Examen scris la Algoritmică și Programare – 14.01.2020
Rândul 3

Nume și prenume: _____ Grupa: _____

Timp de lucru: **1 oră și 30 minute**. Fiecare subiect valorează **0.75 puncte**.

1. Completați spațiile subliniate astfel încât funcția de mai jos să returneze echivalentul expresiei list(range(a, b, k)). Presupunem a < b, k > 0 și a, b, k întregi.

```
def my_range(___, ___, ___):
        _________ = []
        while a ___ ___:
                result._________(a)
                a ___ k
        return _________
```

2. Scrieți un apel al funcției de la punctul 1 care ar returna lista [1, 5, 10, 15, 20, ..., 2020].

3. Rescrieți funcția de mai jos fără a folosi **range**:

```
def mat_sum(mat):
    result = 0
    for i in range(len(mat)):
        for j in range(len(mat[i])):
            result += mat[i][j]
    return result
```

4. Scrieți un apel al funcției de la punctul 3 care ar returna 2020.

5. Rescrieți funcția de la **punctul 3** folosind **o singură structură repetitivă**.

6. Scrieți o funcție care primește ca parametru o listă de matrici și returnează o listă cu suma elementelor fiecărei matrice. Folosiți funcția de la punctul 3 și **list comprehensions**.

7. Scrieți 3 aserțiuni pentru a testa funcția de la **punctul 6**.

8. Implementați funcția de mai jos conform specificațiilor:

```
def perechi(lst):
    # returnează o listă care conține, pentru fiecare întreg din lst, o pereche de întregi în care primul element este
    # puterea la care apare numărul 2 în întreg, iar al doilea element este puterea la care apare numărul 3.
    # Exemplu: perechi([5, 4, 18]) => [(0, 0), (2, 0), (1, 2)]
```

9. Completați specificațiile funcției de mai jos:

```
def sums(lista):
    # Returnează _____________________________________________________________________________
    #
    return list(sum(p) for p in perechi(lista) if sum(p) > 0)
```

10. Considerăm un algoritm de programare dinamică. Încercuiți toate afirmațiile **adevărate**:
    a. Va rezolva problema mai eficient decât o abordare backtracking
    b. O abordare Greedy ar genera o soluție mai corectă
    c. Poate fi implementat recursiv, cu memoizare
    d. Nu este necesar ca problema să prezinte substructură optimă
    e. Complexitatea va fi întotdeauna cel puțin pătratică

11. Specificați complexitatea ca timp și spațiu suplimentar pentru funcția de mai jos, folosind notațiile O, Theta, Omega:

```
def f(n):
    return map(lambda x: x*x, range(n))
```

Timp:	Spațiu:

12. Implementați o clasă **AgendaTelefonica** ce suportă **asocierea unei persoane cu un număr de telefon** și returnarea numărului de telefon al unei persoane.

13. Explicați, pe scurt, ce paradigmă de rezolvare a problemelor s-ar potrivi pentru rezolvarea următoarei probleme: afișarea tuturor posibilităților de a umple un rucsac de o anumită capacitate cu un număr maxim de obiecte de diferite dimensiuni.

14. Precizați trei algoritmi cunoscuți care au complexitatea timp $O(n^{2020})$:

15. Se dă o lista de tupluri formate din 3 elemente. Scrieți o secvență care creează trei liste: prima formată din toate primele elemente ale tuplurilor, a doua formată din toate elementele de pe poziția a doua a tuplurilor etc.