






Python Course

Course 1

First things first

-  Grupul de Facebook
 - Materiale și anunțuri
-  Luni și Joi, ora 18:00
 - Sala EG202
-   Ask questions anytime
-  Feedback

Python

Specificații de limbaj

- Interpretat
- Independent de platformă (cod sursă -> bytecode -> sistem de operare).
- Open Source
- High Level
- Memory Management automat
- Tipat Dinamic

Python - Interpretor

- Ultima versiune: 3.7.2 -> versiunea de la laborator
- Există două seturi de interpretoare, 2.x și 3.x.
- 3.x a fost scris pe baza greșelilor de design din 2.x
- Interpretorul folosește module ca sistem de reutilizare a codului
- Are propriul manager de module, pentru download și versionare: pip.
- Este backwards compatible cu C-ul; poți trece cod python în C, poți compila C pentru python.

Python – Independent de platformă

- Orice sursă python este trecută într-un format intermediar, bytecode
- Bytecode-ul este trecut în format nativ pentru fiecare OS/procesor
- Python-ul este modulul high level, pentru detalii de arhitectură referitor la memory management -> CPython
- Intuitiv, OS-ul știe câte referințe ai către o zonă de memorie, dacă numărul de referințe devine 0, CPython-ul folosește free-ul din malloc/jemalloc

Python – Tipat dinamic

- O asignare a unei variabile leagă un nume la un obiect
- Interpretorul își dă seama la runtime de tipul fiecărui obiect “

```
>>> a = 10
>>> type(a)
<class 'int'>
>>> type(10)
<class 'int'>
```

```
>>> a = "Dynamic Type"
>>> type(a)
<class 'str'>
>>> type("Dynamic Type")
<class 'str'>
```

```
>>> a = [1, 2, 3, 4]
>>> type(a)
<class 'list'>
>>> type([1, 2, 3, 4])
<class 'list'>
```

HandsOnPython

- Entry point-ul în program este la începutul fișierului, fiind interpretat
- Contează precedența

```
1 #merge
2 a = 10
3 b = a + 2
4
```

```
1 #merge
2 def func1():
3     None
4
5 a = func1()
6
7 def func2():
8     None
```

```
1 #nu merge
2 b = a + 2
3 a = 10
4
```

```
1 #nu merge
2 def func2():
3     None
4
5 a = func1()
6
7 def func1():
8     None
```


HandsOnPython: Scopes 1

- Contextul este definit de tab-uri în python

```
1  if a == b:  
2      #aici incepe branch-ul functiei if  
3      print(a + b)  
4      #inca este in branch-ul functiei if  
5  #aici se termina corpul functiei if
```

- Tab-ul este echivalent acoladelor din C/C++

HandsOnPython: Scopes 2

- Instrucțiune de tip if cu branch-uri multiple și scope-urile lor

```

1  if a == b:
2      #aici incepe primul branch
3      print(a + b)
4      #aici se termina primul branch
5  elif b == c:
6      #aici incepe al doilea branch
7      print(b + c)
8      #aici se termina al doilea branch
9  elif c == d:
10     #aici incepe al treilea branch
11     print(c + d)
12     #aici se termina al treilea branch
13 #am terminat branching-ul
..

```

HandsOnPython – While statement

```
1  contor = 0
2  while contor < 9:
3      print("Contorul este: ", contor)
4      contor = contor + 1
5  print("Bbye!")
6
```

```
Contorul este: 0
Contorul este: 1
Contorul este: 2
Contorul este: 3
Contorul este: 4
Contorul este: 5
Contorul este: 6
Contorul este: 7
Contorul este: 8
Bbye!
ok
>>>
```

HandsOnPython – Handling Strings 1

- Reprezentare unicode a string-urilor în memorie
- Imutabile, fiecare string nou este reprezentat ca un obiect nou în memorie

```
1 t = "Ce jmek e py!"  
2 t[0] = "M"
```

- Variabila t va ține minte unde se află în memorie obiectul; cum string-urile sunt imutabile în Python, linia 2 va ridica următoarea excepție:

```
Traceback (most recent call last):  
  File "/home/python_ccna/test.py", line 2, in <module>  
    t[0] = "M"  
TypeError: 'str' object does not support item assignment  
>>>
```

HandsOnPython – Handling Strings 2

- De ce avem nevoie de imutabilitate?

```

1  statement = "Am fost la Poli si mi-a placut!"
2  new_statement = statement[:19] + 'nu mi-a placut'
3
4  print(statement)
5  print(new_statement)
6
7  for i in range(17, 21):
8      print(id(statement[i]))
9      print(id(new_statement[i]))
10     print()

```

HandsOnPython – Lists 1

- Listele sunt modul uzual de a reține date în python; față de alte limbaje tipate static, în cadrul unei liste poți avea mai multe tipuri

```
1 list_example_v1 = ["string1", "string2", 100, 200]
2 list_example_v2 = [list_example_v1, "string3"]
```

HandsOnPython – Lists 2

- Care este output-ul următorului snippet de cod?

```
1  matrix = []
2  height = 2
3  width = 3
4
5  for i in range(0, height):
6      new_line = []
7      for j in range(0, width):
8          new_line.append(i + j)
9      matrix.append(new_line)
```


HandsOnPython – For statement

- For-ul funcționează prin iterarea unei liste

```
1  #vrem sa generam toate numerele de la 0 la 9
2  index_list = range(0, 10)
3  #index_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
4
5  #for-ul va lua in ordine fiecare element din lista
6  for index in index_list:
7      print(index)
```


HandsOnPython - Tuples

- Tuplurile sunt o secvență de elemente
- Care este diferența între tupluri și liste?

```

1  #tuplu vs lista
2
3  a = (1, 2, "Tudy")
4  b = [1, 2, "Tudy"]
5  print(a)
6  print(b)
7  b[0] = 2
8  a[0] = 2

```

HandsOnPython – Dicționare

- Asocieri de tip cheie valoare. De ce folosim tuplu-uri în cadrul dicționarelor?

```

1  #exemplu de dictionar
2
3  dict_v1 = {'Nume:' : 'Preda', 'Prenume:' : 'Andreea', 'Age' : 24}
4
5  for k in dict_v1:
6      print(k)
7
8  #putem intoarce si tupluri de cheie valoare
9  for k,v in dict_v1.items():
10     print(k)
11     print(v)
12
13  #putem crea dictionare din tuplu-ri
14  dict_v2 = dict([('Nume:' , 'Preda'), ('Prenume:' 'Andreea')])

```

Python – Import modules

- Sistemul de modularizare și reutilizare a codului folosit de python

```

1  #will import everything
2  from os import getcwd
3  from os import uname
4
5  print(getcwd())
6  print(uname())

```

```

1  #will import everything
2  import os
3
4  print(os.getcwd())
5  print(os.uname())
6

```



Întrebări?