1. Introducere în Python

Limbajul de programare Python
Instalarea SDK-ului Python 2.7
Sintaxa Python
Modulele Python

Scopul acestei lucrări de laborator este acela de a introduce principalele elemente ale limbajului de programare *Python* [**?**], dintre care remarcăm: descrierea limbajului, câteva particularități ale acestuia, sintaxa, precum și extinderea limbajului prin anumite module (biblioteci). Python este un limbaj de programare de nivel înalt, bazat pe interpretare, care favorizează lizibilitatea codului, în detrimentul vitezei de execuție. Suportă mai multe paradigme, dintre care programarea orientată pe obiecte, imperativă, sau funcțională. Este un limbaj de programare cross-platform, având versiuni disponibile pentru Windows, Linux sau OSX, iar codul sursă este open source. Datorită sintaxei mai simple decât al altor limbaje de programare (ex. C++, Java, etc.), Python se folosește deseori în crearea de prototipuri rapide pentru diverse aplicații. Unul dintre domeniile în care se folosește acest limbaj este cel de Machine Learning, unde algoritmii moderni pot fi implementați pentru început în Python, ulterior urmând ca aceștia să fie portați în alte limbaje mai rapide, precum C++.

Versiunile folosite în mod curent sunt 2.7.x, respectiv 3.x.x. Versiunea stabilă curentă, la data publicării acestui îndrumar, este 3.6.0. Python 2.7.x este considerat *legacy*, noile funcționalități ale limbajului ne mai fiind suportate peste această versiune.

1.1 Instalarea SDK-ului Python

Pentru început, SDK-ul Python se descărcă de la adresa `https://www.python.org/downloads/`. Pasul următor este de instalare, având grijă ca opțiunea de adăugare a executabilul în variabila de sistem *Path* să fie bifată. SDK-ul vine cu un mediu de dezvoltare integrat, denumit IDLE. Dacă calea către executabilul python.exe a fost adăugată în *Path*, IDLE se poate porni deschizând Command Prompt și scriind "python". În consolă apar astfel informații despre versiunea de Python instalată. Pentru a închide IDLE, se poate executa comanda `exit()`.

Programatorul poate scrie un script Python direct în IDLE, interpretorul rulând script-ul pas cu pas, sau poate crea un fișier de tipul python script (care are extensia .py) și îl poate rula din linia de comandă folosind comanda `python sript.py`.

1.2 Sintaxa

În continuare, se prezintă câteva elemente specifice de sintaxă ale limbajului de programare Python.

În primul rând, blocurile de instrucțiuni se construiesc folosind *indentări*: fie tab-uri, fie spații libere (nu se mai folosesc acolade, ca în alte limbaje de programare). În al doilea rând, pentru variabile, tipul de date nu se specifică explicit. Acesta se determină automat, la rularea programului. De exemplu, dacă vrem să folosim un şir de lungime 3, format din numerele 1, 2 și 3, vom utiliza următorul format: `sir` = [1, 2, 3].

Un alt aspect important este prezența așa-numitului *Garbage Collector*, care determină automat dacă o variabilă nu mai este folosită și eliberează memoria ocupată de aceasta, fără a fi nevoie de intervenția programatorului (similar cu limbajul Java).

1.2.1 Tipuri de date

Tipurile de date prezente în Python se pot grupa în tipuri numerice, secvențe, seturi, tipuri mapate ș.a. Principalele tipuri de date folosite sunt:

- boolean - ia valorile `True` sau `False`; acest tip de date este de cele mai multe ori interschimbabil cu întregii 0 sau 1; valoarea `False` este echivalentă, de asemenea, și cu string-ul gol `""`;
- întregi: int (numere întregi), long (doar în Python 2.x), float (echivalentul tipului double din C) și complex;
- str - string reprezentat ca un șir de caractere pe 8 biți în Python 2.x şi o secvență Unicode în Python 3.x;
- bytes - șir de întregi între 0 și 255;
- listă - listă care poate stoca orice tip de date; ex: `l` = [1, 2, 3];
- tuplu - similară cu lista, doar că din punct de vedere principial se folosește pentru a stoca tipuri diferite de date; ex: `t` = (`'pret'`, 15000).

1.2.2 Afișarea în consolă

```
1 sir = [0, 1, 2, 3]
2 print(sir)
```

Observație: pentru a se putea printa un șir, nu este nevoie de parcurgerea acestuia.

1.2.3 Printarea formatată

```
1 sir = [0.123, 1.021, 2.3212, 3.0]
2 print("Valoarea a doua a sirului este: {0}".format(sir[1]))
3 print("Valoarea a doua a sirului este: {0:.2f},
4       iar a treia este: {1:.2f}".format(sir[1], sir[2]))
```

Observație: în versiunea de Python 2.7.x, instrucțiunea `print` se poate folosi și în felul următor: `print "Text"`. În noua versiune însă este corectă doar prima variantă: `print("Text")`.

1.2.4 Instrucțiunea condițională *if*

```
if cond1:
    instructiuni
    ...
elif cond2:
    instructiuni
    ...
else:
    instructiuni
    ...
```

1.2.5 Instrucțiuni repetitive

1.2.5.1 Instrucțiunea for *varianta 1:*

```
sir = [0, 1, 2, 3]
for i in range(0, len(sir)):
    sir[i] += 1
print(sir)
```

1.2.5.2 Instrucțiunea for *varianta 2:*

```
sir = [0, 1, 2, 3]
for i in sir:
    i += 1
print(sir)
```

1.2.5.3 Instrucțiunea while*:*

```
sir = [0, 1, 2, 3]
i = 0
while i < len(sir):
    sir[i] += 1
    i += 1
print(sir)
```

1.2.6 Corpul unei funcții Python

```
def nume_functie(param1, ...):
    instructiuni
    ...
```

1.2.7 Corpul unei clase în Python

```
class NumeClasa:
    """
    Comentariu pe multiple linii; se foloseste de obicei pentru
    a descrie functionalitatea clasei / metodei
    """
    def __init__(self, param1, ...):
        instructiuni
        ...

    # Comentariu pe o singura linie
    def functie_1(self, param1, ...):
        instructiuni
        ...

    def functie_2(self, param1, ...):
        instructiuni
        ...

    @staticmethod
    def functie_statica(param1, ...):
        instructiuni
        ...
```

În acest exemplu se poate observa comentariul pe multiple linii, comentariul pe o singura linie, definiția a două funcții membre, o funcție statică și funcția `__init__`. Funcția `__init__` este echivalentul din Python pentru constructorul din alte limbaje de programare orientate pe obiecte. Așadar, aici este locul pentru a inițializa variabilele și a executa rutine necesare la crearea obiectului. Se mai poate observa și faptul că toate funcțiile membre au ca prim parametru cuvântul-cheie `self`.

Observații:

- în Python, funcțiile pot returna oricâte valori. De exemplu, dacă în interiorul funcției se operează asupra a două variabile, ambele se pot returna:

```
def functie(a, b):
    c = a + 1
```

```
3       d = b + 2
4       return c, d
5    z = functie(1, 2) # z => tuplu care contine (2, 3)
```

În acest fel, funcția va returna un tuplu format din c și d;

- parametrii funcțiilor în Python se transmit prin referință, așadar nu se fac copii locale inutile;
- funcțiile statice nu pot accesa implicit variabilele membre ale clasei.

1.3 Module în Python

Python are un mare avantaj față de alte limbaje de programare prin modulele suplimentare care pot fi folosite pentru dezvoltarea aplicațiilor și care cuprind foarte multe funcționalități: module proiectate pentru a efectua calcule matematice avansate; parsere de text; module specializate pentru a efectua operații asupra bazelor de date; funcționalități pentru construirea aplicațiilor cu interfață grafică, etc.

Unele dintre aceste module se găsesc în librăria standard Python, fiind disponibile programatorului odată cu instalarea distribuției. Altele pot fi instalate folosind utilitare de instalare. Un asemenea utilitar este *Pip*. Acesta este un *package manager* care începând cu versiunea 2.7.9, respectiv 3.4, vine preinstalat cu distribuția de Python. Este un script care descarcă arhiva modulului ce se dorește a fi instalat, dezarhivând-o în folder-ul pentru pachete al distribuției (de exemplu C:/Python36/Lib/site-packages). Dacă utilitarul nu este preinstalat, acesta se poate descărca de pe internet apelând script-ul "get-pip.py". Pip se utilizează din linia de comandă folosind sintaxa `pip install nume_librarie`, sau `pip --help` pentru a afișa opțiunile disponibile.

Atunci când se scrie un program care necesită un modul adițional, acesta se include în program folosind sintaxa `import nume_modul` sau `import nume_modul as nume_prescurat`. Ulterior, când se dorește să se folosească o anumită funcție din modulul încărcat, funcţia se apelează în felul următor: `nume_modul.nume_functie(param)` sau `nume_prescurtat.nume_functie(param)`.

1.3.1 Modulul NumPy

NumPy este un pachet foarte des folosit pentru calcule științifice avansate. Conține, printre altele, obiecte de tip array multidimensional, utile pentru integrarea codului de C/C++, funcții de algebră liniară, transformări Fourier, un generator de numere pseudo-aleatore, etc. Se poate instala din linia de comandă folosind *Pip*: `pip install numpy`.

1.3.1.1 Exemplu

```
1 import numpy as np
2 a = np.array([[1, 2, 3],
```

```
3                 [4, 5, 6],
4                 [7, 8, 9],
5                 [3, 2, 1]])
6 b = np.ones((3, 4))
7 c = np.dot(a, b)
8 d = np.subtract(a.T, b)
```

În acest exemplu am folosit câteva elemente din modulul NumPy și anume: am creat un array de tip numpy.ndarray (numpy.array), având dimensiunile 4 * 3, am creat apoi un array cu dimensiunile 3 * 4 și l-am inițializat cu valorile 1 folosind funcția numpy.ones, am înmulțit cele două matrice folosind funcția numpy.dot și le-am scăzut folosind funcția numpy.subtract. Documentația modulului NumPy se găsește la adresa `http://www.numpy.org`.

1.4 Cerinţe

1. Să se implementeze căutarea binară;
2. Să se implementeze o funcție care returnează primele n elemente ale secvenței fibonacci;
3. Să se implementeze o funcție care calculează produsul elementelor unui șir;
4. Să se implementeze o funcție care întoarce inversul unui șir;
5. Să se implementeze o funcție care întoarce True dacă un șir este palindrom, respectiv False dacă nu.