FUNKCIONALNO PROGRAMIRANJE

Oznaka predmeta: FPR

Predavanje broj: 05

Nastavna jedinica: PYTHON,

Nastavne teme:

Assertions. Izuzeci (try-except-else, try-finally, prikaz informacija). Korisničko generisanje izuzetka (raise). Definisanje korisničkih izuzetaka. Klase (definicija, članovi, konstruktor, instance, destruktor, metode, baratanje atributima, ugrađeni atributi, nasleđivanje, garbage collector, nadjačavanje, preklapanje operatora).

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

Literatura:

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

Python: assertions

- Assertion bi trebalo posmatrati kao **raise-if-not** naredbu gde se nakon testiranja izraza u zavisnosti od rezultata generiše izuzetak.
  - Assertion se stavlja obično na početak funkcije da se proveri da li je ulaz validan i nakon funkcije da se proveri validnost izlaza.
- Naredba assert će generisati *AssertionError* izuzetak ako je dati izraz False pri čemu se koriste dati argumenti:

```
assert Expression[, Arguments]
```

Ako je u programu postavljeno da se ovaj izuzetak obrađuje isti će biti obrađen inače program završava rad i generiše *traceback*.

```
#!/usr/bin/python
def KelvinToFahrenheit(Temperature):
   assert (Temperature >= 0),"Colder than absolute zero!"
   return ((Temperature-273)*1.8)+32
print(KelvinToFahrenheit(273))
print(int(KelvinToFahrenheit(505.78)))
print(KelvinToFahrenheit(-5))
```

```
32.0
451
Traceback (most recent call last):
File "test.py", line 9, in
print KelvinToFahrenheit(-5)
File "test.py", line 4, in
KelvinToFahrenheit
assert (Temperature >= 0),"Colder than
         absolute zero!" AssertionError:
         Colder than absolute zero!
```

Python: izuzetak (exception)

- Izuzetak (exception) je događaj koji de desi tokom izvršavanja programa i koji prekida normalan tok programa.
- Kada Python skript naiđe na situaciju koju ne može da reši onda generiše izuzetak.
- *Izuzetak* je Python objekat koji reprezentuje grešku.
  - Kada Python skript generiše izuzetak on mora ili da odmah obradi izuzetak ili da terminira program.
- Kod upravljanja izuzetkom potrebno je da se kod koji može generisati izuzetak postavi u **try:** blok. Posle try: bloka slede blokovi **except izuzetak.** Iza svakog except izuzetak sledi blok naredbi koji rukuje navedenim izuzetkom. Na kraju sledi **else** blok koji se izvršava ako se nije desio izuzetak.

  Primer sintakse *try....except...else* bloka.

```
try:
   You do your operations here;
   ......................
except ExceptionI:
   If there is ExceptionI, then execute this block.
except ExceptionII:
   If there is ExceptionII, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

Python: izuzetak, try-except-else

- Malo pojašnjenje prethodnog:
  - Jedna naredba try može imati više except naredbi.
    Ovo je korisno kada try blok sadrži naredbe koje mogu baciti različite tipove izuzetaka.
  - Moguće je navesti i klauzulu za generički izuzetak čime se postiže obrada bilo kog izuzetka.
  - Posle except klauzule(a) navodi se else klauzula. Kod u else bloku se izvršava ako se nije dogodio izuzetak u pripadnom try bloku.
  - Blok else je podesno mesto za kod koji je siguran u smislu da neće generisati izuzetak.

Primer: otvaranje nepostojećeg fajla:

```
try:
   fh  = open("nonamefile", "r")
   str = fh.readlines()
except IOError:
   print ("Error: can\'t find file or read data")
else:
   print (str)
   fh.close()
```

Python: izuzetak, try-except-else

- Primer koji sledi pokušava da otvori fajl za koga se nema dozvola upisa, tako da će se generisati izuzetak:

```
try:
   fh = open("testfile", "r")
   fh.write("This is my test file for exception handling!!")
except IOError:
   print ("Error: can\'t find file or read or write data")
else:
   print ("Written content in the file successfully")
   fh.close()
```

```
Error: can't find file or read or write data
```

- Moguće je koristiti except naredbu bez definisanja izuzetka tako da će biti uhvaćeni svi izuzeci (bolje je napraviti sitniju granulaciju):

```
try:
   You do your operations here;
   ......................
except:
   If there is any exception, then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

Python: izuzetak, try-finally

- Moguće je da jedan except rukuje sa više tipova izuzetaka:

```
try:
   You do your operations here;
   ......................
except(Exception1[, Exception2[,...ExceptionN]]]):
   If there is any exception from the given exception list,
   then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

- Iza try: bloka može se koristiti finally: blok. U finally blok se smešta kod koji se mora izvršiti bez obzira da li se desio izuzetak u try bloku:

```
try:
   You do your operations here;
   ......................
   Due to any exception, this may be skipped.
finally:
   This would always be executed. ......................
```

- Moguće je koristiti ekskluzivno ili except klauzule ili finally klauzulu.
- Ne može se koristiti else klauzula uz finally klauzulu.

Python: izuzetak, dvonivovska obrada

```
try:
   fh = open("testfile", "r")
   fh.write("This is my test file for exception handling!!")
finally:
   print ("In any case")
fh.close()
# ako nema dozvole za upis u dati fajl izlaz će biti:
                  In any case
try:
   fh = open("testfile", "r")
   try:
      fh.write("This is my test file for exception handling!!")
   finally:
      print ("Going to close the file")
      fh.close()
except IOError:
   print ("Error: can\'t find file or read data")
```

```
Going to close the file
Error: can't find file or read data
```

- Kada se baci izuzetak u *try* bloku izvršavanje programa se prebacuje na pripadni *finally* block.
- Nakon izvršenja svih naredbi u *finally* bloku izuzetak se postavlja ponovo i sada njime rukuje *except* blok ako postoji u sledećem višem nivou.

Python: izuzetak sa parametrima

- Izuzetak može imati argument koji predstavlja dodatnu informaciju o nastalom problemu. Sadržaj argumenta varira prema izuzetku.
  - Prihvatanje argumenta izuzetka je kao što sledi:

```
try:
   You do your operations here;
   ......................
except ExceptionType as Argument:
   You can print value of Argument here...
```

- Varijabla može imati jednostruku vrednost (poruka o uzroku problema) ili n-torku.

```
def temp_convert(var):
   try:
      return int(var)
   except ValueError as Argument:
      print ("The argument does not contain numbers\n", Argument)
temp_convert("xyz");
                  The argument does not contain numbers
                  invalid literal for int() with base 10: 'xyz'
```

Python: korisničko generisanje izuzetka

- Korisničko generisanje izuzetka može se izvesti naredbom **raise**:

```
raise [Exception [, args [, traceback]]]
```

  Exception je tip izuzetka (npr. NameError).
  - args je vrednost argumenta (koji je opcionalan, inače je None) izuzetka.
  - traceback je opcionalan (retko se koristi) a ako postoji predstavlja traceback objekat koji se koristi za izuzetak.

- Obično je izuzetak klasa sa argumentom koji je instanca te klase.
- Generisanje izuzetka:

```
def functionName( level ):
   if level < 1:
      raise NameError("Invalid level!")
      # The code below to this would not be executed
      # if we raise the exception
try:
   print("Business Logic here..."); functionName(0);
except NameError as ne:
   print("Exception handling here...", ne)
else:
   print("without exception...")
```

Python: korisničko definisanje izuzetka

- Korisnički definisan izuzetak u Python-u se relizuje nasleđivanjem klasa standardnih ugrađenih izuzetaka.
- Neka je korisnička klasa izvedena iz klase *RuntimeError*.
  - Ovo je korisno ako se želi prikazati više specifičnih informacija kada je izuzetak uhvaćen.
- U try bloku korisnički definisan izuzetak je generisan i uhvaćen u pripadnom except bloku.
- Varijabla e biće korišćena za kreiranje instance klase Networkerror.

```
class Networkerror(RuntimeError):
    def __init__(self, arg, arg2):
        self.args = arg # args je ugradjena n-torka parametara
        self.arg2 = arg2
```

Nakon ovoga moguće je generisati i uhvatiti korisnički izuzetak:

```
try:
    raise Networkerror(["prvi","drugi","treci"], "drugi argument")
except Networkerror as e:
    print (e.args, e.arg2)
```

```
('prvi', 'drugi', 'treci') drugi argument
```

U nastavku radimo detaljno, a ovde samo da pomenemo da metoda __init__() ima ulogu konstruktora.

Python: traceback

- Ideja je da se minijaturno "simulira" Python interaktivni prompt.
- Unesite nekoliko regularnih izraza a onda neki neregularan.

```
import sys, traceback

def run_user_code(glodic):
    source = input(">>> ")
    try:
        exec(source, glodic)
    except Exception:
        print("Exception in user code:")
        print("-"*30)
        #traceback.print_stack()
        traceback.print_exc(file=sys.stdout)
        print("-"*30)

glodic = {}
while True:
    run_user_code(glodic)
```

```
>>> a=10
>>> b=20
>>> c=a+b
>>> print(c)
30
>>> g
Exception in user code:
------------------------------
Traceback (most recent call
last):
  File
"C:/__Projects/Py/ProbniPY/p06.p
y", line 317, in run_user_code
    exec(source, envdir)
  File "<string>", line 1, in
<module>
NameError: name 'g' is not
defined
------------------------------
>>> g=[1,2,3]
>>> print(g)
[1, 2, 3]
>>>
```

Python: klase, definicija

- Definiciju nove klase u Python-u realizuje naredba class.
- Naziv klase sledi neposredno iza ključne reči class a onda sledi dvotačka:

```
class ClassName:
   'Optional class documentation string'
   class_suite
```

- Klasa ima dokumentacioni string koji je dostupan preko podatka člana `ClassName.__doc__`
- Blok class_suite sastoji se od svih naredbi koje definišu članove klase (podatke članove i funkcije članice, odnosno, atribute i metode).

```
class Employee:
   'Common base class for all employees'
   empCount = 0
   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1
   def displayCount(self):
      print ("Total Employee %d" % Employee.empCount )
   def displayEmployee(self):
      print ("Name : ", self.name, ", Salary: ", self.salary)
```

Python: klase, instance

- Varijabla `empCount` je podatak član čiju vrednost dele sve instance klase.
  - Ovome podatku članu može se pristupiti pomoću *Employee.empCount* unutar ili van klase.
- Prvi metod `__init__()` je specijalni metod koji predstavlja konstruktor klase koji se poziva kada se kreira nova instanca ove klase.
- Ostali metodi deklarišu se kao normalne funkcije uz razliku da je prvi argument metoda `self`.
  - Python dodaje `self` argument implicitno prilikom poziva metoda tako da ovaj argument korisnik ne navodi.
- Kreiranje instanci

```
"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
```

- Za pristup atributima objekta koristi se operator `.` (tačka), za atribut na nivou klase koristi se ime klase:

```
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" % Employee.empCount)
```

Python: primer

- Kompletan kod:

```
class Employee:
   'Common base class for all employees'
   empCount = 0
   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1
   def displayCount(self):
      print ("Total Employee %d" % Employee.empCount )
   def displayEmployee(self):
      print ("Name : ", self.name, ", Salary: ", self.salary )

"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" % Employee.empCount)
```

```
Name : Zara ,Salary: 2000
Name : Manni ,Salary: 5000
Total Employee 2
```

Python: manipulacija atributima

- Mogu se dodavati, uklanjati i modifikovati atributi klase objekta:

```
emp1.age = 7
# Add an 'age' attribute.
emp1.age = 8
# Modify 'age' attribute.
del emp1.age
# Delete 'age' attribute.
```

- Za pristup atributima mogu se koristiti sledeće funkcije:
  - `getattr(obj, name[, default])` : uzimanje vrednosti atributa objekta.
  - `hasattr(obj,name)` : da li postoji atribut objekta.
  - `setattr(obj,name,value)` : postavljanje atributa objekta (ako ne postoji kreira se).
  - `delattr(obj, name)` : brisanje atributa objekta.

```
hasattr(emp1, 'age')
# Returns true if 'age' attribute exists
getattr(emp1, 'age')
# Returns value of 'age' attribute
setattr(emp1, 'age', 8)
# Set attribute 'age' at 8
delattr(empl, 'age')
# Delete attribute 'age'
```

Python: klase, ugrađeni atributi

- Ugrađeni atributi klase su kao što sledi:

__dict__	rečnik koji sadrži prostor imena (namespace) klase.
__doc__	dokumentacioni string klase ili ništa ako isti nije definisan.
__name__	naziv klase.
__module__	ime modula u kom je klasa definisana. U interaktivnom modu ovo je atribut "__main__".
__bases__	n-torka koja sadrži osnovne klase u redosledu njihovog pojavljivanja u listi osnovnih klasa.

```
class Employee:
   'Common base class for all employees'
   empCount = 0
   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1
   def displayCount(self):
      print ("Total Employee %d" % Employee.empCount )
   def displayEmployee(self):
      print ("Name : ", self.name, ", Salary: ", self.salary )
print ("Employee.__doc__:", Employee.__doc__)
```

Python: klase, garbage collector

```
print ("Employee.__name__:", Employee.__name__)
print ("Employee.__module__:", Employee.__module__)
print ("Employee.__bases__:", Employee.__bases__)
print ("Employee.__dict__:", Employee.__dict__)
        Employee.__doc__: Common base class for all employees
        Employee.__name__: Employee
        Employee.__module__: __main__
        Employee.__bases__: (<class 'object'>,)
        Employee.__dict__: {'__init__': <function Employee.__init__ at
0x0000000000ABF730>, 'displayEmployee': <function Employee.displayEmployee at
0x0000000000ABF840>, '__doc__': 'Common base class for all employees', 'empCount':
0, '__dict__': <attribute '__dict__' of 'Employee' objects>, '__module__':
'__main__', 'displayCount': <function Employee.displayCount at 0x0000000000ABF7B8>,
'__weakref__': <attribute '__weakref__' of 'Employee' objects>}
```

- Uništavanje objekata (Garbage Collection) Python obavlja automatski čime oslobađa memorijski prostor koji se više ne koristi.
- Python-ov garbage collector radi tokom izvršavanja programa a pokreće se kada broj referenci na neki objekat padne na nulu.
- Brojač referenci na objekat se uvećava kada se izvrši dodela novom imenu ili kada se objekt stavlja u kontejner (lista, n-torka, rečnik).
- Brojač referenci na objekat se umanjuje kada se koristi naredba del, nova dodela reference ili referenca izlazi iz opsega.

Python: klase, destruktor

```
a = 40     # Create object <40>
b = a      # Increase ref. count of <40>
c = [b]    # Increase ref. count of <40>
del a      # Decrease ref. count of <40>
b = 100    # Decrease ref. count of <40>
c[0] = -1 # Decrease ref. count of <40>
```

- Klasa ima specijalni metod `__del__()` koji ima ulogu destruktora i poziva se kada se instanca uništava.
  - Ovaj metod se može koristiti za oslobađanje resursa koje koristi data instanca.

```
class Point:
   def __init__( self, x=0, y=0):
      self.x = x
      self.y = y
   def __del__(self):
      class_name = self.__class__.__name__
      print (class_name, "destroyed")
pt1 = Point()
pt2 = pt1
pt3 = pt1
print (id(pt1), id(pt2), id(pt3)) # prints the ids of the objects
```

```
7566056 7566056 7566056
Point destroyed
```

Python: nasleđivanje

```
del pt1
del pt2
del pt3    # probajte i bez ove 3 del naredbe
               7763112 7763112 7763112
               Point destroyed
```

- Nasleđivanje klasa omogućuje da se kreira nova klasa koja je izvedena iz postojeće klase.
- Izvedena klasa nasleđuje atribute roditeljske klase.
- Izvedena klasa može nadjačati članove roditeljske klase.
- Deklarisanje izvedene klase je kao što sledi:

```
class SubClassName (ParentClass1[, ParentClass2, ...]):
   'Optional class documentation string'
   class_suite
```

Primer:

```
class Parent: # define parent class
   parentAttr = 100
   def __init__(self):
      print ("Calling parent constructor")
   def parentMethod(self):
      print ('Calling parent method')
```

Python: nasleđivanje

```
    def setAttr(self, attr):
        Parent.parentAttr = attr
    def showAttr(self):
        print ("Parent attribute :", Parent.parentAttr)

class Child(Parent): # define child class
    def __init__(self):
        print ("Calling child constructor")
    def childMethod(self):
        print ('Calling child method')

c = Child()          # instance of child
c.childMethod()      # child calls its method
c.parentMethod()     # calls parent's method
c.setAttr(200)       # again call parent's method
c.showAttr()         # again call parent's method
```

```
Calling child constructor
Calling child method
Calling parent method
Parent attribute : 200
```

Izvođenje klase iz više osnovnih klasa je kao što sledi:

```
class A:        # define your class A  .....
class B:        # define your calss B  .....
class C(A, B): # subclass of A and B  .....
```

Python: nadjačavanje

- Funkcije za proveru odnosa dve klase ili instance:
  - **`issubclass(sub, sup),`**
    - vraća True ako je klasa **sub** potklasa superklase **sup**.
  - **`isinstance(obj, Class),`**
    - vraća True ako je instanca **obj** instanca klase Class ili je instanca potklase klase **Class**.
- Nadjačavanje metoda roditeljske klase vrši se ako se želi da dati metod u izvedenoj klasi ima drugačiju funkcionalnost.

```
# define parent class
class Parent:
    def myMethod(self):
        print ('Calling parent method')
# define child class
class Child(Parent):
    def myMethod(self):
        print ('Calling child method')
# instance of child
c = Child()
c.myMethod() # child calls overridden method
                          Calling child method
```

Python: predefinisane vrednosti argumenata konstruktora

- Predefinisane vrednosti argumenata konstruktora se realizuju isto kao i za obične funkcije:

```
class Dokument:
    """
    Klasa Dokument omogucuje instanciranje objekata Dokument
    """
    DokumentList=[]
    def __init__(self,name='Dokument',content=''):
        self.ime=name
        self.sadrzaj=content
        Dokument.DokumentList.append(self)
    def Info(self):
        print ('Naziv dokumenta:',self.ime)
        print ('Sadrzaj:\n',self.sadrzaj)

d1=Dokument('Esej','Prva recenica')
d2=Dokument('Blog01')
d3=Dokument()
for d in Dokument.DokumentList:
    d.Info()
```

```
Naziv dokumenta: Esej
Sadrzaj:
 Prva recenica
Naziv dokumenta: Blog01
Sadrzaj:

Naziv dokumenta: Dokument
Sadrzaj:
```

Python: preopterećenje operatora

```
from math import hypot
class Point:
    def __init__(self,x,y):
        self.x=x
        self.y=y
    def R(self): # funkcija koja vraca udaljenost tacke od ishodista
        return hypot(self.x, self.y)
    # Preopterecenі operatori poredjenja tacke prema udaljenosti od (0,0)
    def __lt__(self,other): # less then
        return self.R() < other.R()
    def __le__(self,other): # less or equal
        return self.R() <= other.R()
    def __gt__(self,other): # greather then
        return self.R() > other.R()
    def __ge__(self,other): # greater or equal
        return self.R() >= other.R()
p1=Point(2,1)
p2=Point(1,2)
p3=Point(3,0)
print (p1>p2) ;print (p1>=p2)
print (p1<=p2);print (p1<p3)
print (p1==p2)
```

```
False
True
True
True
False
```

Python: preopterećenje operatora

```
import math
class Pravougaonik:
    id=1
    def __init__(self,a=5.0,b=5.0):
        self.id = Pravougaonik.id
        Pravougaonik.id+=1
        self.a = a
        self.b = b
        # atribut povrsine
        self.Area = self.a*self.b
    def __str__(self):
        return 'Pravougaonik (id=%2d) = %4.2f x %4.2f , P = %6.2f' \
               % (self.id,self.a,self.b,self.Area)
    def Zameni(self):
        # stranice zamenjuju vrednosti, moze i kraci kod
        tmpA=self.a;     tmpB=self.b
        self.a=tmpB;     self.b=tmpA
    def __add__(self,other):
        # P1=P1+P2 , P1.a=P1.b
        self.Area = self.Area + other.Area
        self.a = self.b = math.sqrt(self.Area)
        return self
```

Python: preopterećenje operatora

```
 def __sub__(self,other):
        # P1=P1-P2 , P1.a=P1.b
        self.Area = math.fabs(self.Area - other.Area)
        self.a = self.b = math.sqrt(self.Area)
        return self
p1=Pravougaonik(2,3)
p2=Pravougaonik(4,8)
print (p1)
print (p2)
print(p1-p2)
print(p1+p2)
p2.Zameni()
print (p2)
```

```
Pravougaonik (id= 1) = 2.00 x 3.00 , P =   6.00
Pravougaonik (id= 2) = 4.00 x 8.00 , P =  32.00
Pravougaonik (id= 1) = 5.10 x 5.10 , P =  26.00
Pravougaonik (id= 1) = 7.62 x 7.62 , P =  58.00
Pravougaonik (id= 2) = 8.00 x 4.00 , P =  32.00
```

Python: nasleđivanje

```
class Road:
    def __init__(self,name,length):
        self.name=name
        self.length=length
    def Info(self):
        print ("Road definition")
        print ("Name:\t %15s" % self.name)
        print ("Length:\t %15s" % self.length)
class Highway(Road):
    pass
class MainRoad(Road):
    pass
class LocalRoad(Road):
    pass
h1=Highway('A1',1000)
m1=MainRoad('E54',540)
l1=LocalRoad('S16',125)
h1.Info()
m1.Info()
l1.Info()
```

```
Road definition
Name:                    A1
Length:                1000
Road definition
Name:                   E54
Length:                 540
Road definition
Name:                   S16
Length:                 125
```

Python: nasleđivanje, pozivanje konstruktora nadklase

```
class Road:
    def __init__(self,name,length):
        self.name=name
        self.length=length
    def __str__(self):
        strinfo="Road definition\n"+\
        ("Name:\t %15s\n" % self.name)+\
        ("Length:\t %15s\n" % self.length)
        return strinfo
class Highway(Road):
    def __init__(self,name,length):
        # poziv konstruktora natklase
        Road.__init__(self,name,length)
        self.type='highway'
        self.toll=50 # putarina
class MainRoad(Road):
    def __init__(self,name,length): # override
        Road.__init__(self,name,length) # invokacija
        self.type='mainroad'
class LocalRoad(Road):
    def __init__(self,name,length): # override
        Road.__init__(self,name,length) # invokacija
        self.type='localroad'
```

Python: nasleđivanje , pozivanje konstruktora nadklase

```
hw=Highway('H01',1000)
mr=MainRoad('M02',540)
lr=LocalRoad('L03',125)
print(hw)
print(mr)
print(lr)
print('Toll = ',hw.toll)
```

```
Road definition
Name:                   H01
Length:                 1000

Road definition
Name:                   M02
Length:                 540

Road definition
Name:                   L03
Length:                 125

Toll =  50
```