Fundamentele Programării

Curs 2

Programarea procedurală

- Tipuri de date secvențiale (liste, tuple, string)
- Dictionare
- Ce este o funcție?
- Cum scriem funcții?

Liste – mutabile `(mutable)`

- Domeniu: secvențe de elemente (similare sau diferite ca tip) separate prin „,” și încadrate de „[ ]”
- Operații
  - Creare (manuală, *range*)
  - Accesare (index, *len*) și modificare elemente
  - Eliminare (*pop*) și inserție (*insert*) de elemente
  - Feliere (Slicing) și încapsulare
  - Utilizare ca stive (*append, pop*)

Liste- operații, exemple

```
# create
a = [1, 2, 'a']
print (a)
x, y, z = a
print(x, y, z)
# indices: 0, 1, ..., len(a) - 1
print a[0]
print ('last element=', a[len(a)-1])
# lists are mutable
a[1] = 3
print a
```

```
# slicing
print(a[:2])
b = a[:]
print(b)
b[1] = 5
print(b)
a[3:] = [7, 9]
print(a)
a[:0] = [-1]
print(a)
a[0:2] = [-10, 10]
print(a)
```

```
#generate lists
using range
l1 = range(10)
print(l1)
l2 = range(0,10)
print(l2)
l3 = range(0,10,2)
print(l3)
l4 = range(9,0,-1)
print(l4)
```

```
#list in a for loop
l = range(0,10)
for i in l:
    print(i)
```

```
# lists as stacks
stack = [1, 2, 3]
stack.append(4)
print(stack)
print(stack.pop())
print(stack)
```

```
# nesting
b=[4,5]
c = [1, b, 9]
print (c))
```

Stringuri – ne-mutabile (immutable)

- Domeniu: șiruri de caractere
- Operații: concatentare, căutare …
- Literali: „abc” ,’abc’

```
#concatenate
a = "abc"
b = "xybc"
c = a + b
print(c)


#immutable
c[0]='b' #eroare
```

```
#search
n = c.find("bc")
print(n)
# n = 1
m = c.rfind("bc")
print(m) # m = 5

print(len(c))
```

• Tuple – ne-mutabile (immutable)

- Domeniu: secvențe de valori (similare sau diferite ca tip) separate prin „,”
- Operații
  - Creare (manuală – împachetare) și despachetare
  - Încapsulare
  - Tuplu cu 0 elemente și tuplu cu un singur element

```
# A tuple consists of a
number of values separated
by commas
# tuple packing
t = (12, 21, 'ab')
print(t[0])

# empty tuple (0 items)
empty = ()

# sequence unpacking
x, y, z = t
print (x, y, z)
```

```
# tuple with one item
singleton = (12,)
print (singleton)
print (len(singleton))

#tuple in a for
t = 1,2,3
for el in t:
    print (el)

# Tuples may be nested
u = t, (23, 32)
print(u)
```

Dicționare

- Domeniu: mulțimi ne-ordonate de perechi (cheie, valoare) cu chei unice
- Operații:
  - Creare
  - Accesarea valorii pentru o cheie dată
  - Adăugarea/modificarea/eliminarea unei perechi (cheie, valoare)
  - Verificarea existenței unie chei

```
#create a dictionary
a = {'num': 1, 'denom': 2}
print(a)
#get a value for a key
print(a['num'])

#set a value for a key
a['num'] = 3
print(a)
print(a['num'])
```

```
#delete a key value pair
del a['num']
print (a)

#check for a key
if 'denom' in a:
   print('denom = ', a['denom'])
if 'num' in a:
   print('num = ', a['num'])
```

Programare procedurală –Funcții

Programare procedurală

- un program este alcătuit din mai multe proceduri (**subrutine sau funcții**)

Funcție - Un bloc de instrucțiuni de sine stătător care:

- are un nume
- poate avea o listă de parametri (formali)
- poate returna o valoare
- are un corp (format din instrucțiuni)
- are o documentație (specificare) alcătuită din:
  - o scurtă descriere
  - tipul și descrierea parametrilor
  - condiții impuse parametrilor de intrare (precondiții)
  - tipul și descrierea rezultatelor (valorilor returnate)
  - condiții impuse rezultatelor (postcondiții)
  - excepții care pot să apară în execuția ei

Funcții in Python

- Definiția unei funcții
  - instrucțiune executabilă introdusă prin cuvântul rezervat def
  - Nu execută corpul funcției (execuția se produce doar la apelul funcției)

```
Descrieti functia de mai jos!!!

def x(n):
    k=0
    y=2
    while y<=n//2:
        if (n%y==0):
            k=k+1
        y=y+1
    return (k==0)
```

Funcții in Python

- Definiția unei funcții
  - instrucțiune executabilă introdusă prin cuvântul rezervat def
  - Nu execută corpul funcției (execuția se produce doar la apelul funcției)

```
def is_prime(n):
    '''
       verifica daca un nr e prim
       parametrii: n
       preconditii: n-nr natural
       returneaza: True sau False
       Postconditii: True - daca n e prim, False altfel
    '''

    prim=True #presupunem ca numarul este prim
    if n==0 or n==1: prim=False
    for i in range (2,n//2+1): #cautam divizori in multime 2,3,...n/2
        if n%i==0: #daca i este divizor a lui n
            prim=False #nr nu e prim
    return prim
```

Funcții în Python

- Apelul unei funcții
  - RE: bloc = parte a unui program Python (delimitată prin indentare) care este executat ca o unitate
  - Corpul unei funcții este un bloc
  - Un bloc este executat într-un cadru de execuție nou care:
    - conține informații administrative (utile în faza de depanare)
    - determină unde și cum va continua execuția programului (după terminarea execuției blocului curent)
    - definește 2 spații de nume (spațiul local și cel global) care afectează execuția blocului de instrucțiuni

Funcții în Python

- Apelul unei funcții
  - Spațiu de nume
    - un container de nume
    - o legătură între nume și obiecte
    - înlătură ambiguitățile numelor homonime
    - poate fi referit de mai multe cadre de execuție
    - are funcționalități similare unui dicționar
  - Legarea unui nume de un obiect (binding)
    - adăugarea unui nume într-un spațiu de nume
  - Re-legarea (rebinding)
    - schimbarea legăturii dintr-un nume și un obiect
  - Dezlegarea (unbinding)
    - eliminarea unui nume dintr-un spațiu de nume

Funcții în Python

Apelul unei funcții

- Parametrii unei funcții pot fi:
  - Formali
    - Identificatori ai parametrilor de intrare
    - fiecare apel al funcției trebuie să respecte numărul, ordinea și tipul parametrilor (obligatorii)
  - Actuali (argumente)
    - valori oferite parametrilor (formali ai) unei funcții atunci când este apelată
    - Sunt introduși în tabele de simboluri locale ale funcției apelate
    - Se transmit prin referință

```
def search(element, list):
    for x in list:
        if (x == element):
            return True
    return False
a = [2, 3, 4, 5, 6]
el = 3
if (search(el, a) == True):
    print("el was found...")
else:
    print("el was not found...")
```

Funcții în Python

Vizibilitatea variabilelor

- Scop – definește vizibilitatea unui nume (de variabilă) într-un bloc
  - Scopul unei variabile definite într-un bloc este blocul respectiv
  - Toate variabilele definite pe un anumit nivel de indentare sau scop sunt considerate locale acelui nivel sau scop
- Tipuri de variabile
  - Locale – un nume (de variabilă) definit într-un bloc
  - Globale – un nume (de variabilă) definit într-un modul
  - Libere – un nume folosit într-un bloc, dar ne-definit acolo (definit in altă parte)

Funcții în Python

Vizibilitatea variabilelor

```
g1 = 1 # g1 - global variable (also local, a module being a block)
def fun1(a): # a is a formal parameter
    b = a + g1 # b - local variable, g1 - free variable
    if b > 0: # a, b, and g1 are visible in all blocks of this function
        c = b - g1 # b is visible here, also g1
        b = c # c is a local variable defined in this block
    return b # c is not visible here
def fun2():
    global g1
    d = g1 # g1 - global variable
    g1 = 2
    return d + g1
print (fun1(1))
print (fun2())
```

Funcții în Python

- Vizibilitatea variabilelor
  - Reguli pentru stabilirea scopului unui nume (de variabilă sau de funcție)
    - un nume este vizibil doar în interiorul blocului în care a fost definit
    - parametrii formali ai unei funcții aparțin scopului corpului funcției (sunt vizibili doar în interiorul funcției)
    - numele definite în exteriorul unei funcții (la nivelul modulului) aparțin scopului modulului
    - când se folosește un nume într-un bloc, vizibilitatea lui este determinată de cel mai apropiat scop (care conține acel nume)

Funcții în Python

- Vizibilitatea variabilelor
  - Inspectarea variabilelor globale/locale ale unui program
    - globals()
    - locals()

```
a = 300
def f():
    a = 500
    print(a)

print(locals())

print(globals())
f()
print(a)
```

Functii in Python

- Transmiterea parametrilor

```
def change_or_not_immutable(a):
        print('Locals ', locals())
        print('Before assignment: a = ', a, ' id = ', id(a))
        a = 0
        print ('After assignment: a = ', a, ' id = ', id(a))
g1 = 1 #global immutable int
print ('Globals ', globals())
print ('Before call: g1 = ', g1, ' id = ', id(g1))
change_or_not_immutable(g1)
print ('After call: g1 = ', g1, ' id = ', id(g1))
```

```
def change_or_not_mutable(a):
    #print ('Locals ', locals())
    print ('Before assignment: a = ', a, ' id = ', id(a))
    a[1] = 3
    a = [0]
    print ('After assignment: a = ', a, ' id = ', id(a))
g2 = [0, 1] #global mutable list
#print ('Globals ', globals())
print ('Before call: g2 = ', g2, ' id = ', id(g2))
change_or_not_mutable(g2)
print ('After call: g2 = ', g2, ' id = ', id(g2))
```

Cum se scriu funcții?

Dezvoltarea dirijată de teste (TDD – test driven development)

- Implică crearea de teste înainte de a scrie efectiv codul funcției

- Pași pentru crearea unei noi funcții f()
  - Adăugarea unui/unor test/teste
  - Execuția testelor și verificarea dacă cel puțin unul dintre ele a eșuat
  - Scrierea corpului funcției
  - Rularea tuturor testelor
  - Refactorizarea codului

Cursul următor

- Dezvoltarea dirijată de teste (TDD – test driven development)
- Programarea modulară

Bibliografie

- Limbajul Python
  - http://docs.python.org/3/reference/index.html
- Biblioteca standard Python
  - http://docs.python.org/3/library/index.html
- Tutorial Python
  - http://docs.python.org/3/tutorial/index.html
- Kent Beck. Test Driven Development: By Example. *Addison-Wesley* Longman, 2002
  - http://en.wikipedia.org/wiki/Test-driven_development
- Martin Fowler. Refactoring. Improving the Design of Existing Code. *Addison-Wesley*, 1999
  - http://refactoring.com/catalog/index.html