PROGRAMIRANJE U INTEGRISANIM TEHNOLOGIJAMA

Oznaka predmeta: PIT

Predavanje broj: 03

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). Preklapanje operatora. Dekoratori. Skrivanje podataka. Regularni izrazi (madifikatori, zagrade, kvantifikatori, metakarakteri, match, search, sub). CGI (HTTP, varijable okruženja, get, post, forme, upload).

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 exceptions,
   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 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 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)
    # Preopterecni 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
```

Python, preklapanje operatora

- Neke metode tipa `__naziv__`

  `__init__ ( self [,args...] )`,

  konstruktor (sa opcionalnim argumentima), kreira instancu klase

  primer: `obj = className(args)`

  `__del__( self )`,

  destruktor, briše objekat,

  primer: `del obj`

  `__repr__( self )`,

  string reprezentacija (sadrži __str__),

  primer. : `repr(obj)`

  `__str__( self )`,

  printabilna string reprezentacija,

  primer: `str(obj)`

  `__cmp__ ( self, x )`,

  poređenje objekta,

  primer: `cmp(obj,x)`

Python: str, repr

- Objašnjenje str i repr:

```
>>> import datetime
>>> today = datetime.datetime.now()
>>> str(today)
'2012-03-14 09:21:58.130922'
>>> repr(today)
'datetime.datetime(2012, 3, 14, 9, 21, 58, 130922)'

>>> class Sic:
...   def __repr__(self): return 'foo'
...
>>> print str(Sic())
foo
>>> print repr(Sic())
foo
>>> class Sic:
...   def __str__(self): return 'foo'
...
>>> print str(Sic())
foo
>>> print repr(Sic())
<__main__.Sic object at 0x2617f0>
>>>
```

Python, preklapanje operatora

Preklapanje operatora

- Neka je kreirana klasa (2D) Vector koja prikazuje dvodimenzionalni vektor.
- Ideja je da se implementira metoda __add__.
- Može se definisati __add__ metoda u klasi koji će da obavlja sabiranje 2D vektora, tako da se može jednostavno pisati umesto __add__ samo znak +.

```
class Vector:
   def __init__(self, a, b):
      self.a = a
      self.b = b
   def __str__(self):
      return 'Vector (%d, %d)' % (self.a, self.b)
   def __add__(self,other):
      return Vector(self.a + other.a, self.b + other.b)
v1 = Vector(2,10)
v2 = Vector(5,-2)
print (v1 + v2)
```

```
Vector(7,8)
```

Python: skrivanje podataka

Skrivanje podatka

- Atributi objekta mogu ili ne mogu biti vidljivi izvan definicije klase
- Ako se nazivu atributa doda prefiks dve donje crte, takav atribut postaje direktno nedostupan izvan klase.
  - Sa jednom donjom crtom nedostupan je u smislu poštovanja dogovora (konvencije, *weak "internal use" indicator*).
    Ako bi uradili

    ```
    from M import *
    ```

    ne bi bili importovani objekti iz M čije ime počinje sa jednom donjom crtom.

```
class JustCounter:
    __secretCount = 0
    def count(self):
        self.__secretCount += 1
        print (self.__secretCount)

counter = JustCounter()
counter.count()
counter.count()
print (counter.__secretCount)
```

```
1
2
Traceback (most recent call last):
   File "test.py", line 12, in
<module>
      print counter.__secretCount
AttributeError: JustCounter instance
has no attribute '__secretCount'
```

Python: dekoratori

- Python štiti prikazani član tako što interno menja njegovo ime tako da ono uključuje ime klase.
  - Sada se pristup atributu može izvesti kao: *object._className__attrName*.
- Izlaz izvršavanja prethodnog koda će biti 1, 2, 2, repsektivno ako poslednju liniju tog koda zamenimo sa:

```
print (counter._JustCounter__secretCount)
```

```
1
2
2
```

Dekoratori @classmethod i @staticmethod

Za korišćenje zajedničkih metoda klase u ranijim verzijama se koristila funkcija koja je metodu "prevodila" u statičku ili klasnu (npr. nakon definicije funkcije f unutar date klase sledilo je f=classmethod(f) ili f=staticmethod(f)).
Sada se koriste dekoratori.

```
class Roditelj:
  @staticmethod
  def staticki_pozdrav(): print("Roditeljski staticki pozdrav !")
  @classmethod
  def klasni_pozdrav(cls):
     if(cls.__name__=="Sin"):          print("Zdravo sine !")
     elif(cls.__name__=="Kcerka"):     print("Zdravo kceri !")
     elif(cls.__name__=="Roditelj"): print("Roditeljski klasni
pozdrav !")
     else: print("Zdravo ???")
```

Python: dekoratori

```
class Sin(Roditelj):
  def PozdravPrekoObjekta(self):
        print("Sin pozdravlja")

  @staticmethod
  def staticki_pozdrav():
        print("Sinov staticki pozdrav !")

  @classmethod
  def klasni_pozdrav(cls):
       if(cls.__name__=="Sin"):
              print("Sinov klasni pozdrav !")
       else:
              print("Neka je druga klasa");

class SinovSin(Sin):
       pass

class Kcerka(Roditelj):
  def PozdravPrekoObjekta(self):
        print("Kcerka pozdravlja")
```

Python: dekoratori

```
Roditelj.staticki_pozdrav() #Roditeljski staticki pozdrav !
Roditelj.klasni_pozdrav()   #Roditeljski klasni pozdrav !
roditelj = Roditelj()       #
roditelj.staticki_pozdrav() #Roditeljski staticki pozdrav !
roditelj.klasni_pozdrav()   #Roditeljski klasni pozdrav !
print()
Sin.staticki_pozdrav()      #Sinov staticki pozdrav !
Sin.klasni_pozdrav()        #Sinov klasni pozdrav !
sin = Sin()                 #
sin.staticki_pozdrav()      #Sinov staticki pozdrav !
sin.klasni_pozdrav()        #Sinov klasni pozdrav !
print()
SinovSin.staticki_pozdrav() #Sinov staticki pozdrav !
SinovSin.klasni_pozdrav()   #Neka je druga klasa
print()
Kcerka.staticki_pozdrav()   #Roditeljski staticki pozdrav !
Kcerka.klasni_pozdrav()     #Zdravo kceri !
kcerka = Kcerka()           #
kcerka.staticki_pozdrav()   #Roditeljski staticki pozdrav !
kcerka.klasni_pozdrav()     #Zdravo kceri !
```

Python: regularni izrazi

Regularni izrazi
predstavljaju specijalnu sekvencu karaktera (uzorak, pattern) koji pomažu da se nađe string ili skup stringova koji odgovaraju kriterijumu predstavljenom na ovaj način.

- Modul **re** (**r**egular-**e**xpression) omogućuje punu podršku regularnih izraza u Python-u.
- Funkcija re.match() pokušava da pronađe poklapanje na početku stringa sa pattern-om:

```
re.match(pattern, string, flags=0)
```

  - pattern regularni izraz za poklapanje.
  - string string koji se pretražuje u smislu poklapanja sa datim uzorkom.
  - flags modifikatori.

  poklapanje može biti uspešno ili neuspešno.

Python: re, match

- Funkcija *re.match* vraća **match** objekat ako je uspešno poklapanje ili **None** ako poklapanje nije uspešno.
- Za uzimanje rezultata poklapanja **match** objekta koriste se funkcije:
  - group(num=0), vraća rezultat poklapanja za datu grupu (ili specifičnog podgrupnog broja - num)
  - groups(), vraća sve podgrupe koje se poklapaju sa uzorkom kao n-torku (praznu ako nema poklapanja)

```
import re
line = "Cats are smarter than dogs"
matchObj = re.match( r'(.*) are (.*?) .*', line, re.M|re.I)
if matchObj:
   print ("matchObj.group( ) : ", matchObj.group( ))
   print ("matchObj.group(1) : ", matchObj.group(1))
   print ("matchObj.group(2) : ", matchObj.group(2))
else:
   print ("No match!!")
```

```
matchObj.group( ) : Cats are smarter than dogs
matchObj.group(1) : Cats
matchObj.group(2) : smarter
```

Python: re, search

- Funkcija *search* traži prvo pojavljivanje uzorka (*pattern*) unutar stringa (*string*) sa opcionalnim zastavicama (*flags*).

```
re.search(pattern, string, flags=0)
```

pattern,	regularni izraz koji predstavlja uzorak za poklapanje.
string,	string u kom se traži poklapanje sa uzorkom.
flags,	modifikatori.

- Funkcija *re.search()* ako je uspešna vraća **match** objekat ili **None** ako ne nađe uzorak.

```
import re
line = "Cats are smarter than dogs"
searchObj = re.search( r'(.*) are (.*?) .*', line, re.M|re.I)
if searchObj:
   print ("searchObj.group( ) : ", searchObj.group())
   print ("searchObj.group(1) : ", searchObj.group(1))
   print ("searchObj.group(2) : ", searchObj.group(2))
else:
   print ("Nothing found!!")
```

```
searchObj.group( ) : Cats are smarter than dogs
searchObj.group(1) : Cats
searchObj.group(2) : smarter
```

Python: re.match, re.search, re.sub

- Razlika između search i match je u tome što u uzorku **match** proverava poklapanje samo na početku stringa dok **search** proverava poklapanje bilo gde u stringu.

```
import re
line = "Cats are smarter than dogs"

matchObj = re.match( r'dogs', line, re.M|re.I)
if matchObj:
   print ("match --> matchObj.group() : ", matchObj.group() )
else:
   print ("No match!!")

searchObj = re.search( r'dogs', line, re.M|re.I)
if searchObj:
   print ("search --> searchObj.group() : ", searchObj.group() )
else:
   print ("Nothing found!!")
```

```
No match!!
search --> matchObj.group() : dogs
```

- Jedna od bitnih metoda je sub (substitution) koja zamenjuje sve pojave uzorka ili do maksimalno zadatog broja zamena:

```
re.sub(pattern, repl, string, max=0)
```

ovaj metod vraća modifikovani string.

Python: re.sub, modifikatori

```
import re
phone = "2004-959-559 # This is Phone Number"
# Delete Python-style comments
num = re.sub(r'#.*$', "", phone)
print ("Phone Num : ", num)
# Remove anything other than digits
num = re.sub(r'\D', "", phone)
print ("Phone Num : ", num)
```

```
Phone Num :  2004-959-559
Phone Num :  2004959559
```

Modifikator	Opis
re.I	case-insensitive poklapanje.
re.L	Interpretira reči prema tekućoj lokalizaciji (utiče na alfabetsku grupu \w \W te na \b \B).
re.M	Multilinijsko poklapanje (utiče na ^ $).
re.S	Znak tačka (dot, period) poklapa svaki karakter uključujući i *newline*.
re.U	Interpretira slova prema Unicode-u (utiče na \w, \W, \b, \B, \d \D, \s \S).
re.X	Bolja čitljivost uzorka jer ignoriše beline (osim ako su navedene u uzorku) i omogućuje stavljanje komentara (#) u uzorak.

Python: uzorci

Uzorak	Opis
^	Poklapanje na početku linije.
$	Poklapanje od kraju linije.
.	Poklapanje bilo kog karaktera osim newline (osim ako je re.S).
[...]	Poklapanje bilo kog navedenog karaktera.
[^...]	Nepoklapanje bilo kog navedenog karaktera.
n?	Poklapanje sa stringom koji sadrži 0 ili jedan n.
n*	Poklapanje sa stringom koji sadrži 0, 1 ili više pojavljivanja n.
n+	Poklapanje sa stringom koji sadrži bar jedan n.
n{ x}	Pronalazi string koji sadrži sekvencu od x n-ova.
n{ min,}	Pronalazi string koji sadrži sekvencu od bar min n-ova.
n{ min, max}	Pronalazi string koji sadrži sekvencu od min do max n-ova.
a| b	Poklapanje sa a ili b.
(re)	Poklapanje sa grupom regularnih izraza.
(?imx)	Privremeno menja (uključenjem) i, m, ili x opciju unutar regularnog izraza.

Python: re, uzorci

Uzorak	Opis
(?-imx)	Privremeno menja (isključenjem) opcije i, m ili x unutar regularnog izraza.
(?: re)	Kreira non-capturing grupu. npr. (?:abc){3} -> abcabcabc. Nema grupa.
(?#...)	Postavljanje komentara unutar regularnog izraza npr. \d(?#digit)
(?= n)	Pronalazi string iza koga sledi n.
(?! n)	Pronalazi string iza koga ne sledi n.
(?> re)	Atomska grupa, optimizuje traženje, npr. (?>his|this) u reči 'smashing' ako ne nađe ***his*** neće tražiti ni ***this.***
\w	Nađi slovo.
\W	Nađi što nije slovo.
\s	Whitespace, isto kao [\t\n\r\f].
\S	Nonwhitespace.
\d	Nađi cifru (isto kao [0-9]).

Python: re uzorci

Uzorak	Opis
\D	Nađi što nije cifra.
\A	Poklapanje samo na početku stringa.
\Z	Poklapanje samo na kraju stringa (za newline proverava pre njega)
\z	Poklapanje na kraju stringa.
\G	Nastavlja proveru gde je prethodna provera stala. pr. \G\w u 'this this' daće ponavljanjem uzorka **t h i s failed**.
\b	Nađi poklapanje na početku ili kraju reči.
\B	Nađi što nema poklapanje na početku ili kraju reči.
\n, \t,	Poklapanje sa *newlines*, *tabs*, itd.
\1...\9	Poklapanje n-te grupe podizraza.

- Primeri regularnih izraza:

```
a = re.compile(r"""\d+ # the integral part
                   \.  #the decimal point
                   \d* # some fractional digits""", re.X)
b = re.compile(r"\d+\.\d*") #VERBOSE
```

  - python poklapanje: "python".
  - [Pp]ython poklapanje: "Python" ili "python"
  - rub[ye] poklapanje: "ruby" ili "rube"

Python: re uzorci

- [aeiou] poklapanje: bilo koji mali samoglasnik
- [0-9] poklapanje: bilo koja cifra; isto kao [0123456789]
- [a-z] poklapanje: bilo koje malo ASCII slovo
- [A-Z] poklapanje: bilo koje veliko ASCII slovo
- [a-zA-Z0-9] poklapanje: bilo šta od navedenog
- [^aeiou] poklapanje: bilo šta različito od malih samoglasnika
- [^0-9] poklapanje: bilo šta različito od cifre

- Specijalni karakteri:
  - . poklapanje: bilo koji karakter osim *newline* (osim ako je dat re.S)
  - \d isto kao [0-9]
  - \D isto kao [^0-9]
  - \s poklapanje praznina, isto kao [\t\r\n\f]
  - \S poklapanje nepraznina, isto kao [^ \t\r\n\f]
  - \w poklapanje slova, isto kao [A-Za-z0-9_]
  - \W nađi što nije slovo, isto kao [^A-Za-z0-9_]

Python: re uzorci

- Ponavljanja:
  - ruby? poklapanje: "rub", "ruby"
  - ruby* poklapanje: "rub", "ruby", "rubyy", ...
  - ruby+ poklapanje: "ruby", "rubyy", ...
  - \d{3} poklapanje: tačno 3 cifre
  - \d{3,} poklapanje: 3 ili više cifara
  - \d{3,5} poklapanje: 3, 4 ili 5 cifara
- Pohlepno i nepohlepno poklapanje
  - <.*> Pohlepno (greedy) ponavljanje: poklapanje "<**python>perl**>"
  - <.*?> Nongreedy: matches "<python>" in "<**python**>perl>"
- Grupisanje sa zagradama:
  - \D\d+ bez grupe: + ponavlja \d
  - (\D\d)+ grupisano: + ponavlja par \D\d
  - ([Pp]ython(,)?)+ poklapanje: "Python", "Python, python, python", ...
- Povratne reference (*backreferences*):
  - ([Pp])ython&\1ails poklapanje: python&pails ili Python&Pails
  - (['"])[^\1]*\1 string sa apostrofima ili navodnicima.
    \1 odnosi se na prvu grupu

Python: re uzorci

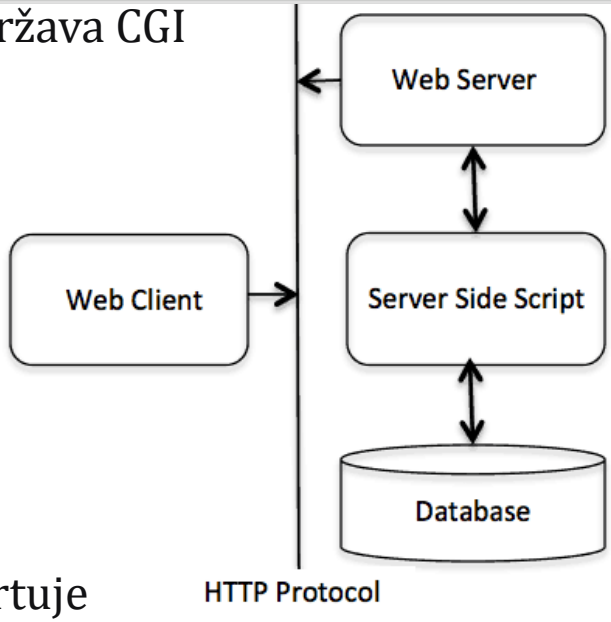
- Alternative:
  - python|perl — poklapanje: "python" ili "perl"
  - rub(y|le) — poklapanje: "ruby" ili "ruble"
  - Python(!+|\?) — poklapanje: iza "Python" sledi jedan ili više ! ili jedan ?
- Ankeri (specificiraju poziciju poklapanja):
  - ^Python — poklapanje: "Python" je na početku stringa.
  - Python$ — poklapanje: "Python" je na kraju stringa.
  - \APython — poklapanje: "Python" je **samo** na početku stringa.
  - Python\Z — poklapanje: "Python" je **samo** na kraju stringa.
  - \BPython\B — poklapanje: "Python" je unutar reči.
  - \brub\B — poklapanje: "rub" npr. u reči "rube" i "ruby"
  - Python(?=!) — poklapanje: "Python", ako je iza znak !
  - Python(?!!) — poklapanje: "Python", ano iza nema znak !
- Posebna sintaksa sa zagradama:
  - R(?#comment) — poklapanje: "R". Sve ostalo je komentar.
  - R(?i)uby — case-**i**nsensitive dok se traži "uby"
  - R(?i:uby) — kao iznad

Python: CGI

- Common Gateway Interface (CGI) je skup standarda koji definišu kako se informacije razmenjuju između web server-a i korisničkog skript-a.
- Primer: kada korisnik klikne na hiperlink da bi pregledao određenu web stranu ili URL dešava se sledeće:
  - browser kontaktira HTTP web server i zahteva URL (ime fajla).
  - Web server parsira URL i traži ime fajla.
    - ako nađe zahtevani fajl onda ga šalje nazad ka browser-u,
    - ako ne nađe zahtevani fajl onda šalje poruku o grešci indicirajući da se tražio pogrešan fajl.
  - Web browser uzima odgovor od web server-a i prikazuje ili primljeni fajl ili poruku o grešci.
  - Međutim, moguće je podesiti HTTP server tako da ako se traži fajl u nekom datom direktorijumu da se fajl ne prosleđuje nazad nego da se umesto toga taj program izvrši i da se rezultat tog programa vrati nazad browser-u i da se isti prikaže.
  - Ova funkcija se naziva Common Gateway Interface (CGI) a programi se zovu CGI skriptovi (Python Script, PERL Script, Shell Script, C, C++,...).

Python: CGI

- Za Web Server prvo se proveri da li Web Server podržava CGI i da li je konfigurisan da rukuje CGI programima.
- Svi CGI programi koje izvršava HTTP server su smešteni u prekonfigurisani direktorijum koji se naziva CGI Directory i po konvenciji to je /var/www/cgi-bin.
- Po konvenciji CGI fajlovi imaju ekstenziju .cgi, ali može se ostaviti da Python-ovi fajlovi imaju ekstenziju .py.



- Podrazumevano Linux server je konfigurisan da startuje samo skriptove u direktorijumu cgi-bin u /var/www (za specifikaciju nekog drugog direktorijuma izvrše se izmene u httpd.conf fajlu)

```
<Directory "/var/www/cgi-bin">
  AllowOverride None
  Options ExecCGI
  Order allow,deny
  Allow from all
</Directory>
<Directory "/var/www/cgi-bin">
  Options All
</Directory>
```

Python: CGI

- Sledi primer CGI programa (hello.py) koji se nalazi u /var/www/cgi-bin direktorijumu i koji će se zahtevati jednostavnim linkom iz browser-a:

```
#!/usr/bin/python
print ("Content-type:text/html\r\n\r\n")
print ('<html>' )
print ('<head>' )
print ('<title>Hello Word - CGI Program</title>' )
print ('</head>' )
print ('<body>' )
print ('<h2>Hello Word! This is CGI program</h2>' )
print ('</body>' )
print ('</html>' )
```

Hello Word! This is CGI program

- Prva linija koda kada stigne do browser-a specificira da se sadržaj ispiše na ekran (Content-type:text/html\r\n\r\n). Ova linija je deo HTTP zaglavlja koje se šalje browser-u da shvati koji se sadržaj šalje.
- HTTP zaglavlje ima sledeću formu:

HTTP **Field Name:** Field Content

npr. **Content-type:** text/html\r\n\r\n

Python: CGI

- Neka značajna HTTP zaglavlja (header-i):

Header	Opis
`Content-type:`	MIME string koji definiše format fajla koji se vraća. Npr. Content-type:text/html
`Expires: Date`	Koristi se da browser odluči kada da se osveži stranica. Validni format je npr. 01 Jan 1998 12:00:00 GMT.
`Location: URL`	URL koji se vraća umesto zahtevanog URLa. Koristi se za redirekciju zahteva za fajlom.
`Last-modified: Date`	Vreme poslednje modifikacije resursa.
`Content-length: N`	Dužina podataka u bajtovima koja će biti vraćena. Koristeći ovu informaciju browser može dati procenjeno vreme za učitavanje.
`Set-Cookie: String`	Postavljanje cookie-a prosleđenog kao string.

Python: CGI environment variables

Variabla	Opis
CONTENT_TYPE	Tip podataka sadržaja. Koristi se npr. kada klijent šalje attach-ovan sadržaj ka serveru (npr. file upload).
CONTENT_LENGTH	Dužina upita za POST zahteve.
HTTP_COOKIE	Vraća skup cookie-a u formi key-value.
HTTP_USER_AGENT	Naziv WEB browser-a.
PATH_INFO	Staza CGI skripta.
QUERY_STRING	URL-kodovana informacija koja se šalje sa GET zahtevom.
REMOTE_ADDR	IP adresa klijenta.
REMOTE_HOST	Naziv hosta klijenta.
REQUEST_METHOD	Metod zahteva (npr. GET, POST).
SCRIPT_FILENAME	Puna staza CGI skripta.
SCRIPT_NAME	Naziv CGI skripta.
SERVER_NAME	Naziv servera (hostname ili IP adresa).
SERVER_SOFTWARE	Naziv i verzija softvera na strani servera.

Python: CGI, get, post

- Program koji izlistava sve CGI varijable:

```
#!/usr/bin/python
import os
print ("Content-type: text/html\r\n\r\n")
print ("<h1>Environment</h1><\br>")
for param in os.environ.keys():
   print ("<b>%20s</b>: %s<\br>" % (param, os.environ[param]))
```

GET i POST metode

- Slanje informacije GET metodom:
  - GET je podrazumevani metod koji šalje kodovane korisničke informacije dodate zahtevu za stranicom u formi parova ključ=vrednost (name=value) sa delimiterom & koje se vide u adresnom baru.
  - U zahtevu su nakon navoda stranice ove informacije odvojene znakom ?

```
http://www.test.com/cgi-bin/hello.py?key1=value1&key2=value2
```

  - Ovim metodom se ne šalju osetljive informacije (npr. password) ka serveru.
  - GET metod ima ograničenje u količini podataka: 2-8 KB po zahtevu.
  - GET metod šalje infromacije korišćenjem QUERY_STRING zaglavlja (dostupno preko QUERY_STRING promenljive okruženja).

Python: CGI, get, post

- Primer: slanje vrednosti metodom GET:

  `localhost/cgi-bin/hello_get.py?first_name=ZARA&last_name=ALI`

program hello_get.py:

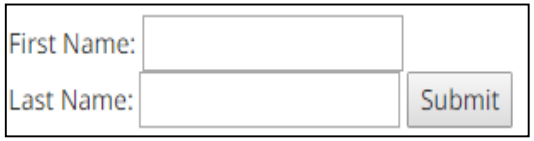
```
#!/usr/bin/python
# Import modules for CGI handling
import cgi, cgitb
#Traceback      Create instance of FieldStorage
cgitb.enable();
form = cgi.FieldStorage()
# Get data from fields
first_name = form.getvalue('first_name')
last_name = form.getvalue('last_name')
print ("Content-type:text/html\r\n\r\n" )
print ("<html>" )
print ("<head>" )
print ("<title>Hello - Second CGI Program</title>" )
print ("</head>" )
print ("<body>" )
print ("<h2>Hello %s %s</h2>" % (first_name, last_name) )
print ("</body>" )
print ("</html>")
```

Hello ZARA ALI

Python: CGI, forme

- Slanje vrednosti može se obaviti i HTML formom:

```
<form action="http://localhost/cgi-bin/hello_get.py" method="get">
  First Name: <input type="text" name="first_name"> <br />
  Last Name: <input type="text" name="last_name" />
  <input type="submit" value="Submit" />
</form>
```



- POST metod šalje informacije kao odvojenu poruku.
  Program hello_get.py je isti kao na prethodnom slajdu, samo je sada HTML forma sa navedenom POST metodom. Izgled forme se ovim ne menja.

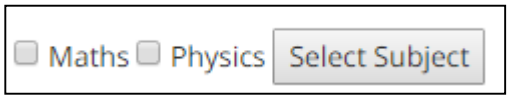
```
<form action="http://localhost/cgi-bin/hello_get.py" method="post">
  First Name: <input type="text" name="first_name"><br />
  Last Name: <input type="text" name="last_name" />
  <input type="submit" value="Submit" />
</form>
```

- Slanje checkbox podataka je kao što sledi
  (npr. action="http://127.0.0.1/cgi-bin/checkbox.py"):

```
<form action="http://localhost/cgi-bin/checkbox.py" method="POST"
target="_blank">
  <input type="checkbox" name="maths" value="on" /> Maths
  <input type="checkbox" name="physics" value="on" /> Physics
  <input type="submit" value="Select Subject" />    </form>
```

Python: CGI, forme

- Sledi kod checkbox.cgi skripta koji rukuje ulaznim podacima koji se dobijaju od web browser-a za checkbox dugmad.

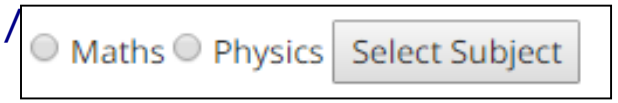



```
#!/usr/bin/python
# Import modules for CGI handling
import cgi, cgitb
#Traceback      Create instance of FieldStorage
cgitb.enable(); form = cgi.FieldStorage()
# Get data from fields
if form.getvalue('maths'):   math_flag = "ON"
else:                        math_flag = "OFF"
if form.getvalue('physics'): physics_flag = "ON"
else:                        physics_flag = "OFF"
print ("Content-type:text/html\r\n\r\n")
print ("<html>" )
print ("<head>" )
print ("<title>Checkbox - Third CGI Program</title>" )
print ("</head>" )
print ("<body>" )
print ("<h2> CheckBox Maths is : %s</h2>" % math_flag )
print ("<h2> CheckBox Physics is : %s</h2>" % physics_flag )
print ("</body>" )
print ("</html>")
```

Python: CGI, forme

- Slanje infomacija o radio dugmadima (radio button):

```
<form action="http://localhost/cgi-bin/radiobutton.py" method="post"
target="_blank">
    <input type="radio" name="subject" value="maths" /> Maths
    <input type="radio" name="subject" value="physics" /> Physics
    <input type="submit" value="Select Subject" />
</form>
```



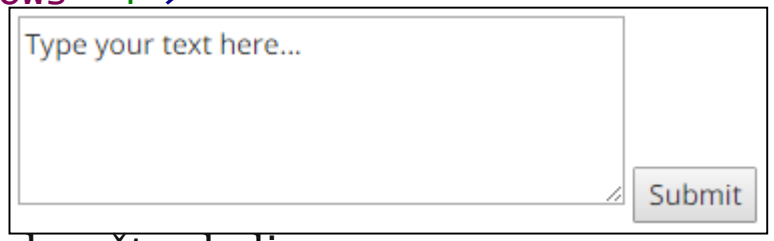
- Pripadni kod radiobutton.py skripta je kao što sledi:

```
#!/usr/bin/python
# Import modules for CGI handling
import cgi, cgitb
#Traceback      Create instance of FieldStorage
cgitb.enable(); form = cgi.FieldStorage()
# Get data from fields
if form.getvalue('subject'): subject = form.getvalue('subject')
else:                        subject = "Not set"
print ("Content-type:text/html\r\n\r\n" )
print ("<html>"); print ("<head>" )
print ("<title>Radio - Fourth CGI Program</title>" )
print ("</head>" ); print ("<body>")
print ("<h2> Selected Subject is %s</h2>" % subject )
print ("</body>"); print ("</html>" )
```

Python: CGI, forme

- Forma za slanje TEXTAREA (višelinijskog) sadržaja je kao što sledi:

```
  <form action="http://localhost/cgi-bin/textarea.py" method="post"
target="_blank">
    <textarea name="textcontent" cols="40" rows="4">
      Type your text here...
    </textarea>
  <input type="submit" value="Submit" />
  </form>
```



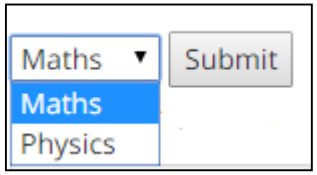
- Pripadni kod za rukovanje podacima textarea je kao što sledi:

```
#!/usr/bin/python
# Import modules for CGI handling
import cgi, cgitb
#Traceback      Create instance of FieldStorage
cgitb.enable();
form = cgi.FieldStorage()
if form.getvalue('textcontent'):
   text_content = form.getvalue('textcontent')
else: text_content = "Not entered"
print ("Content-type:text/html\r\n\r\n");print("<html>");print("<head>")
print ("<title>Text Area - Fifth CGI Program</title>"); print("</head>")
print ("<body>")
print ("<h2> Entered Text Content is %s</h2>" % text_content)
print ("</body>"); print ("</html>")
```

Python: CGI, forme

- Slanje podatka iz selekcione liste (drop-down box):

```
  <form action="http://localhost/cgi-bin/dropdown.py" method="post"
target="_blank">
    <select name="dropdown">
      <option value="Maths" selected>Maths</option>
      <option value="Physics">Physics</option>
    </select>
    <input type="submit" value="Submit"/>
  </form>
```



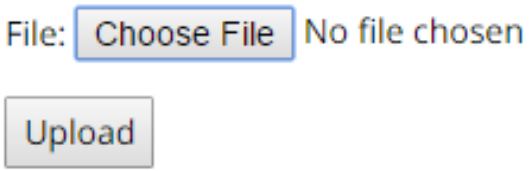
- Pripadni kod za prihvatanje i obradu podatka odabranog iz selekcione liste je:

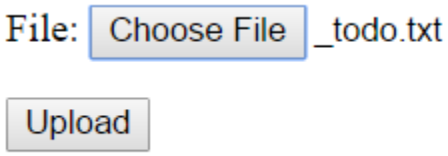
```
#!/usr/bin/python
import cgi, cgitb
cgitb.enable(); form = cgi.FieldStorage()
if form.getvalue('dropdown'): subject = form.getvalue('dropdown')
else: subject = "Not entered"
print ("Content-type:text/html\r\n\r\n")
print ("<html>")
print ("<head>")
print ("<title>Dropdown Box - Sixth CGI Program</title>")
print ("</head>")
print ("<body>")
print ("<h2> Selected Subject is %s</h2>" % subject)
print ("</body>"); print ("</html>")
```

Python: CGI, upload

- Uploadovanje fajlova se obavlja tako da HTML forma ima postavljen atribut **enctype=multipart/form-data**.
- Input tag tipa file type kreira dugme "Browse".

```
<html>
<body>
<form enctype="multipart/form-data"
      action="http://127.0.0.1/cgi-bin/save_file.py" method="post">
  <p>File: <input type="file" name="filename" ></p>
  <p><input type="submit" value="Upload" ></p>
</form>
</body>
</html>
```





Python: CGI, upload

- Sledi skript **save_file.py** koji rukuje uploadom fajla:

```
#!C:/Users/Pero/AppData/Local/Programs/Python/Python35/python.exe
import cgi, os
import cgitb # Traceback manager.
cgitb.enable()
form = cgi.FieldStorage()
# Get filename here.
fileitem = form['filename']
# Test if the file was uploaded
if fileitem.filename:
  # strip leading path from file name to avoid
  # directory traversal attacks
  fn = os.path.basename(fileitem.filename)
  fp=open( fn, 'wb'); fp.write(fileitem.file.read()); fp.close()
  message = 'The file ' + fn + ' was uploaded successfully'
else:
  message = 'No file was uploaded'
print ("Content-Type: text/html\r\n\r\n <html> <body> <p>%s</p> </body>
</html> " % (message,))
```

- Na Linux-u bi trebalo postaviti gore navedenu liniju koda za funkciju open kao:
  `fn = os.path.basename(fileitem.filename.replace("\\", "/" ))`

Download fajla

```
#!C:/Users/Peeet/AppData/Local/Programs/Python/Python35/python.exe

import cgitb
cgitb.enable()

print ("Content-Type: text/plain")
print ("Content-Disposition:attachment;filename=_dovucen_fajl.txt")
print ()

filename = "_todo.txt"
f = open(filename, 'r')
for line in f:
    print (line)
f.close()
```