Obiective

- Cunoaşterea conceptelor fundamentale programării
- Introducere concepte de bază legate de ingineria software (design, arhitectură, implementare şi întreţinere)
- Înţelegerea instrumentelor software folosite în dezvoltarea de aplicaţii
- Învătarea limbajului Python şi utilizarea lui pentru implementarea, testarea, rularea, depanarea de programe.
- Însuşirea/Înbunătăţirea stilului de programare.

Conținut

Evaluare

Lab (30%) - o notă pe activitatea de laborator din timpul semestrului.
T (30%) - examen practic (în sesiune)
E (40%) - examen scris (în sesiune)
S 0 - 0.5p - bonus pe activitatea la seminar

Pentru promovare trebuie să aveți cel puțin nota 5 la toate (Lab,T,E >= 5)
Toate activiățile sunt obligatorii
Dacă nu opțineți nota 5 la laborator nu puteți intra in examen in sesiunea normală.

Restanțe
În sesiunea de restanțe puteți preda laboratoare dar nota maximă este 5.
Se poate re-susține examenul practic
Se poate re-susține examenul scris

Data examen

19 Ian gr: 212,213
20 Ian gr: 211,216
26 Ian gr: 215,217
2 Febr gr: 214

Data secundara de examen

19 Ian gr: 211,216,215
20 Ian gr: 212,213,217
26 Ian gr: 214

Pentru a participa la data secundara este nevoie de acordul prealabil (minim 48 ore inainte)

Elemente de bază Python:

- Instrucțiuni: =, if, while, for
- tipuri de date: integer, real, string, list, dictionary
- funcții: definiție, transmiterea de parameteri
- tipuri definite de utilizator: classe
- excepţii

1) Care este rezultatul execuţiei pentru următorul cod Python:

```
def f(l):
    print "A"
    if l==[]:
        raise ValueError()
    print "B"

def start():
    l = []
    try:
        print "A"
        f(l)
        print "D"
    except ValueError:
        print "C"
start()
```

2)

```
class A:
    def f(self, l,nr):
        l.append(nr)
class B:
    def g(self, l, nr):
        nr=nr-1
        l = l+[-2]

a = A()
b = B()
l = [1,2]

c = -1
a.f(l,6)
b.g(l,c)
print l,c
```

Algoritmi - specificații/test/implementare

Variante

Se dă specificația - implementati și testați

Se dă implementare – specificați și testați

Se dă o funcție de test: specificați și implementați

Exemple:

Implementați și testați funcția care are specificația

```
"""
   Calculeaza suma elementelor pare
   l - lista de numere
   return un numar, suma elementelor pare
   aruncă ValueError daca lista nu conține numere pare
"""
```

Specificați și testați funcția:

```
def f(n):
    d = 2
    while (d<n-1) and n%d>0: d+=1
    return d>=n-1
```

Complexitate

Se dă o funcție pyton – se cere să analizați complexitatea ca timp si/sau spațiu de memorie

Exemple

Analizați complexitatea ca timp de execuție pentru următoarea funcție:

1)

```
def f(x):
    m = len(x)
    found = False
    while m>=1:
        c = m - m/3*3
        if c==1: found=True
        m = m/3
```

2)

```
def f(x):
    found = False
    n = len(x)-1
    while n!=0 and not found:
        if x[n]==7: found=True
        else: n=n-1
    return found
```

2)

```
def prel(x,i,j):
    if (i<j-1):
        k1 = (j-i+1)/3
        k2 = 2*k1
        prel(x,i,i+k1)
        prel(x,i+k1+1,i+k2)
        prel(x,i+k2+1,j)
        for k in range(i,j):
            print x[k]
    else:
        print x[i]
```

Căutări/sortări complexitatea lor

Implementați una din funcțiile de cautare/sortare studiate

- cautare:
  - secvențială
  - succesivă
  - binară
- soratări:
  - sortare prin selecție
  - sortare prin selecție directă
  - sortare prin inserție
  - metoda bulelor
  - quicksort
  - mergesort

Exemple:
1 Scrieți o funcție care sortează o listă de numere și are complexitatea ca timp de execuție în caz defavorabil n^2

2) O listă de cumpărături conține produse:

Produs = Product name, Product type, Price

Scrieți o funcție de sortare care permite sortarea listei de cumpărături:

- alfabetic dupa tip
- descrescător dupa preț

Technici de programare

- Backtracking
- Divide et. Impera
- Greedy
- Dynamic Programming

Se dă o problemă – se cere rezolvarea folosind o metoda dată

Alegeți cea mai potrivită metodă de rezolvare pentru o problemă dată

Se poate cere:

- indicați schematic soluția
    - backtracking: soluție candidat, valid, soluție
    - Divide et impera: recurența
    - Greedy: soluție candidat, valid, soluție, funcția de selecție greedy
    - Dynamic Programming: principiul optimalității, recurența care descrie algoritmul

implementare pentru elementele de bază sau implementare pentru tot (alg, elemente de bază)

Exemplu:

1 Se da o lista de numere, să se determine cel mai lung subsir cu elemente pare folosind programare dinamică.

2 Alegeți technica cea mai potrivită (Backtracking, Divide et. Impera, Greedy, Dynamic Programming) pentru a calcula suma numerelor pare intr-o lista.

Examen practic

Scrieti o aplicație pentru gestiunea activităților de laborator

Aplicația permite cadrului didactic sa efectueze repetat operațiile:
- vizualizare listă studenți
-căutare student după id
-assignare laborator la un student
- aplicația semnalează (și nu permite) adăugarea de două probleme la același laboratoar (lab number)
- vizualizare toate laboratoare de la un student
- vizualizare studenti si laboratoare asignate pentru un laborator dat

Studenții și laboratoarele sunt stochate in 2 fisiere: “student.txt”, “labs.txt”

Arhitectura aplicației: