Introducere în Python

1.1 Noţiuni de bază

Principalele caracteristici ale limbajului Python sunt:

1. este un limbaj de nivel înalt;
2. are o sintaxă simplă;
3. este open source;
4. există implementate numeroase biblioteci pentru Python:
   - · pachetul standard *Python* – este un ansamblul de biblioteci standard care oferă foarte multe funcţionalităţi;
   - · biblioteca *NumPy* – oferă numeroase funcţionalităţi pentru calcul numeric;
   - · biblioteca *SciPy* – oferă rutine specifice pentru rezolvarea de ecuaţii, integrări, ecuaţii diferenţiale;
   - · *PyGame* – se utilizează pentru dezvoltarea de interfeţe grafice şi evenimente;
   - · *PIL (Python Image Library)* – se utilizează pentru manipularea şi procesarea de imagini;
   - · *Biopython* este un utilitar foarte puternic care permite citirea multor tipuri de fişiere utilizate în bioinformatică. De asemenea, asigură accesul la cele mai utilizate baze de date locale sau pe Internet.
5. este un limbaj interpretat;
6. este un limbaj uşor de învăţat şi de utilizat;
7. este un limbaj preferat în diverse comunităţi ştiintifice, academice şi de cercetare;
8. este flexibil la numeroase paradigme de programare: programare imperativă, programare funcţională, programare orientată pe obiecte;
9. este un limbaj complet dinamic şi are management automat al memoriei (garbage collector);
10. blocurile sunt delimitate prin indentare;
11. este un limbaj case sensitive;
12. codul sursă este scurt, uşor de testat, de depanat şi de întreţinut;
13. permite concentrarea efortului de dezvoltare pe problemă şi nu pe detaliile de implementare ale limbajului de programare;
14. codul sursă este independent de platformă;
15. indexarea începe de la 0.

Adrese utile şi instrucţiuni de instalare Python

- Site oficial: http://www.python.org/
- Interpretor Python: http://www.python.org/download/releases/
- În cadrul acestui laborator se va utiliza versiunea Python 3.3.0
- Biblioteca *NumPy*: http://sourceforge.net/projects/numpy/files/
- Biblioteca *SciPy*: http://sourceforge.net/projects/scipy/files/
- Biblioteca *Biopython*: http://biopython.org

- Plugin PyDev pentru Eclipse: http://pydev.org/

- Mediul de dezvoltare Eclipse Clasic v.4.2.1: http://www.eclipse.org/downloads/

1.2 Formatare şi sintaxă

1. **Atenţie!** Limbajul Python nu utilizează delimitatori pentru evidenţierea unor blocuri de cod. Modalitatea prin care se delimitează un bloc de cod în Python este **indentarea**.

   Exemplu:
   ```
   x = 1
   if x:
           y = 2
           if y:
                   print('bloc2')
           print('bloc1')
   print('bloc0')
   ```

2. O instrucţiune poate fi continuată pe rândul următor folosind caracterul *'\n'*
3. Pentru a comenta o singură linie de cod se foloseşte caracterul *'#'*
4. Pentru a comenta mai multe linii de cod, se încadrează textul de comentat între 3 ghilimele simple sau duble
5. În Python nu este necesară semnalarea sfârşitului unei linii prin utilizarea caracterului *';'* .

   Exemple:
   ```
   #  acesta este un comentariu

   a = 3   # variabila a este iniţializată cu valoarea 3

   """
   Acesta este
   un comentariu
    pe mai multe linii
   """

   '''
   Acesta este
   alt comentariu
   pe mai multe linii
   '''

   print(1+2\
   +3)
   ```

1.3 Instrucţiunea *print*

1. Este o instrucţiune built-in a limbajului
2. Se foloseşte pentru afişarea unor parametrii
3. Sintaxa instrucţiunii:
   print (valoare, valoare, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

4. Instrucţiunea *print* separă parametrii de afişat prin spaţiu şi impune automat caracterul *'\n'* (linie nouă) la sfârşitul afişării.

Exemple:

```
print('Hello world!')
print("Hello world!")
print('Hello "world!" ')
print('Hello' world')
print('Hello\
      world')
```

Exemplu: Ce afişează următorul cod?

```
a = 3
b =18.5
c = 7

print ("Valorile variabilelor a si b sunt", a, \
                'respectiv', b)
print (a+c)
print (c/a)
print (c%a)
print (a**3)    #ridicare la putere
```

1.4 Numere

1. Spre deosebire de alte limbaje, în Python nu trebuie să declarăm tipul variabilelor utilizate. Variabilele vor fi create atunci când li se atribuie o valoare pentru prima dată, moment în care variabilele dobândesc tipul valorii primite.
2. Tipurile de variabile numerice întâlnite în Python sunt:
   - Numere întregi şi numere cu virgulă mobilă: *1401* sau *2.24*
   - Numere complexe: *4-11j*
   - Numere cu zecimale cu precizie fixă: *Decimal(0.02)*
   - Numere raţionale: *Fraction(2, 5)*
   - Seturi
   - Booleeni
   - Numere întregi cu precizie nelimitată
   - Numere binare, octale sau hexazecimale: *0b11010, 0o6443, 0x23AE*
3. Ca orice alt limbaj de programare, Python utilizează precedenţa operatorilor.

4. Spre deosebire de alte limbaje, în Python sunt permise operaţii între numere de tipuri diferite, interpretorul Python realizând conversia automată către tipul cel mai complex
   Exemplu: *11 + 6 + 2.42 + 3.01*
5. Pentru lucrul cu numere se pot utiliza şi librăriile *random*, *math*, etc.

Exemple:

```
a = 3
b = 4

print(a+1, a-1)
print(b*3)
print(b/2)
print(a/2)
print(a//2)
print(a%2)

print(a+2.2)
print(b ** 3)

print(b/2+a)

print(1<2)
print(1>2)

print(0b1001)
print(0o47)
print(0x2F)

print(bin(9))
print(oct(39))
print(hex(47))

print(2+1j)
print((3-2j)*2)

x = 1
print(x<<2)
```

1.5 Şiruri de caractere (*string*)

1. Şirurile de caractere sunt obiecte imuabile în Python (i.e. elementele şirului nu pot fi modificat ulterior declarării lor)
2. Sunt vectori de caractere, care pornesc de la indicele 0.
3. Un *string* are elemente de la indicele 0 la indicele *len(string)-1*
4. Numărul de caractere dintr-un string se obţine cu funcţia *len()*

Exemple:

```
sir1 = 'Acesta este un sir.'
```

```
sir2 = 'Sirurile sunt imuabile.'
print(sir1+sir2)

sir2[0] = "X"
print(sir2)

print(len(sir1))

print(sir2[-2])
print(sir2[-3])

print(sir1[7:])
print(sir1[0:7])
```

1.6 Liste

1. Listele sunt colecţii de obiecte ordonate poziţional ce pot fi modificate pe parcurs.
2. Listele sunt asemănătoare vectorilor din alte limbaje (C, Java, etc.)
3. O lista Python poate conţine obiecte de diverse tipuri: numere, şiruri, tupluri sau alte liste.
4. Listele sunt demarcate prin paranteze pătrate: *L = [1, 2]*
5. Ca şi string-urile, listele au elemente între indicii 0 şi *len(listă)-1*

Exemplu: Să se ruleze următorul cod. Care va fi valoarea variabilei *p*?

```
m=[[1,2,3], [4,5,6], [7,8,9]]

print(m[0][2])

m[0][2] = 0
print(m[0])

p = m[0] * 3
print(p)
```

Exemplu: Ce afişează codul de mai jos?

```
n = 4
m = [[0]]*n
print(m)

m[0].append(1)
print(m)

m[0].pop()
print(m)
```

Exemple:

```
print(['a']+['s'])
print('a' in ['b', 'd', 'a', 'c'])
print('a' in ['b', 'aa', 'c'])
print('b' not in ['q', 'w', 'e'])
```

Exemplu: Ce afişează următorul cod?

```
a=[1,2,3,4,5]
print(a)
print(a[::2])
print(a[-2])
print(a[::-1])
```

Exemplu: Sortarea unei liste se poate face folosind fie funcţia built-in *sorted*, fie funcţia *sort*, care aparţine obiectelor de tip listă

```
lista1 = sorted([3,2,1,9,7,10])
print(lista1)

lista2 = [9,2,5,12,6]
lista2.sort()
print(lista2)
```

1.7 Tupluri

1. Tuplurile sunt liste imuabile (i.e. elementele unui tuplu nu pot fi modificate ulterior declarării lor).
2. Un tuplu este asemănător unei liste, cu excepţia faptului că nu îi pot fi modificate elementele.
3. Tuplurile suportă aproape toate operaţiile unei liste.
4. Tuplurile sunt demarcate prin paranteze rotunde: *T = (1, 2, 'tuplu')*

Exemplu: Să se testeze codul următor şi să se şteargă liniile care produc erori. Ce va fi afişat?

```
tuplu = ('b', '', 'f', 'k')
print(tuplu)

tuplu[2] = 'modificare'
tuplu = tuplu[0:2] + (5,6,7)
print(tuplu)

print((1,2,3)*3)
```

1.8 Dicţionare

1. Dicţionarele, numite şi *hashmaps*, sunt liste de perechi cheie: valoare
2. Un dicţionar este o listă modificabilă (i.e. valorile lui pot fi modificate ulterior declarării lor).
3. Dicţionarele sunt demarcate prin acolade: *D = {cheie: 21}*
4. Ca şi listele, dicţionarele pot conţine orice tip de valoare (numere, şiruri, liste, dicţionare, etc.)
5. Chiar dacă dicţionarele în sine sunt modificabile, cheile unui dicţionar sunt unice şi imuabile.
6. Ca şi o listă, un dicţionar permite imbricarea pe mai multe niveluri

Exemplu:

```
h = {"a": 1, 10: "x"}
print(h)
print(h["a"], h[10], sep=', ')
print('a' in h)
print('x' in h)

h["e"] = 7
print(h)

h['e'] = 'sapte'
print(h)
print(len(h))
```

1.9 Citirea şi scrierea în fişiere

1. Ca şi alte limbaje, Python permite lucrul cu fişiere externe.
2. Funcţia de citire din fişier este: *variabila = open('fişier', mod_de_deschidere)*
3. Modalitatea de deschidere a unui fişiere poate fi *'w'* atunci când vrem să scriem în fişier sau *'r'* atunci când doar vrem să citim din fişier (aceasta din urmă este opţiunea standard a funcţiei de apel *open*).
4. Funcţiile de citire şi scriere în fişier sunt *variabila.read() şi variabilă.write()*.
5. Un fişier se închide prin comanda *variabila.close()*.

Exemplu:

```
f = open('myfile3.txt', 'w')

f.write('test1 ')
f.close()

f = open('myfile3.txt', 'a')
f.write('test2')
f.close()
```

```
f = open('myfile3.txt', 'r')
print(f.read())
f.close()
```

Exemplu:

```
f = open('myfile3.txt', 'a')
f.write('\ntest3')
f.close()

f = open('myfile3.txt', 'r')
print(f.readlines())
f.close()

f = open('myfile3.txt', 'r')
print(f.readline())
print(f.readline())
f.close()
```