3].extend([x, y])
y = x**2 + t[1]
t[3][3] -= 1
```

Esercizio 1

Date le seguenti istruzioni, dire senza eseguire, quale sarà il valore contenuto nella variabile `t`.

```
x = 1
y = 2
t = (x, y*x, x*x, [y,3])
x = x * t[2]
t[3][1] = x
t[3].extend([x, y])
y = x**2 + t[1]
t[3][3] -= 1
```

Soluzione:

```
t = (1, 2, 1, [2, 1, 1, 1])
```

Esercizio 2

Date le seguenti istruzioni, dire senza eseguire, quale sarà il valore contenuto nella variabile `x`.

```
x = [42]
for i in range(4):
    y = x
    x[0] = i
    x = [x, y]
```

Esercizio 2

Date le seguenti istruzioni, dire senza eseguire, quale sarà il valore contenuto nella variabile `x`.

```
x = [42]
for i in range(4):
    y = x
    x[0] = i
    x = [x, y]
```

Soluzione:

```
x = [[3, [2, [1, [0]]]], [3, [2, [1, [0]]]]]
```

Esercizio 3

Creare una matrice contenente la tabellina dei numeri coompresi tra 1 e 10.
Generare la lista di tutti i numeri presenti nella tabellina, senza ripetizioni, in due modi:

1. con un descrittore di lista partendo dalla matrice
2. con un descrittore di lista partendo dalla lista di tutti i numeri compresi tra 1 e 100

Esercizio 3

Creare una matrice contenente la tabellina dei numeri coompresi tra 1 e 10.
Generare la lista di tutti i numeri presenti nella tabellina, senza ripetizioni, in due modi:

1. con un descrittore di lista partendo dalla matrice
2. con un descrittore di lista partendo dalla lista di tutti i numeri compresi tra 1 e 100

```
tab = [[i*j for j in range(1,11)] for i in range(1,11)]

l1 = list(set([i for row in tab for i in row]))

l2 = [i for i in range(1,101) if (sum([1 if ((mod(i,j)
    ==0 and  i/j<11)) else 0  for j in range(1,11)])>0)]
```

Esercizio 4

Partendo da `s = "Cualcossa cui non va"` generare le seguenti due stringhe:

1. `s1= "Qualcosa qui non va"`
2. `s2= "Qui qualcosa va"`

Esercizio 4

Partendo da `s = "Cualcossa cui non va"` generare le seguenti due stringhe:

1. `s1= "Qualcosa qui non va"`
2. `s2= "Qui qualcosa va"`

Soluzione:

```
s="Cualcossa cui non va"
s=s.lower()
s=s.replace('cu','qu')
s=s.replace('ss','s')
s1=s.capitalize()
l=s.split()
l2=[l[1],l[0],l[3]]
s2=' '.join(l2)
s2=s2.capitalize()
```

Esercizio 5

Date le seguenti funzioni:

```
def f1(n):
    return (n+7) % 26

def f2(n):
    return (n-3) % 26

def f3(n):
    return n

f = [f1, f2, f3]
```

scrivere un programma che data la stringa `s='zuzzurrellonando'` cifri i singoli caratteri di `s` applicando ciclicamente le funzioni contenute nella lista `f`.

Esercizio 5 – Soluzione

```
def f1(n):
    return (n+7) % 26

def f2(n):
    return (n-3) % 26

def f3(n):
    return n

f = [f1, f2, f3]
s = "zuzzurrellonando"
t = "".join([chr(f[i%len(f)](ord(c) - ord('a')) + ord('a
    ')) for i, c in enumerate(s)])
print t
#grzgrryblslnhkdv
```

Esercizio 6

Definire la classe Matrice e un metodo *mul* che prenda in input una lista (che considereremo come un vettore) ed esegua il prodotto matrice-vettore, ritornandone il risultato.

Esercizio 6

Definire la classe Matrice e un metodo *mul* che prenda in input una lista (che considereremo come un vettore) ed esegua il prodotto matrice-vettore, ritornandone il risultato.

```
class Matrice:
    def __init__(self, l):
        self.mat = l

    def mul(self,v):
        return [sum([self.mat[row][col] * v[col]
                     for col in range(len(v))])
                for row in range(len(self.mat))]

m = Matrice([[0,1],[1,0]])
vec = [3,4]
print m.mul(vec)
# [4, 3]
```

Esercizio 6

Definire le funzioni:

a) *norma* che presa in input una lista (per noi un vettore v) ne ritorna il valore della norma 2 (cioè $\sqrt{\sum_i v_i^2}$)

b) *diff* che prese in input due liste ritorna la lista che rappresenta il vettore della differenza dei due vettori dati

Esercizio 6

Definire le funzioni:

a) *norma* che presa in input una lista (per noi un vettore v) ne ritorna il valore della norma 2 (cioè $\sqrt{\sum_i v_i^2}$)

b) *diff* che prese in input due liste ritorna la lista che rappresenta il vettore della differenza dei due vettori dati

```
import math
def norma(v):
    return math.sqrt(sum([i**2 for i in v]))

def diff(v1,v2):
    return [v1[i]-v2[i] for i in range(len(v1))]

vec1, vec2 = [1,2,3], [0,4,2]
print norma(vec1),norma(vec2)
print diff(vec1,vec2)
#3.74165738677 4.472135955
#[1, -2, 1]
```

Esercizio 6

Definire la funzione *normalizzazione* che presa in input una lista (per noi un vettore v) ritorna la lista normalizzata rispetto alla norma 2, cioè la lista i cui elementi sono tutti divisi per la costante *norma(v)*

Esercizio 6

Definire la funzione *normalizzazione* che presa in input una lista (per noi un vettore v) ritorna la lista normalizzata rispetto alla norma 2, cioè la lista i cui elementi sono tutti divisi per la costante *norma(v)*

```
def normalizzazione(v):
    n = norma(v)
    return [i/n for i in v]

vec1 = [1,2,3]
print normalizzazione(vec1)
print norma(normalizzazione(vec1))
#[0.26726..., 0.53452..., 0.80178...]
#1.0
```

Esercizio 6

Partendo dalle seguenti linee di codice:

```
import random
N = 4
M = Matrice([[1,0,0,1],[0,1,2,0],[0,1,1,1],[0,1,0,0]])
vr = [random.random() for i in range(len(N))]
```

Implementare un algoritmo che: 1) faccia il prodotto della matrice M per il vettore vr; 2) normalizzi il vettore risultante; 3) ripeta le operazioni ripartendo da "1)" usando al posto di vr il vettore ottenuto dopo la normalizzazione.
L'algoritmo deve ripetere queste operazioni fintanto che la norma 2 della differenza tra il vettore del passo precedente e il vettore ottenuto non risulta essere inferiore a 1e-8.
Stampare infine il vettore risultante.

Esercizio 6

Soluzione:

```
import random
N = 4
M = Matrice([[1,0,0,1],[0,1,2,0],[0,1,1,1],[0,1,0,0]])
vr = [random.random() for i in range(N)]

vprec = [-42.0]*N
while norma(diff(vprec,vr)) > 1e-8:
    vprec = vr
    vr = normalizzazione(M.mul(vr))
print vr
#[0.165298..., 0.729155..., 0.604822..., 0.274224...]
```

Esercizio 7

Dato un Albero binario creare una funzione `nodi_l(albero,l)` che restituisce il numero di nodi presenti al livello `l` nell'albero.

Esercizio 7

Dato un Albero binario creare una funzione `nodi_l(albero,l)` che restituisce il numero di nodi presenti al livello `l` nell'albero.
Soluzione:

```
class Albero():
        def __init__(self, val=None, sx=None, dx=None):
                self.val = val
                self.sx = sx
                self.dx = dx

def nodi_l(albero,l):
        if (albero == None or l<0):
                return 0
        if l == 0:
                return 1
        return nodi_l(albero.dx, l-1) + nodi_l(albero.sx
            , l-1)
```

Esercizio 8

Creare una classe che gestisca una partita a $7\frac{1}{2}$. Regole base:

- si gioca con un mazzo di carte "all'italiana", ovvero 40 carte con 4 semi: bastoni, spade, denari e coppe;
- ogni carta vale quanto il numero che rappresenta ad eccezione delle figure (i.e., fante(8), cavallo(9) e re(10)) che valgono $\frac{1}{2}$;
- il gioco si svolge tra giocatore e banco: vince chi possiede come somma delle proprie carte il valore più vicino ($\leq$) al $7\frac{1}{2}$. Chi lo supera "sballa" e perde;
- il giocatore riceve carte fintanto che lo vuole o sballa; il banco (CPU) accetta carte fintanto che non ha un valore $\geq 5\frac{1}{2}$;

Esercizio 8

Viene richiesto di:

- creare una classe che gestisca il gioco per un numero di turni fissato;
- ogni turno deve:
  - assegnare una carta a banco e giocatore (il giocatore deve conoscere anche la carta del banco);
  - chiedere all'utente se vuole carta o si ferma, fino a che non si ferma o sballa;
  - successivamente, pesca le carte al banco fino a che non ha un valore $\geq 5\frac{1}{2}$.
- in caso di stessi punti (senza essere sballati) vince il banco; se sballano entrambi non c'è vincitore;
- finiti i turni: dire chi ha vinto più mani.