FUNKCIONALNO PROGRAMIRANJE

Oznaka predmeta: FPR

Predavanje broj: 3

Nastavna jedinica: PYTHON,

Nastavne teme:

Liste. Metode liste. N-torka. Funkcije za n-torke. Rečnici. Funkcije za rečnike. Metode rečnika. Funkcije. Argumenti funkcije (Required arguments, Keyword arguments, Default arguments, Variable-length arguments). Lambda funkcije. Opseg promenljivih Moduli. Paketi.

Predavač: prof. dr Perica S. Štrbac, dipl. ing.

Literatura:

Steven Lott: "Functional Python Programming", Packt Publishing, 2015.

Python: liste

- Osnovna struktura podataka u Python-u je sekvenca. Lista je tip sekvence.
- Lista se predstavlja članovima koji se navode kao CSV unutar uglatih zagrada.

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"];
```

- Pristupanje vrednostima u listi je kao što sledi:

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7 ];
print ("list1[0]: ", list1[0])
print ("list2[1:5]: ", list2[1:5])
```

```
list1[0]: physics
list2[1:5]: [2, 3, 4, 5]
```

- Ažuriranje liste:

```
list = ['physics', 'chemistry', 1997, 2000];
print ("Value available at index 2 : ", list[2]);
list[2] = 2001;
print ("New value available at index 2 : ", list[2]);
```

```
Value available at index 2 : 1997
New value available at index 2 : 2001
```

Python: liste

- Brisanje elemenata liste:

```
list1 = ['physics', 'chemistry', 1997, 2000];
print (list1); del list1[2];
print ("After deleting value at index 2 : ")
print (list1);
```

```
['physics', 'chemistry', 1997, 2000]
After deleting value at index 2 :
['physics', 'chemistry', 2000]
```

- Osnovne operacije sa listom:

Python izraz	Rezultat	Opis
`len([1, 2, 3])`	`3`	Broj elemenata liste
`[1, 2, 3] + [4, 5, 6]`	`[1, 2, 3, 4, 5, 6]`	Konkatenacija
`['Hi!'] * 4`	`['Hi!', 'Hi!', 'Hi!', 'Hi!']`	Ponavljanje
`3 in [1, 2, 3]`	`True`	Pripadnost
`for x in [1, 2, 3]:` `print (x, end=' ')`	`1 2 3`	Iteracija

Python: liste

- Indeksiranje i odsecanje liste:

```
L = ['spam', 'Spam', 'SPAM!'];
```

Python izraz	Rezultat	Opis
`L[2]`	`'SPAM!'`	Ofset ide od nule
`L[-2]`	`'Spam'`	Negativan indeks se broji od kraja liste
`L[1:]`	`['Spam', 'SPAM!']`	Odsečak od indeksa 1 do kraja liste

Ugrađene funkcije za liste:

- koristi se (list1 > list2) - (list1 < list2) umesto `cmp(list1, list2)`, (videti p02)
  - poredi elemente dve liste
  - ako su elementi istog tipa vrši se poređenje i vraća se rezultat
  - ako su elementi različitih tipova
    - proverava se da li su brojevi.
      - ako jesu, svode se na isti tip, porede i vraća se rezultat.
      - ako je jedan element broj drugi element je veći (brojevi su "najmanji" tip)
      - ako nisu, elementi se sortiraju alfabetski.
  - ako se dosegne kraj jedne liste, a druge ne, duža lista je veća
  - ako se iscrpe obe liste i imaju iste podatke vraća se 0.

Python: liste

```
list1, list2 = [123, 'xyz'], [456, 'abc']
print ((list1 > list2) - (list1 < list2));    # -1
print ((list2 > list1) - (list2 < list1));    # 1
list3 = list2 + [786];
print ((list2 > list3) - (list2 < list3))          # -1
```

- len(list)

```
list1, list2 = [123, 'xyz', 'zara'], [456, 'abc']
print ("First list length : ", len(list1));
print ("Second list length : ", len(list2));
```

```
First list length :  3
Second lsit length :  2
```

- max(list)

```
list1, list2 = [123, 'xyz', 'zara', 'abc'], [456, 700, 200]
print ("Max value element : ", max(list1));
print ("Max value element : ", max(list2));
```

```
Max value element : zara
Max value element : 700
```

- min(list), za prethodno date liste

```
print ("Min value element : ", min(list1)); # Min value element : 123
print ("Min value element : ", min(list2)); # Min value element : 200
```

Python: liste, metode liste

- list(sequence)
  - konvertuje sekvencu u listu

```
aTuple = (123, 'xyz', 'zara', 'abc');
aList = list(aTuple)
print ("List elements : ", aList)
```

```
List elements : [123, 'xyz', 'zara', 'abc']
```

Metode liste:

- list.append(obj)

```
aList = [123, 'xyz', 'zara', 'abc'];
aList.append( 2009 );
print ("Updated List : ", aList);
```

```
Updated List : [123, 'xyz', 'zara', 'abc', 2009]
```

- list.count(obj)

```
aList = [123, 'xyz', 'zara', 'abc', 123];
print ("Count for 123 : ", aList.count(123));
print ("Count for zara : ", aList.count('zara'));
```

```
Count for 123 : 2
Count for zara : 1
```

- list.extend(sequence)

```
aList = [123, 'abc', 123];        bList = [2009, 'manni'];
aList.extend(bList);     print ("Extended List : ", aList) ;
```

```
Extended List : [123, 'abc', 123, 2009, 'manni']
```

Python: metode liste

- list.index(obj)

```
aList = [123, 'xyz', 'zara', 'abc'];
print ("Index for xyz : ", aList.index( 'xyz' )) ;
print ("Index for zara : ", aList.index( 'zara' )) ;
```

```
Index for xyz : 1
Index for zara : 2
```

- list.insert(index, obj)

```
aList = [123, 'xyz', 'zara', 'abc']
aList.insert( 3, 2009)
print ("Final List : ", aList)
```

```
Final List : [123, 'xyz', 'zara', 2009, 'abc']
```

- list.pop(obj=list[-1])

```
aList = [123, 'xyz', 'zara', 'abc'];
print ("A List : ", aList.pop());
print ("B List : ", aList.pop(2));
```

```
A List : abc
B List : zara
```

- list.remove(obj)

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz'];
aList.remove('xyz'); print ("List : ", aList);
aList.remove('abc'); print ("List : ", aList);
```

```
List : [123, 'zara', 'abc', 'xyz']
List : [123, 'zara', 'xyz']
```

Python: metode liste, tuple (n-torka)

- list.reverse()

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz'];
aList.reverse();
print ("List : ", aList);
```

```
List : ['xyz', 'abc', 'zara', 'xyz', 123]
```

- list.sort([key],[reverse])

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz'];
aList.sort();
print ("List : ", aList);
```

```
List : [123, 'abc', 'xyz', 'xyz', 'zara']
```

Tuple (n-torka):

- N-torka predstavlja sekvencu neizmenljivih objekata.
- Elementi n-torke su CSV vrednosti navedene u malim zagradama.

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 );
tup3 = "a", "b", "c", "d";
tup4 = ();          # prazna n-torka
tup5 = (123,);      # n-torka sa jednim elementom
```

- Elementi n-torke su indeksirani od 0.

Python: tuple (n-torka)

- Pristupanje vrednostima n-torke.
- Ne mogu se menjati vrednosti članova n-torke.

```
tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');
# ne moze tup1[0] = 100;
tup3 = tup1 + tup2; print (tup3); # konkatenacija moze
                                  (12, 34.56, 'abc', 'xyz')
```

- Nije moguće uklanjati pojedine elemente n-torke:

```
tup = ('physics', 'chemistry', 1997, 2000);
print (tup);
del tup;
print ("After deleting tup : ")
print (tup);
        ('physics', 'chemistry', 1997, 2000)
        After deleting tup :
        Traceback (most recent call last):
                File "test.py", line 9, in <module> print tup;
        NameError: name 'tup' is not defined
```

- Osnovne operacije sa n-torkama obuhvataju: konkatenaciju, vraćanje dužine n-torke, ponavljanje n-torke, provera pripadnosti elementa, iteracija po elementima n-torke.

Python: tuple (n-torka)

Python izraz	Rezultat	Opis
`len((1, 2, 3))`	`3`	Broj elemenata n-torke
`(1, 2, 3) + (4, 5, 6)`	`(1, 2, 3, 4, 5, 6)`	Konkatenacija
`('Hi!') * 4`	`('Hi!', 'Hi!', 'Hi!', 'Hi!')`	Ponavljanje
`3 in (1, 2, 3)`	`True`	Pripadnost
`for x in (1, 2, 3):` `print (x,end=' ')`	`1 2 3`	Iteracija

- Indeksiranje i odsecanje n-torke:

```
L = ('spam', 'Spam', 'SPAM!');
```

Python izraz	Rezultat	Opis
`L[2]`	`'SPAM!'`	Ofset ide od nule
`L[-2]`	`'Spam'`	Negativan indeks se broji od kraja n-torke
`L[1:]`	`('Spam', 'SPAM!')`	Odsečak od indeksa 1 do kraja n-torke

Python: funkcije za n-torke

Funkcije za n-torke su kao što sledi:

- koristi se (tuple1>tuple2)-(tuple1<tuple2) umesto cmp(tuple1, tuple2)
  - poredi elemente dve n-torke
  - ako su elementi istog tipa vrši se poređenje i vraća se rezultat
  - ako su elementi različitih tipova
    - proverava se da li su brojevi.
      - ako jesu, svode se na isti tip, porede i vraća se rezultat.
      - ako je jedan element broj drugi element je veći (brojevi su "najmanji" tip)
      - ako nisu, elementi se porede alfabetski.
  - ako se dosegne kraj jedne n-torke, a druge ne, duža n-torka je veća
  - ako se iscrpe obe n-torke i imaju iste podatke vraća se 0.

```
tuple1, tuple2 = (123, 'xyz'), (456, 'abc')
print ((tuple1>tuple2)-(tuple1<tuple2) ); # -1
print ((tuple2>tuple1)-(tuple2<tuple1) ); # 1
tuple3 = tuple2 + (786,);
print ((tuple2>tuple3)-(tuple2<tuple3) )  # -1
```

- len(tuple)

```
tuple1, tuple2 = (123, 'xyz', 'zara'), (456, 'abc')
print (len(tuple1)); # 3
print (len(tuple2)); # 2
```

Python: : funkcije za n-torke

- max(tuple)

```
tuple1, tuple2 = (123, 'xyz', 'zara', 'abc'), (456, 700, 200)
print ("Max value element : ", max(tuple1));
print ("Max value element : ", max(tuple2));
```

```
Max value element : zara
Max value element : 700
```

- min(tuple)

```
tuple1, tuple2 = (123, 'xyz', 'zara', 'abc'), (456, 700, 200)
print ("min value element : ", min(tuple1));
print ("min value element : ", min(tuple2));
```

```
min value element : 123
min value element : 200
```

- tuple(sequence)
  - pretvara sekvencu u n-torku

```
aList = [123, 'xyz', 'zara', 'abc'];
aTuple = tuple(aList)
print ("Tuple elements : ", aTuple)
```

```
Tuple elements : (123, 'xyz', 'zara', 'abc')
```

Python: dictionary (rečnik)

- Rečnik je sekvenca koja se zapisuje u velike zagrade u kojima se navode parovi ključ-vrednost.
- Par ključ vrednost je razdvojen sa : dok su parovi odvojeni CSV formatom.

```
dict = {'Name': 'Jane', 'Age': 30, 'Class': 'First'};
print ("dict['Name']: ", dict['Name']);
print ("dict['Age']: ", dict['Age']);
print ("dict['Alice']: ", dict['Alice']);
```

```
dict['Name']: Jane
dict['Age']: 30
dict['Alice']:
Traceback (most recent call last):
  File "test.py", line 4, in <module>
        print "dict['Alice']: ", dict['Alice'];
KeyError: 'Alice'
```

- Ažuriranje elemenata rečnika:

```
dict = {'Na': 'Joe', 'Age': 37};
del dict['Na']; # uklanja ulaz 'Na'
dict.clear();   # uklanja sve ulaze
del dict ;      # brise recnik
print ("dict['Age']: ", dict['Age']);
print ("dict['School']: ", dict['School']);
```

```
dict['Age']:
Traceback (most recent call last):
  File "main.py", line 9, in
    print "dict['Age']: ",
    dict['Age'];
TypeError: 'type' object has no
attribute '__getitem__'
```

Python: dictionary (rečnik), funkcije

- Vrednosti u rečniku nemaju ograničenja, ali ključevi imaju:
  - ključ mora biti jedinstven inače se računa samo poslednji

```
dict = {'Name': 'Jane', 'Age': 30, 'Name': 'Manni'};
print ("dict['Name']: ", dict['Name']);
```

```
dict['Name']: Manni
```

  - ključ mora biti neizmenljiv

```
dict = {['Name']: 'Jane', 'Age': 30};
print ("dict['Name']: ", dict['Name']);
```

```
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    dict = {['Name']: 'Zara', 'Age': 7};
TypeError: list objects are unhashable
```

Ugrađene funkcije koje barataju rečnikom:

- poređenje koje je zadržano je dict==dict2

```
dict1 = {'Name': 'Zara', 'Age': 7};
dict2 = {'Name': 'Mahnaz', 'Age': 27};
dict3 = {'Name': 'Zara', 'Age': 7};
print (dict1==dict2)     # False
print (dict1==dict3))    # True
```

Python: dictionary (rečnik), funkcije i metode

- len(dict)

```
dict = {'Name': 'Zara', 'Age': 7};
print ("Length : %d" % len (dict))       # Length : 2 para n-v
```

- str(dict)

```
dict = {'Name': 'Zara', 'Age': 7};
print ("Equivalent String : %s" % str (dict))
                Equivalent String : {'Age': 7, 'Name': 'Zara'}
```

- type(variable)

```
dict = {'Name': 'Zara', 'Age': 7};
print ("Variable Type : %s" % type (dict))
                Variable Type : <class 'dict'>
```

Metode rečnika:

- dict.clear()

```
dict = {'Name': 'Zara', 'Age': 7};
print ("Start Len : %d" % len(dict)) # Start Len : 2
dict.clear()
print ("End Len : %d" % len(dict))   # End Len : 0
```

- dict.copy()

```
dict1 = {'Name': 'Zara', 'Age': 7}; dict2 = dict1.copy()
print (dict2);  # {'Age': 7, 'Name': 'Zara'}
```

Python: dictionary (rečnik), metode

- dict.fromkeys()

```
seq = ('name', 'age', 'sex')
dict = dict.fromkeys(seq)
print ("New Dictionary : %s" % str(dict))
dict = dict.fromkeys(seq, 10)
print ("New Dictionary : %s" % str(dict))
```

```
New Dictionary : {'age': None, 'name': None, 'sex': None}
New Dictionary : {'age': 10, 'name': 10, 'sex': 10}
```

- dict.get(key, default=None)

```
dict = {'Name': 'Zara', 'Age': 27}
print ("Value : %s" % dict.get('Age'))            # Value : 27
print ("Value : %s" % dict.get('Sex', "Never"))   # Value : Never
```

- dict.has_key(key)

```
dict = {'Name': 'Zara', 'Age': 27}
print ("Value : %s" % dict.has_key('Age'))        # Value : True
print ("Value : %s" % dict.has_key('Sex'))        # Value : False
```

- dict.items()

```
dict = {'Name': 'Zara', 'Age': 27}
print ("Value : %s" % dict.items())
```

```
Value : [('Age', 7), ('Name', 'Zara')]
```

Python: dictionary (rečnik), metode

- dict.keys()

```
dict = {'Name': 'Zara', 'Age': 27}
print ("Value : %s" % dict.keys())
```

```
Value : ['Name', 'Age']
```

- dict.setdefault(key, default=None) #postavi ako nema

```
dict = {'Name': 'Zara', 'Age': 27}
print ("Value : %s" % dict.setdefault('Age', None))
print ("Value : %s" % dict.setdefault('Sex', 'f'))
```

```
Value : 27
Value : 'f'
```

- dict.update(dict2)

```
dict = {'Name': 'Zara', 'Age': 27}
dict2 = {'Sex': 'female' }
dict.update(dict2)
print ("Value : %s" % dict)
```

```
Value : {'Name': 'Zara', 'Age': 27, 'Sex': 'female'}
```

- dict.values()

```
dict = {'Name': 'Zara', 'Age': 27}
print ("Value : %s" % dict.values())
```

```
Value : ['Zara', 27]
```

Python: funkcije

- Slede pravila za definisanje funkcije u Python-u:
  - Blok funkcije počinje ključnom rečju def iza koje sledi ime funkcije i male zagrade (( )).
  - Svi ulazni parametri ili argumenti trebalo bi da budu navedeni unutar pomenutih zagrada. Moguće je i definisati parametre unutar ovih zagrada.
  - Prva naredba funkcije može biti opcionalna naredba koja predstavlja dokumentacioni string funkcije (docstring).
  - Blok koda unutar svake funkcije počinje iza dvotačke (:). Ovaj blok je uvučen.
  - Iz funcije se izlazi naredbom return [expression] gde se opciono vraća izraz iz funkcije onome koji ju je pozvao.

```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```

primer:

```
def printme( str ):
   "This prints a passed string into this function"
   print (str)
   return
```

Python: funkcije

- Pozivanje funkcije je ili iz Python prompta ili iz drugih funkcija.

```
# definicija funkcije
def printme( str ):
   "This prints a passed string into this function"
   print (str);
   return;
# pozivi funkcije
printme("I'm first call to user defined function!");
printme("Again second call to the same function");
```

```
I'm first call to user defined function!
Again second call to the same function
```

- U Python-u se svi argumenti prenose po "referenci" (paziti šta znači dodela):

```
# definicija funkcije
def changeme( mylist ):
   "This changes a passed list into this function"
   mylist.append([1,2,3,4]);
   print ("Values inside the function: ", mylist)
   return
# poziv funkcije
mylist = [10,20,30];
changeme( mylist );
print ("Values outside the function: ", mylist)
```

```
Values inside the function: [10, 20, 30, [1, 2, 3, 4]]
Values outside the function: [10, 20, 30, [1, 2, 3, 4]]
```

Python: funkcije

- U sledećem primeru argument je prosleđen preko reference a onda je referenca prepisana unutar pozvane funkcije:

```
# definicija funkcije
def changeme( mylist ):
   "This changes a passed list into this function"
   mylist = [1,2,3,4]; # dodela za novu referencu
   print ("Values inside the function: ", mylist)
   return
# poziv funkcije
mylist = [10,20,30];
changeme( mylist );
print ("Values outside the function: ", mylist)
```

```
Values inside the function: [1, 2, 3, 4]
Values outside the function: [10, 20, 30]
```

- Argumenti funkcije:
  Funkcija se može pozvati sa sledećim tipovima formalnih argumenata:
  - Required arguments
  - Keyword arguments
  - Default arguments
  - Variable-length arguments

Python: funkcije, *required arguments*

- Zahtevani argumenti (*Required arguments*) su argumenti prosleđeni funkciji u pravilnom redosledu pri čemu broj argumenata koji se prosleđuju funkciji odgovara definiciji funkcije:

```
# definicija funkcije
def printme( str ):
   "This prints a passed string into this function"
   print (str);
   return;
# poziv funkcije
printme("OK");
printme();
```

```
OK
Traceback (most recent call last):
  File "main.py", line 11, in
    printme();
TypeError: printme() takes exactly 1 argument (0 given)
```

- Ključne reči - argumenti, odnose se na identifikatore formalnih argumenta u funkciji i korišćenja tih identifikatora za kopiranje stvarnih argumenta.
  - Ovo je omogućeno u Python-u tako da se može promeniti redosled kojim su navedeni identifikatori formalnih argumenta sa dodelama stvarnih argumenata.

Python: funkcije, *keyword arguments*

```
# Primer 1
# definicija funkcije
def printme( str ):
   "This prints a passed string into this function"
   print (str);
   return;
# poziv funkcije
printme( str = "My string");
```

```
My string
```

```
# Primer 2
# definicija funkcije
def printinfo( name, age ):
   "This prints a passed info into this function"
   print ("Name: ", name);
   print ("Age ", age);
   return;
# poziv funkcije
printinfo( age=50, name="miki" );
printinfo( age=100, name="" );
```

```
Name: miki
Age 50
Name:
Age 100
```

Python: funkcije, *Default arguments, Variable-length arguments*

- Podrazumevani argumenti (*Default arguments*) se navode u definiciji funkcije tako da je moguće da se ne navedu odgovarajući stvarni argumenti u pozivu funkcije.

```
def printinfo( name, age = 35 ):
   "This prints a passed info into this function"
   print ("Name: ", name);
   print ("Age ", age);
   return;
printinfo( age=50, name="miki" );
printinfo( name="miki" );
```

```
Name: miki
Age 50
Name: miki
Age 35
```

- Varijabilni broj argumenata (*Variable-length arguments*) omogućuje da funkciji prosledimo proizvoljan broj argumenta:

```
def functionname([formal_args,] *var_args_tuple ):
   "function_docstring"
   function_suite
   return [expression]
```

Zvezdica (*) ispred identifikatora varijable omogućuje prihvatanje svih vrednosti dodatnih argumenta (ova n-torka ostaje prazna ako nema dodatnih argumenata).

Python: funkcije, *Variable-length arguments, lambda*

```
def printinfo( arg1, *vartuple ):
   "This prints a variable passed arguments"
   print ("Output is: ")
   print (arg1)
   for v in vartuple:
      print (v,end=' ')
   return;
printinfo( 10 );
printinfo( 70, 60, 50 );
```

```
Output is:
10
Output is:
70
60 50
```

Anonimne (lambda) funkcije

- Naziv su dobile jer se ne deklarišu standardno sa def, već se koristi ključna reč lambda.
- Lambda prima proizvoljno argumenata a vraća jedan izraz. Ne sadrži komande ili višestruke izraze.
- Lambda funkcije koriste sopstveni prostor imena.

Python: lambda

- Sintaksa lambda funkcije:

```
lambda [arg1 [,arg2,.....argn]]:expression
```

```
# primer
sum = lambda arg1, arg2: arg1 + arg2;
print ("Value of total : ", sum( 10, 20 ))
print ("Value of total : ", sum( 20, 20 ))
```

```
Value of total : 30
Value of total : 40
```

- Naredba return:

```
def sum( arg1, arg2 ):
   total = arg1 + arg2
   print ("Inside the function : ", total)
   return total;
total = sum( 10, 20 );
print ("Outside the function : ", total)
```

```
Inside the function : 30
Outside the function : 30
```

- Opseg važenja promenljivih:
  - globalne promenljive (definisane su van funkcije)
  - lokalne promenljive (definisane su unutar funkcije)

Python, opseg promenljivih, moduli

- Primer globalne i lokalne varijable:

```
total = 0;                          # globalna.
def sum( arg1, arg2 ):
   total = arg1 + arg2;             # lokalna.
   print ("Inside the function local total : ", total)
   return total;
sum( 10, 20 );
print ("Outside the function global total : ", total)
```

```
Inside the function local total : 30
Outside the function global total : 0
```

Moduli u Python-u:

- Koriste se za logičko organizovanje koda. Modul predstavlja fajl sa Python kodom.
- Modul može definisati: funkcije, klase i promenljive.
- Neka je funkcija `print_func` smeštena u fajl `support.py` i neka je sadržaj kao što sledi:

```
def print_func( par ):
   print ("Hello : ", par)
   return
```

- Bilo koji Python-ov fajl može biti tretiran kao modul korišćenjem naredbe import:

```
import module1[, module2[,... moduleN]
```

Python, opseg promenljivih, moduli

- Primer importa modula i korišćenja:

```
import support
support.print_func("Zara")
```

```
Hello : Zara
```

- Modul se učitava samo jednom.
- Naredba `from` omogućuje uvoz specifičnih atributa iz modula u tekući prostor imena (namespace).

```
from modulname import name1[, name2[, ... nameN]]
```

primer: uvoz funkcije fibonnaci iz modula fib

```
from fib import fibonacci
```

- Naredba `from` ne uključuje modul fib u tekući prostor imena već samo član fibonnaci iz modula fib u tabelu globalnih simbola.
- Moguće je importovati sva imena iz modula u tekući prostor imena korišćenjem naredbe `from` kao što sledi:

```
from modname import *
```

Python, opseg promenljivih, moduli

- Prilikom importa modula, Python interpreter traži dati modul kao što sledi:
  - prvo se proverava da li se modul nalazi u tekućem folderu
  - ako se modul ne pronađe, Python onda traži modul u svakom folderu navedenom u sistemskoj promenljivoj PYTHONPATH.
  - ako i dalje nije našao modul onda Python proverava da li je modul u podrazumevanom folderu (npr. na linux-u /usr/local/lib/python/).
- Staza za traženje modula je navedena u sistemskom modulu sys kao sys.path promenljiva.
  - Promenljiva sys.path sadrži: tekući folder, PYTHONPATH i instalaciono zavisne staze.
- Promenljiva PYTHONPATH (varijable okruženja):
  - sastoji se od liste foldera.
  - sintaksa PYTHONPATH-a je ista kao i promenljive PATH.
    - primer tipične PYTHONPATH varijable u Windows-u:

```
set PYTHONPATH=c:\python30\lib;
```

    - primer tipične PYTHONPATH u linux-u:

```
set PYTHONPATH=/usr/local/lib/python
```

Python, opseg promenljivih

- Naredba Python-a može pristupiti promenljivoj u lokalnom prostoru imena i u globalnom prostoru imena.
- Ako lokalna varijabla ima isto ime kao globalna varijabla onda lokalna varijabla skriva globalnu varijablu.
- Svaka funkcija ima svoj lokalni prostor imena.
- Metode klase takođe imaju pravila vezana za opseg kao i obične funkcije.
- Kod Python-a bilo koja varijabla koja je dodeljena u okviru funkcije predstavlja lokalnu varijablu.
- Međutim, da bi se dodelila vrednost globalnoj varijabli unutar funkcije mora se upotrebiti naredba global.
  - npr. global VarName označava da je VarName globalna varijabla u Python-u.

```
Money = 2000
def AddMoney():
   # global Money #ako se ova linija ukljuci sve ce raditi
   Money = Money + 1
print (Money)
AddMoney()
print (Money)
```

ako bi ostao komentar # global Money, Python bi dojavio grešku da lokalna promenljiva nije definisana.

Python, opseg promenljivih, moduli

- Funkcija locals() pozvana unutar funkcije vratiće sva imena kojima se pristupa lokalno iz te funkcije.
- Funkcija globals() pozvana unutar funkcije vratiće sva imena kojima se može pristupiti globalno iz te funkcije.
- Povratni tip obe navedene funkcije je rečnik (imena mogu biti ekstrahovana korišćenjem funkcije keys()).
- Ugrađena funkcija dir(modul) vraća sortiranu listu stringova koji predstavljaju imena definisana u modulu.
- Lista sadrži imena svih modula, varijabli i funkcija koji su definisani u datom modulu.

```
import math
content = dir(math)
print (content);
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod',
'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf',
'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi',
'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

Navedene su i specijalne string varijable __name__ (odnosi se na naziv modula)...

Python, moduli, paketi

- Funkcija modula importlib.reload(ime_modula) ponovo importuje dati modul.
- Paketi u Python-u predstavljaju hijerarhijsku strukturu fajlova u folderu koji definišu okruženje Python aplikacije koje sadrži module i potpakete.
- Neka je fajl `Pots.py` dostupan u `Phone` folderu sa sadržajem kao što sledi:

```
def Pots():
    print ("I'm Pots Phone")
```

i neka postoje dva fajla koji kao i prethodni imaju i svoje istoimene funkcije:

`Phone/Isdn.py`	`sadrži funkciju Isdn()`
`Phone/G3.py`	`sadrži funkciju G3()`

Sada je potrebno kreirati još jedan fajl u `Phone` direktorijumu:

`Phone/__init__.py`

- Da bi se omogućilo da sve navedene funkcije budu dostupne pri importu paketa `Phone`, potrebno je eksplicitno staviti `from-import` naredbe u fajl `__init__.py` kao što sledi:

```
from Pots import Pots
from Isdn import Isdn
from G3 import G3
```

Python, opseg promenljivih, moduli

- Nakon dodavanja prethodnih linija u fajl `__init__.py`, dobiće se dostupnost svih navedenih funkcija pri importu paketa Phone.

```
import Phone
Phone.Pots()
Phone.Isdn()
Phone.G3()
```

```
I'm Pots Phone
I'm 3G Phone
I'm ISDN Phone
```

- U prethodnom primeru dato je uključenje po jedne funkcije iz svakog fajla, a na isti način se može uključiti više funkcija.
- Mogu se definisati i različite Python klase u navedenim fajlovima a onda kreirati posebni paketi koji ih uključuju.