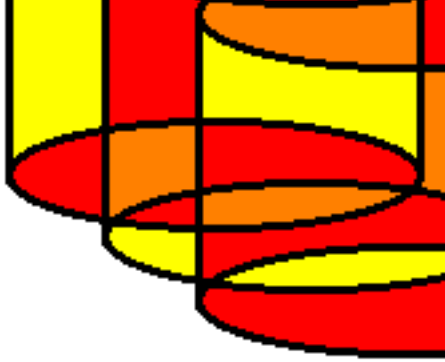
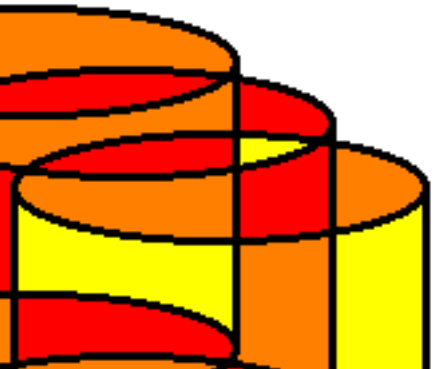
Objektno-orijentirane baze podataka

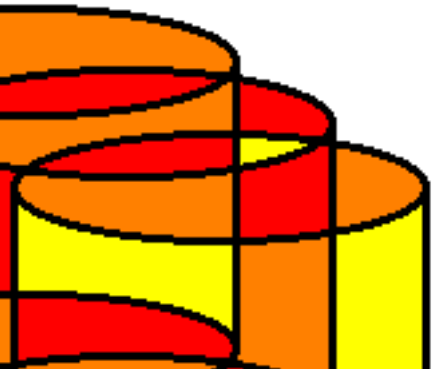


- Potreba za postizanjem što veće sličnosti baze podataka (modela) s aplikacijskom domenom (dijelom 'realnog' svijeta) doveo je do objektno-orijentiranog pristupa, pa tako i objektno-orijentiranih baza podataka.

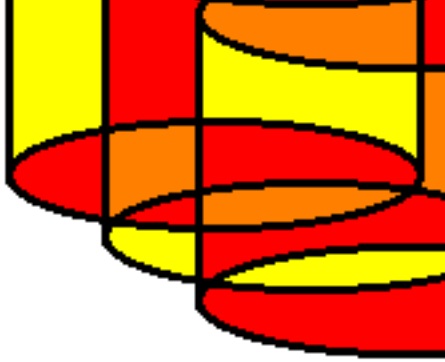


Pristupi

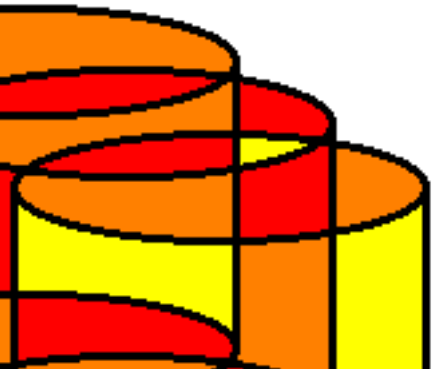
- Postoji velik broj različitih pristupa objektno-orijentiranim bazama podataka koji se najčešće vežu uz određeni objektno-orijentirani programski jezik.
- Mi ćemo objektno-orijentirane baze podataka proučiti na primjeru sustava **ZODB** (Zope Object Data Base) koji omogućava pohranu objekata u programskom jeziku Python.



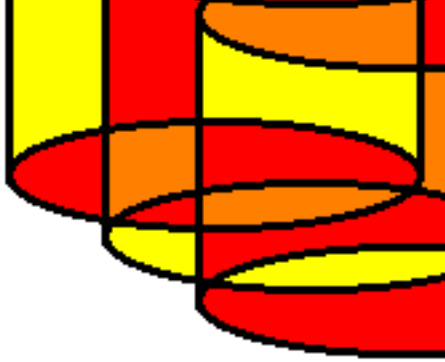
Motivacija za uvođenje OOBP



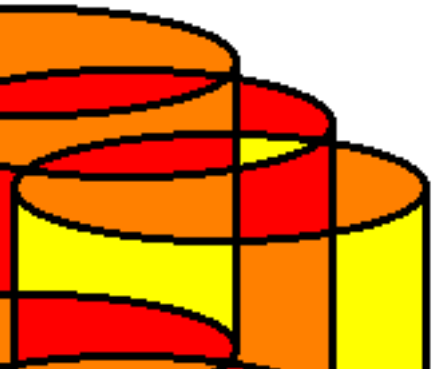
- Objektno-orijentirana aplikacija tijekom svog izvođenja stvara i manipulira objektima.
- Zatvaranjem aplikacije (procesa) objekti se uništavaju (brišu iz radne memorije).
- Ponovnim pokretanjem aplikacije potrebno je nanovo stvoriti sve objekte potrebne za rad aplikacije.
- Stoga je potreban nekakav mehanizam koji će omogućiti pohranu podataka prije kraja procesa te njihovo ponovno čitanje prilikom ponovnog pokretanja.



ZODB

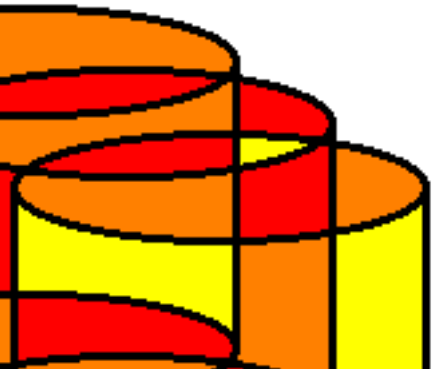


- ZODB automatizira taj postupak na transparentan način
  - nema potrebe za eksplicitnim kreiranjem svakog objekta
  - objekti su pohranjeni u objektnoj bazi podataka

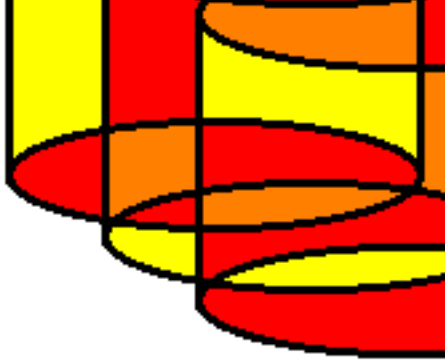


Upute

- U nastavku slijedi nekoliko pitanja i primjera
- Odgovore na pitanja pišite u odgovoru na zadatak, a primjere u python konzoli
- Kopiju python konzole pohranite u datoteku **ime_prezime.txt**

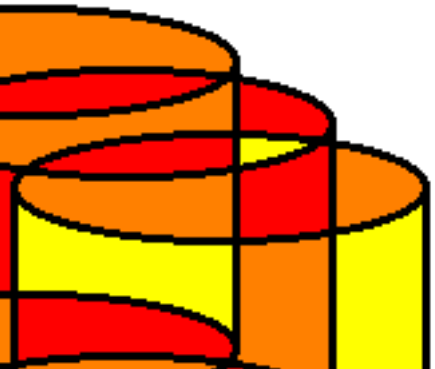


Osnovni koncepti objektno-orijentiranog pristupa

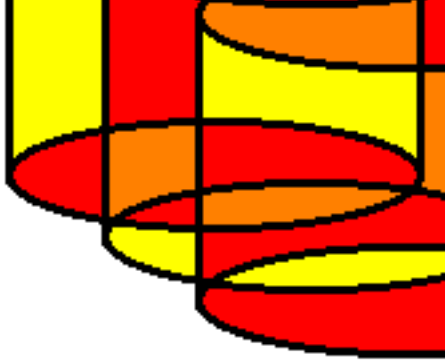


Ponovite definicije sljedećih OO koncepata te ih uočite u sljedećim primjerima:

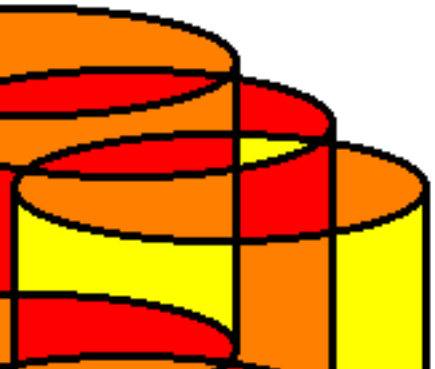
- klase?
- objekti?
- atributi?
- metode?
- nasljeđivanje?



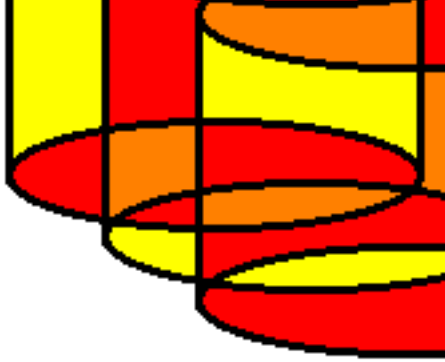
Primjer



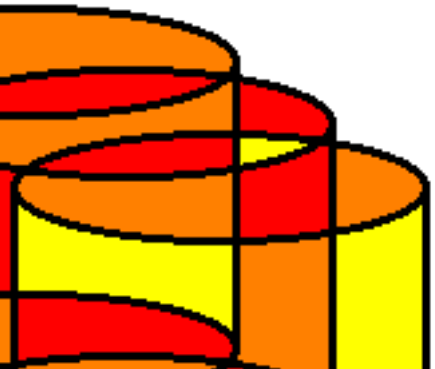
```
class osoba:
     def __init__( self, sifra, ime, prezime, adresa=None):
        self.sifra   = sifra
        self.ime     = ime
        self.prezime = prezime
        self.adresa  = adresa

     def __str__( self ):
        return '%s %s' % ( self.ime, self.prezime )
```



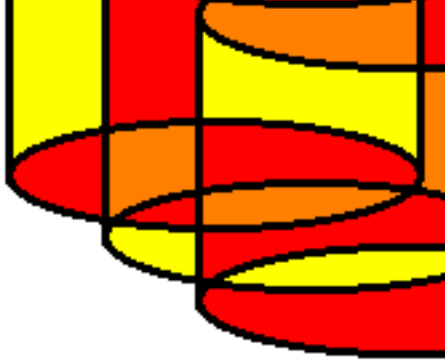
Primjer



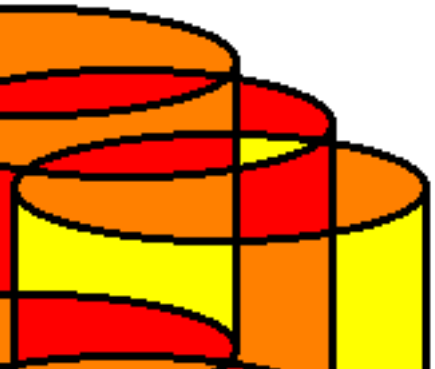
```
joza = osoba( 'js', 'Josip', 'Presvetli' )
print( joza.ime )
print( joza.adresa )
print( joza )
joza.adresa = 'Jalkovecka 24'
```



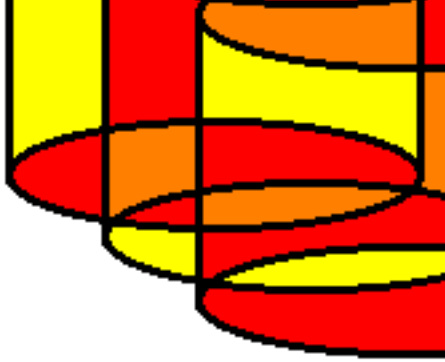
Primjer



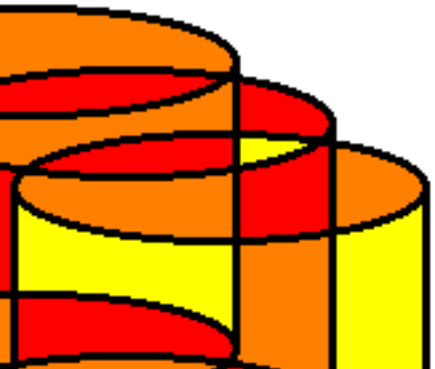
```
class student( osoba ):
     def __init__( self, matbr, ime, prezime, adresa=None ):
        osoba.__init__( self, matbr, ime, prezime, adresa )
        self.matbr = matbr
        self.kolegiji = []

     def upisiKolegij( self, kolegij ):
        self.kolegiji.append( kolegij )
```



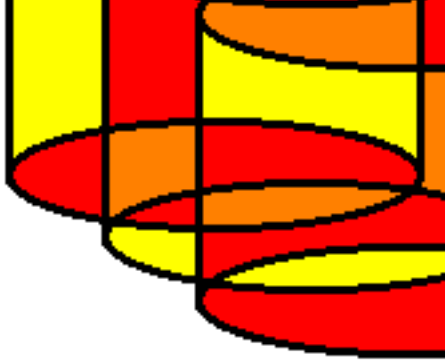
Primjer



```
bara = student( 32324, 'Barica', 'Presvetli' )
print( bara )
print( bara.sifra )
bara.upisiKolegij( 'Teorija baza podataka' )
print( bara.kolegiji )
```



Rječnici



- Sintaksa:

```
x = {}
```

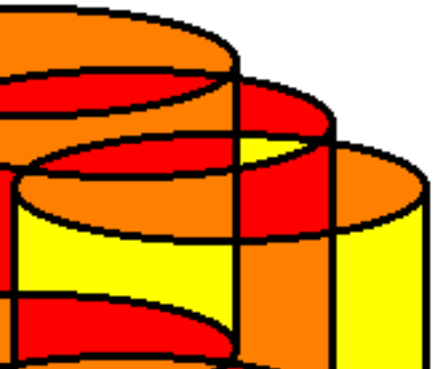
  - prazni riječnik

```
x[ 'kljuc' ] = 'vrijednost'
```

  - postavljanje vrijednosti ključa

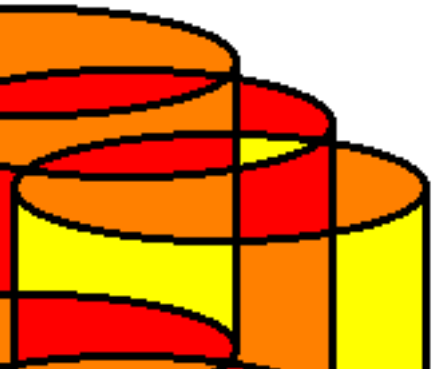
```
del x[ 'kljuc' ]
```

  - brisanje vrijednosti ključa



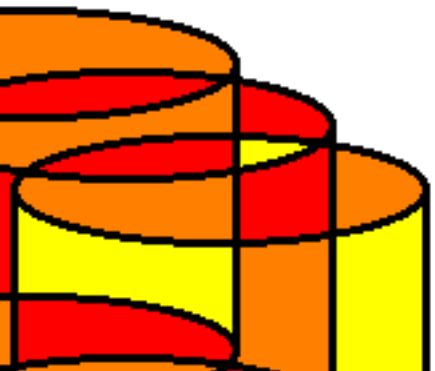
Primjer

```
x = {}
x
x[ 'kljuc' ] = 'vrijednost'
x[ 'kljuc' ]
x[ 2 ] = [ 1, 2, 3, 4 ]
x
del x[ 2 ]
x
x[ 7 ] = 333
```

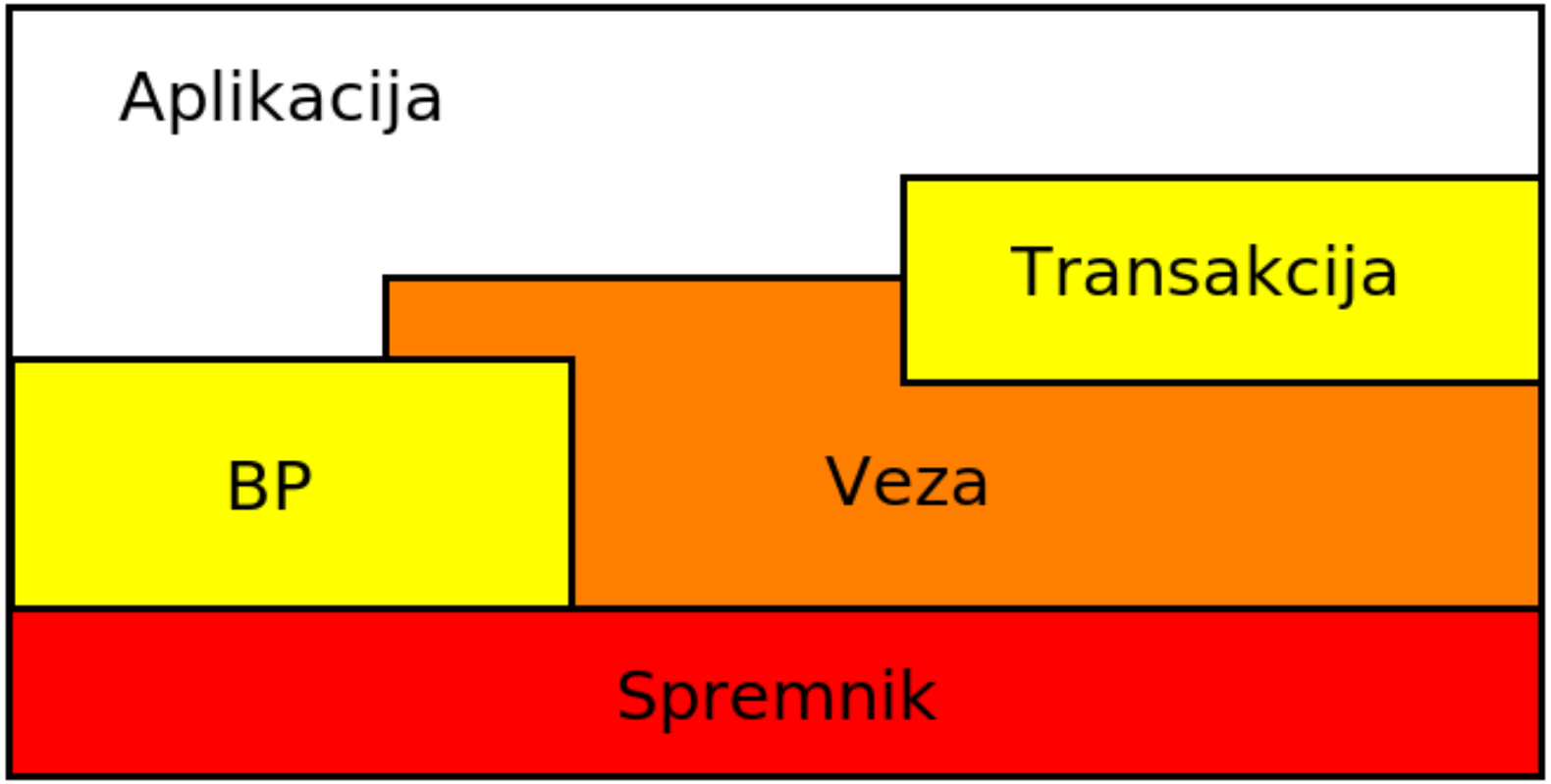


Primjer

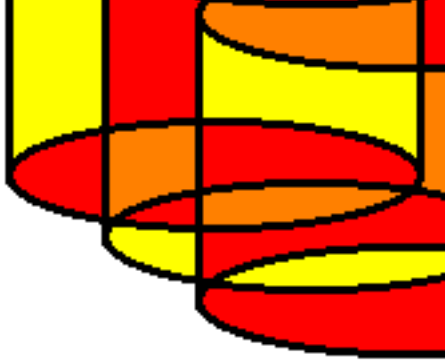
```
x.keys()
x.values()
x.items()

for k, v in x.items():
   print( k, ' ==> ', v )
```



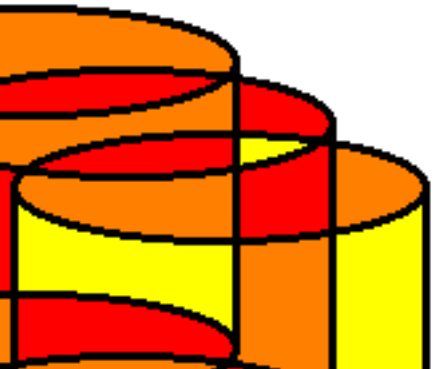
Arhitektura sustava ZODB



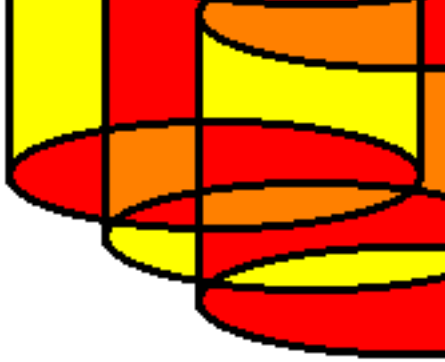
Spremnik



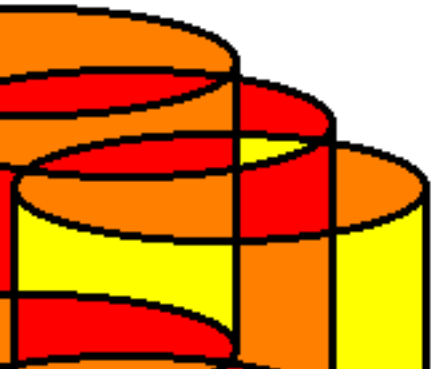
- **Storage Interface** (spremnik) – najniži sloj fizičke pohrane podataka pri čemu postoje različite mogućnosti pohrane
  - **FileStorage** – pohrana u obliku datoteke
  - **BDBFullStorage** – Berkeley DB
  - **DCOracleStorage** – relacijska baza podataka
  - **ClientStorage** – baza na mrežnom poslužitelju ...



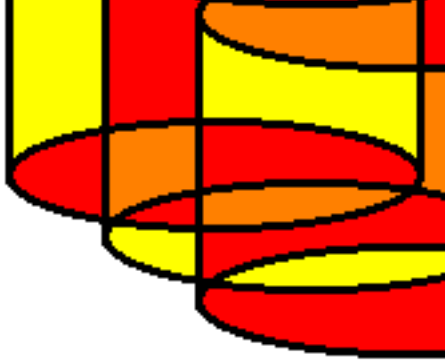
Baza podataka



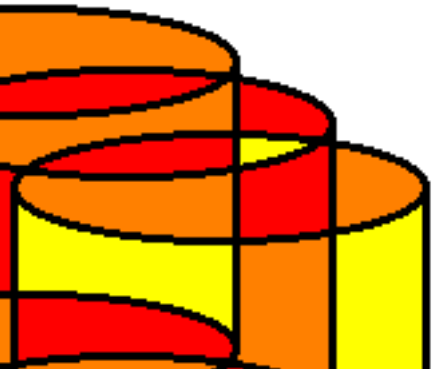
- **DB** (BP)
  - logička baza podataka koja upravlja nizom veza k fizičkoj razini



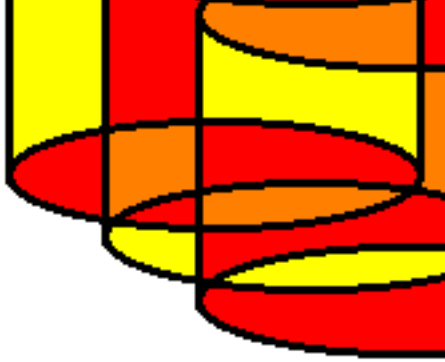
Veza k bazi podataka



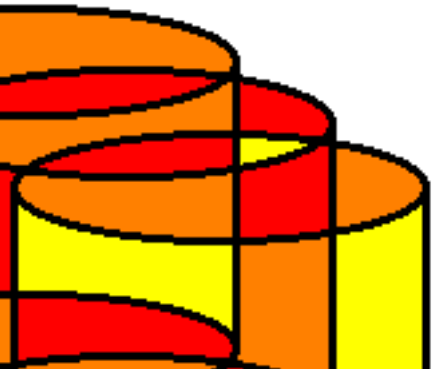
- **Connection** (veza)
  - veza k bazi podataka koja prati promjene u objektima te ih pohranjuje u fizički sloj



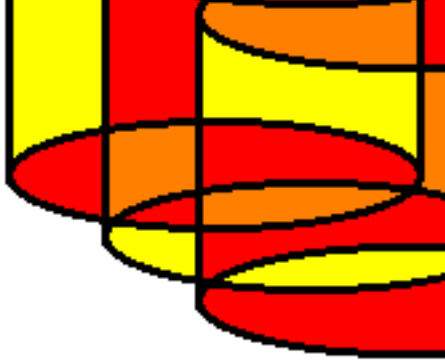
Transakcijski podsustav



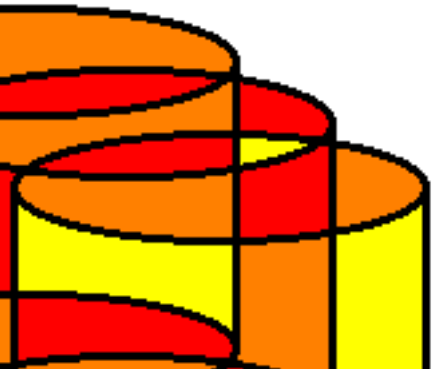
- **Transaction** (transakcija)
  - omogućava rad s transakcijama koja imaju ACID svojstva



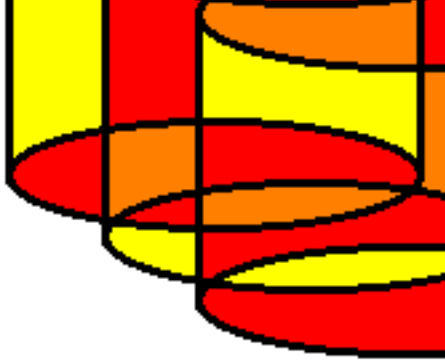
Transakcijski podsustav



- **Transaction** (transakcija)
  - omogućava rad s transakcijama koja imaju ACID svojstva
    - Atomicity
    - Consistency
    - Isolation
    - Durability
- Što znače ova svojstva?

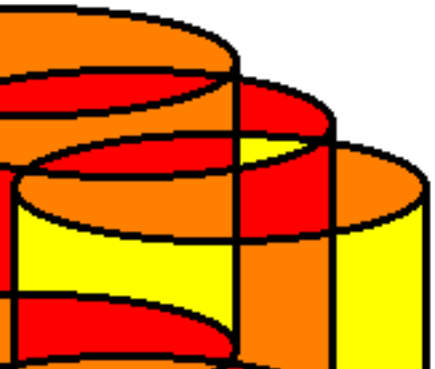


Priprema

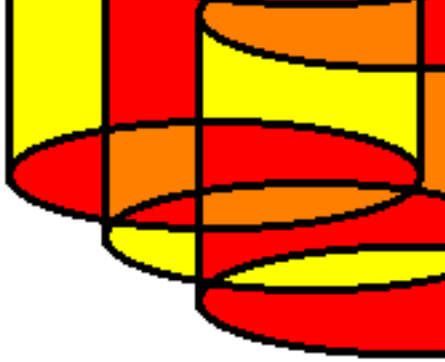


- U slučaju da import modula na sljedećem slajdu ne radi, potrebno je instalirati ZODB sa sljedećom naredbom:

```
sudo pip3 install zodb3 # za Py3
sudo pip install zodb3 # za Py2
```

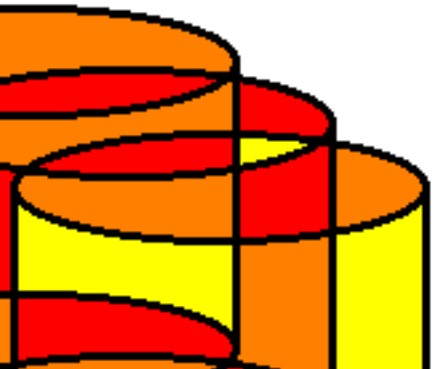


Rad sa ZODB

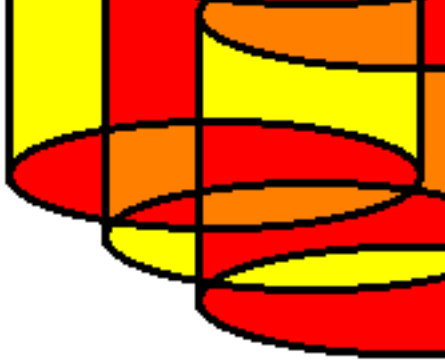


- Za rad sa ZODB sustavom potrebno je importirati određene module:

```
from ZODB import FileStorage, DB
from persistent import Persistent
import transaction
```

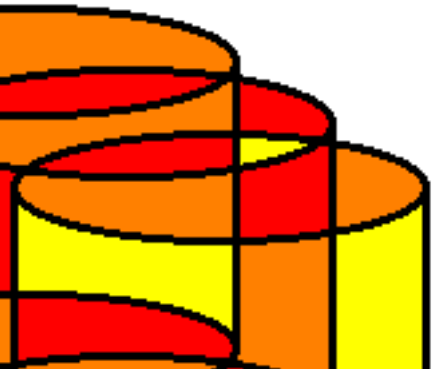


Uspostava veze

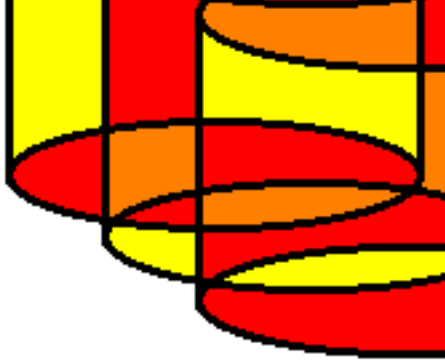


```
storage =
  FileStorage.FileStorage('/tmp/tbp.fs')
db = DB( storage )
conn = db.open()
root = conn.root()
```

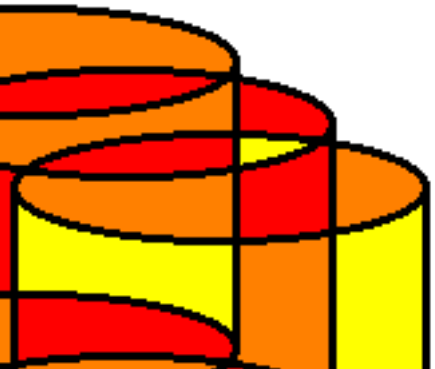
- U varijabli root se sada nalazi korijen baze podataka.
- Korjen se ponaša kao Python rječnik (dictionary)



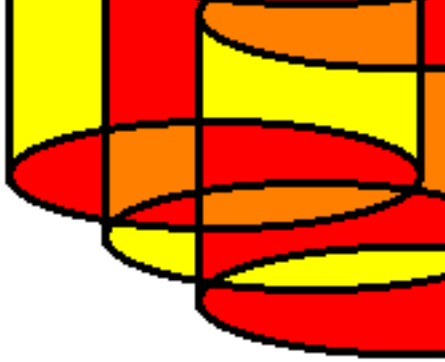
Zadatak



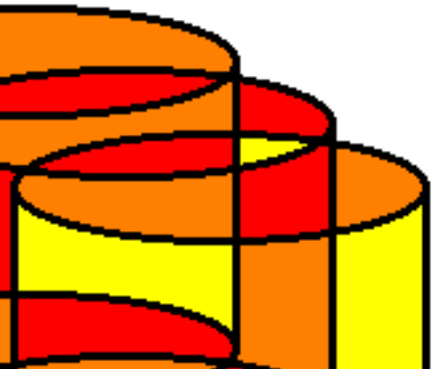
- U datoteci oobp.py implementirajte funkciju:
  - open_fs(*zdatoteka*) - uspostavlja vezu k ZODB bazi podataka pohranjenoj u datoteci *zdatoteka* te vraća objekt veze



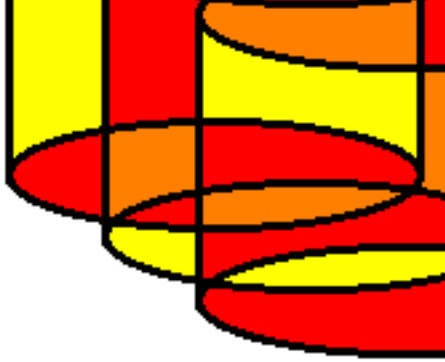
Pohrana objekata



- Da bi mogli pohranjivati neke objekte u bazi podataka potrebno je da oni budu
  - jednostavni objekti (**`int`**, **`str`**, **`float`**, ...) ili
  - instance podklase klase **`Persistent`**

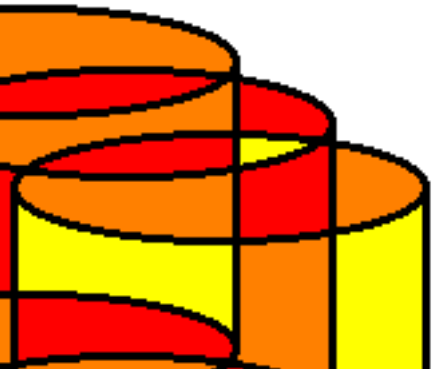


Transakcije

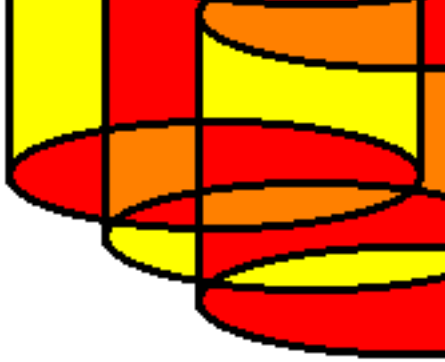


- Promjene u bazi podataka ostaju nevidljive dok se ne završi transakcija najčešće naredbom

```
transaction.commit()
```

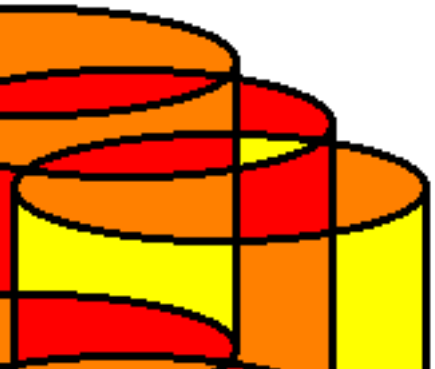


Transakcije

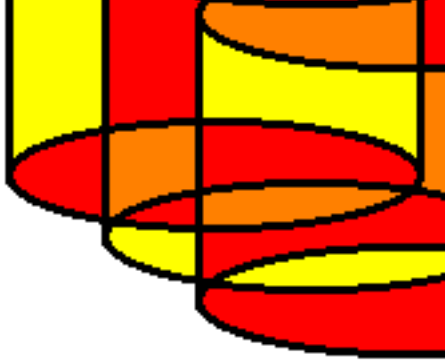


- Od transakcije (tj. pohrane rezultata) moguće je odustati naredbom:

```
transaction.abort()
```

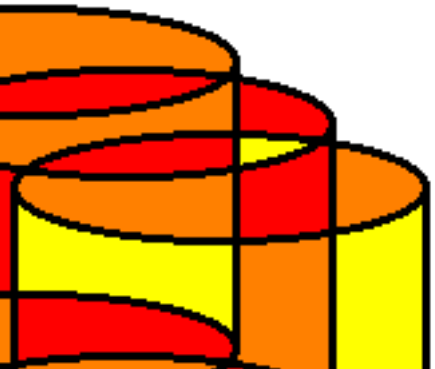


Transakcije

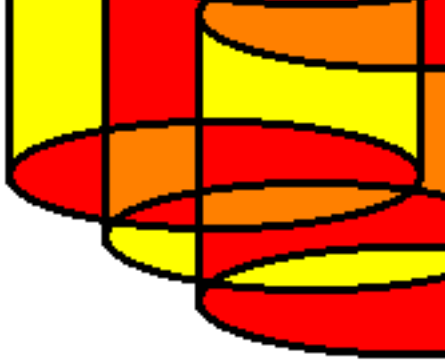


- Za pokretanje nove transakcije koristi se naredba

```
transaction.begin()
```

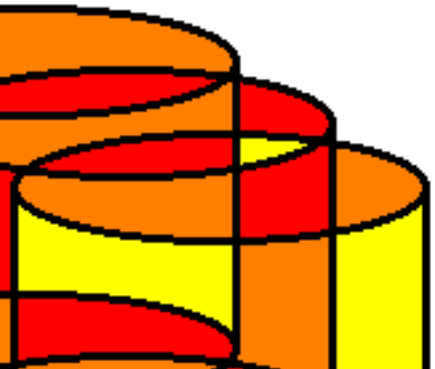


Transakcije



- Za dobivanje instance trenutne transakcije (transakcije su objekti) koristi se

```
transaction.get()
```



Primjer

```
from oobp import * # oobp.py mora biti u trenutnom direktoriju
conn = open_fs( '/tmp/tbp.fs' )
root = conn.root()
class osoba( Persistent ):
      pass

ivek = osoba()
ivek.prezime = 'Ivic'

root[ 'ii' ] = ivek
root[ 'ii' ].prezime

transaction.commit()
conn.close()
```

CTRL+D

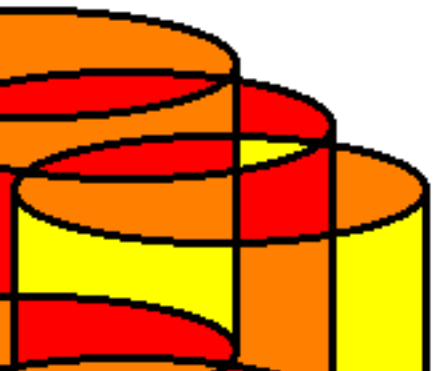
(izlazak iz Python konzole)

Primjer

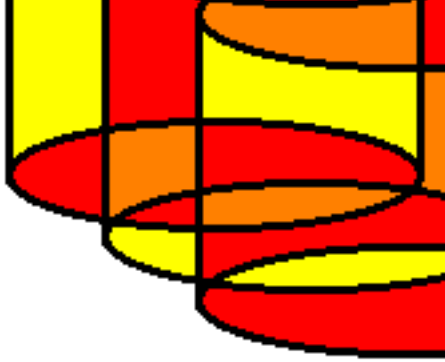
Ponovno pokrenite Python konzolu i upišite:

```
from oobp import *
conn = open_fs( '/tmp/tbp.fs' )
root = conn.root()
class osoba( Persistent ):
    pass
root[ 'ii' ].prezime
root
```

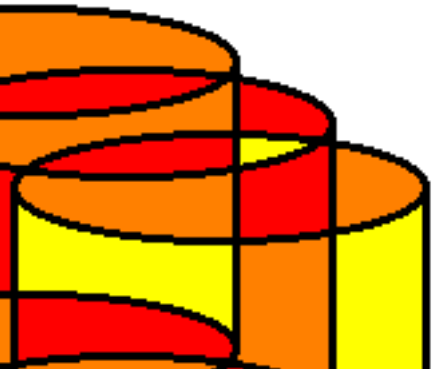
Vidimo da su podaci ostali pohranjeni!



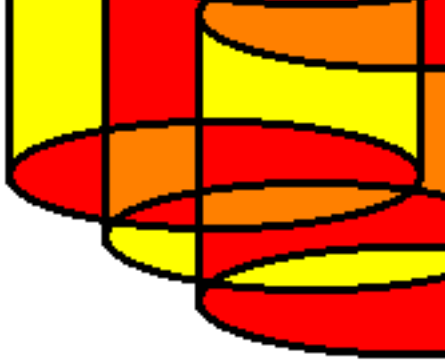
Mrežni spremnici



- Mrežni spremnik omogućava efikasno djeljenje kompleksnih objekata u mreži.

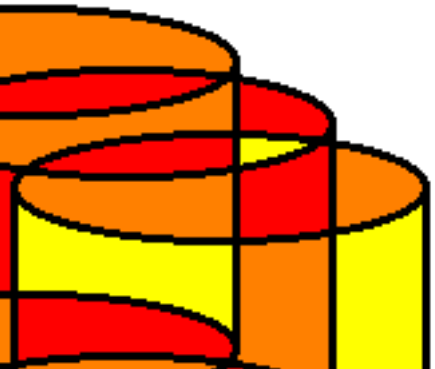


Pokretanje mrežnog spremnika

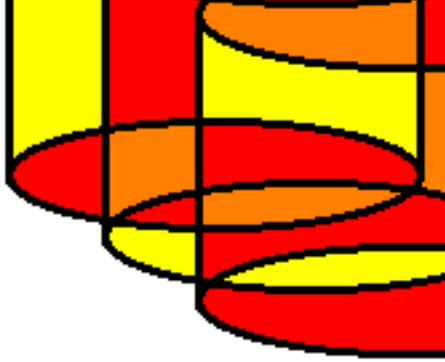


- Pokrenimo mrežni spremnik na lokalnom računalu

```
runzeo –a localhost:2709 –f /tmp/mrezni.fs
```



Spajanje na mrežni spremnik

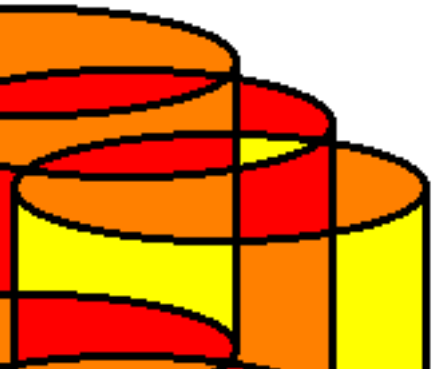


- Za razliku od datotečnog spremnika importira se modul:

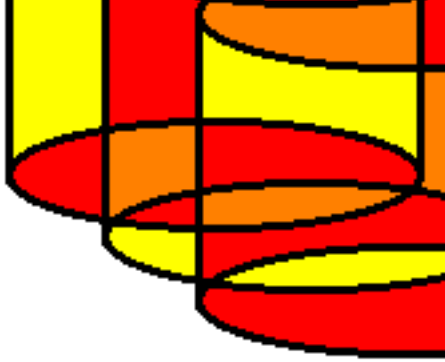
```
from ZEO.ClientStorage import
ClientStorage
```

- Ostali moduli ostaju jednaki. Spremnik se instancira na sljedeći način:

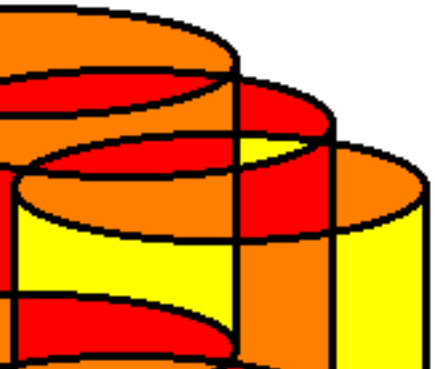
```
st = ClienStorage( ( host, port ) )
```



Zadatak

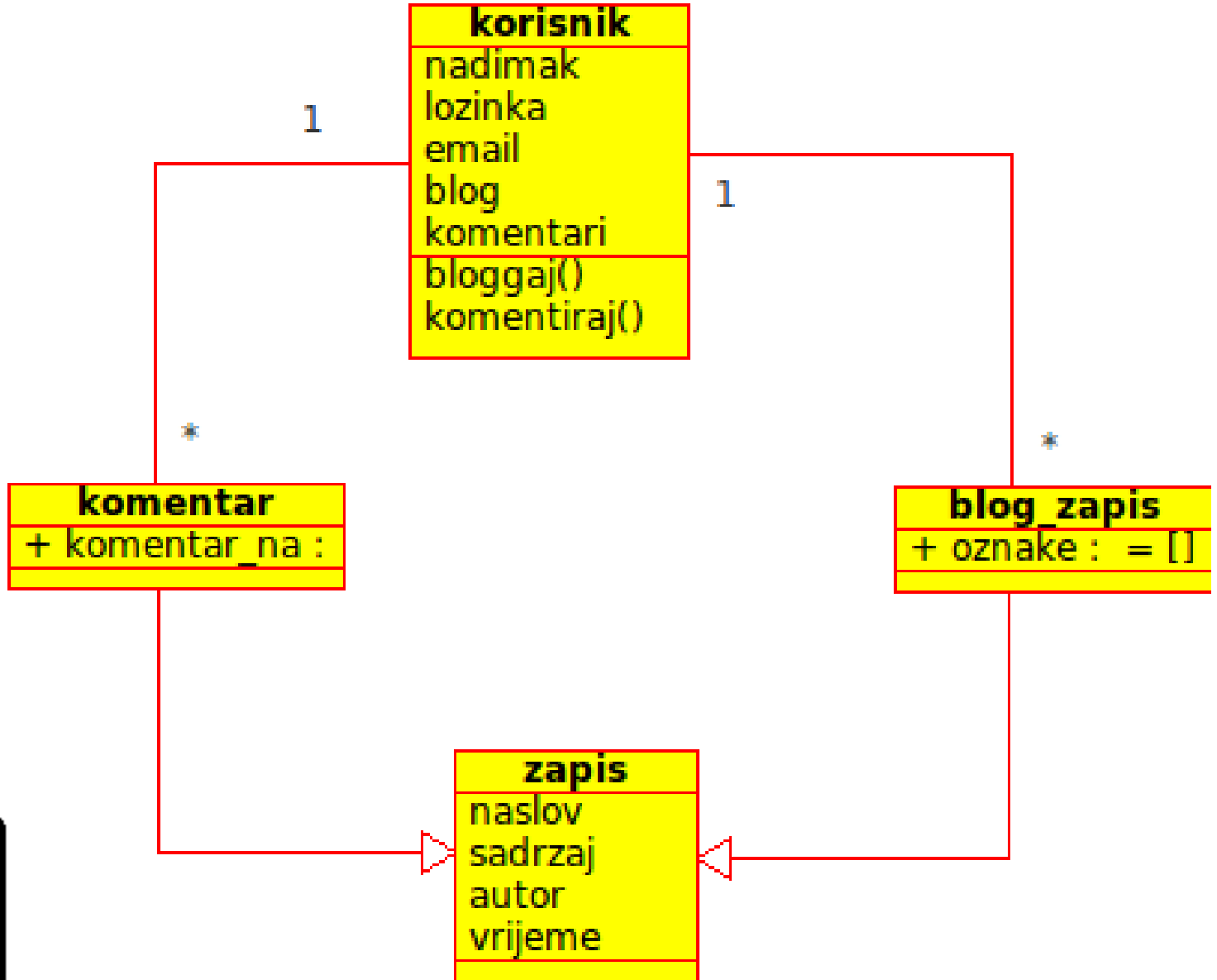


- U datoteci oobp.py implementirajte funkciju:
  - open_cs(zhost, zport) - uspostavlja vezu k ZODB bazi podataka na mrežnom spremniku *zhost:zport* te vraća objekt veze (connection)



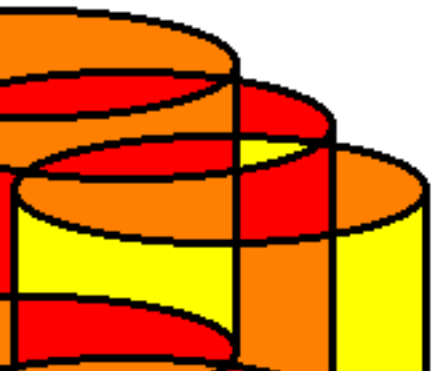
Primjer

- Kreirat ćemo bazu podataka za Weblog

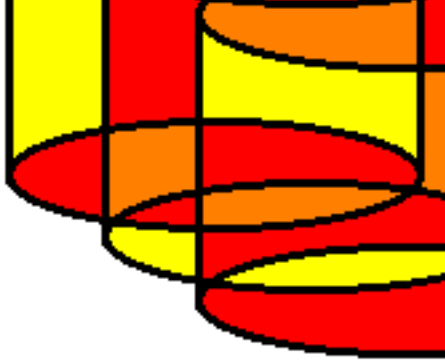


Upute

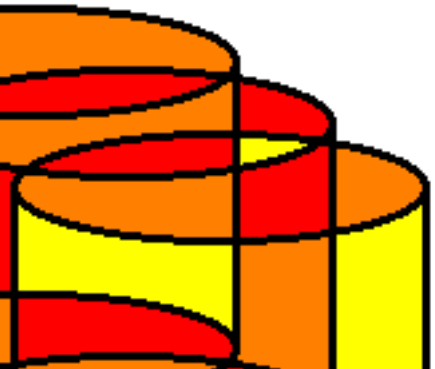
- Implementaciju koja slijedi nadodajte na postojeći sadržaj u datoteci oobp.py



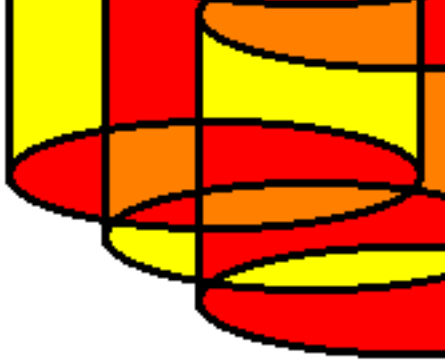
Import modula



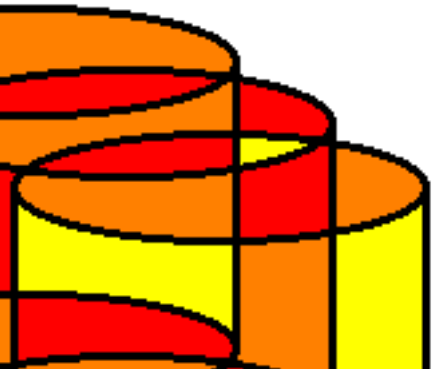
```
from persistent.list import PersistentList
from time import asctime
```



Definicija klasa

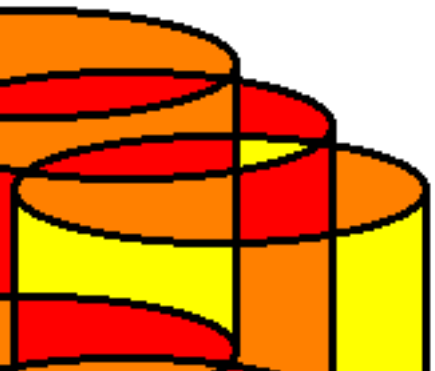


```
class zapis( Persistent ):
  def __init__(self, naslov, sadrzaj, autor):
    self.naslov = naslov
    self.sadrzaj = sadrzaj
    self.autor = autor
    self.vrijeme = asctime()
```



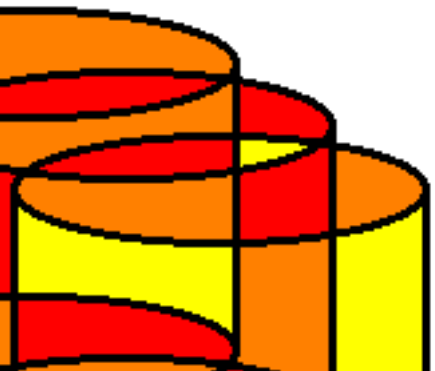
Definicija klasa

```
class komentar( zapis ):
 def __init__(self,naslov,sadrzaj,autor,komentar_na):
    zapis.__init__( self, naslov, sadrzaj, autor )
    self.komentar_na = komentar_na
```



Definicija klasa

```
class blog_zapis( zapis ):
  def __init__(self,naslov,sadrzaj,autor,oznake=[]):
    zapis.__init__( self, naslov, sadrzaj, autor )
    self.oznake = oznake
```



Definicija klasa

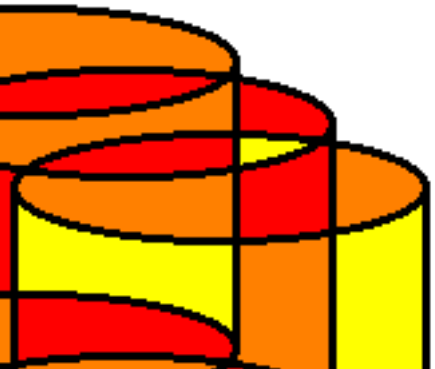
```
class korisnik( Persistent ):
  def __init__( self, nadimak, lozinka, email ):
    self.nadimak = nadimak
    self.lozinka = lozinka
    self.email = email
    self.blog = PersistentList()
    self.komentari = PersistentList()
  def bloggaj( self, naslov, sadrzaj, oznake ):
    zapis = blog_zapis( naslov, sadrzaj, self, oznake )
    self.blog.append( zapis )
    return zapis
  def komentiraj( self, zapis, sadrzaj ):
    zapis = komentar( zapis.naslov, sadrzaj, self, zapis )
    self.komentari.append( zapis )
    return zapis
```

Unos podataka

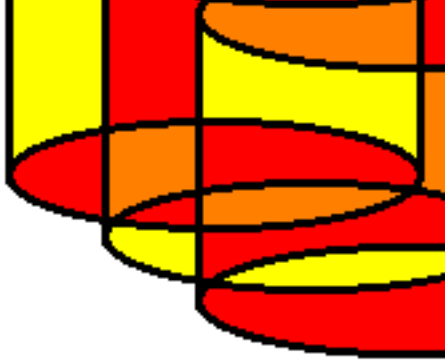
```
transaction.begin()
c = open_fs( '/tmp/tbp2.fs' )
root = c.root()
root[ 'korisnici' ] = PersistentList()
ivek = korisnik( 'ivek', 'tajna', 'ivek@foi.hr' )
bara = korisnik( 'barica', 'velika tajna', 'bara@foi.hr' )
joza = korisnik( 'joza', 'jos veca', 'joza@foi.hr' )
root[ 'korisnici' ].extend( [ ivek, joza, bara ] )
z1 = ivek.bloggaj( 'Setnja na Dravi', 'Danas sam setao Dravom. Bilo je
super!', [ 'drava', 'setnja' ] )
z2 = bara.komentiraj(z1,'Na kojoj strani Drave si setao?' )
z3 = ivek.komentiraj(z2, 'Na sjevernoj, ne volim juznu.' )
z4 = bara.bloggaj( 'Voznja avionom', 'Bas mi se dopada voznja
avionom', [ 'avion', 'letenje', 'voznja' ] )
root[ 'zapisi' ] = PersistentList()
root[ 'zapisi' ].extend( [ z1, z2, z3, z4 ] )
transaction.commit()
```

Definicija funkcija

```
def kub( x ):
   return x * x * x
def pomnozi( x, y ):
   return x * y
def neparan( x ):
   return x % 2
```



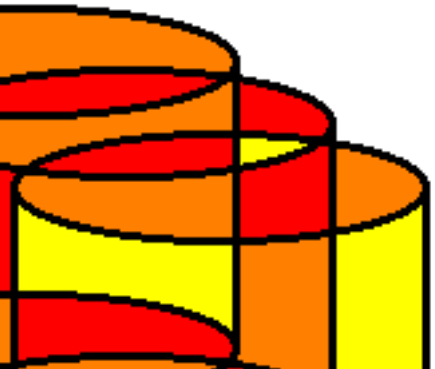
Generatori listi



Generatori listi mogu poslužiti za efikasno filtriranje i preoblikovanje podataka u Pythonu.

Njihov opći oblik je sljedeći:

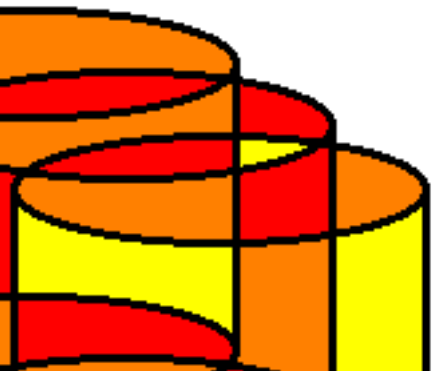
```
[ predlozak for varijabla in lista if uvjet ]
```



Upute

- Upite u nastavku isprobajte na Python konzoli u kojoj ste prethodno učitali sve iz datoteke oobp.py, tj.:

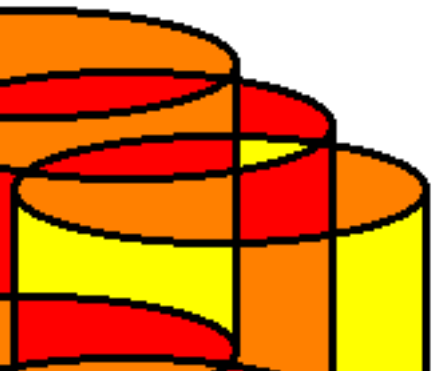
```
from oobp import *
```



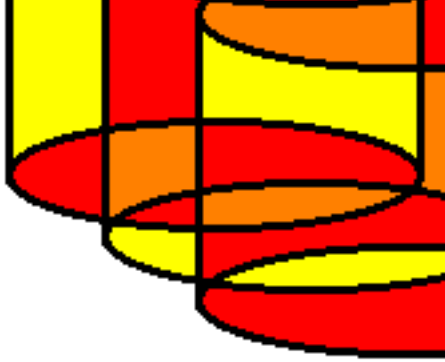
Primjeri upita

- Svi zapisi koje napisao korisnik s nadimkom 'ivek'

```
[ z for z in root[ 'zapisi' ]
        if z.autor.nadimak == 'ivek' ]
```

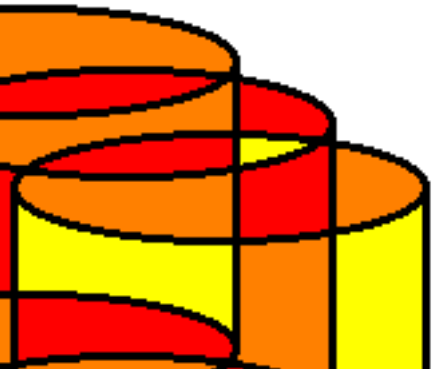


Primjeri upita

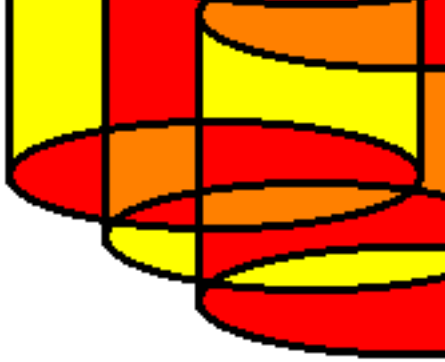


- Vremena svih zapisa koje napisao korisnik s nadimkom 'ivek'

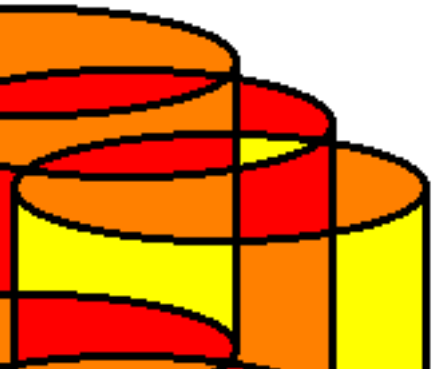
```
[ z.vrijeme for z in root[ 'zapisi' ]
        if z.autor.nadimak == 'ivek' ]
```



Funkcije višeg reda

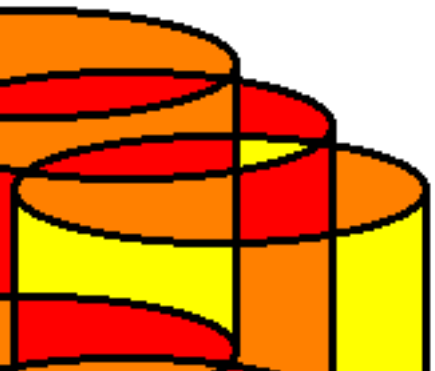


- Funkcije višeg reda su funkcije koje kao argument primaju funkciju
- Najpoznatije su **map** (mapira funkciju na elemente liste) i **reduce** (reducira listu korištenjem funkcije)



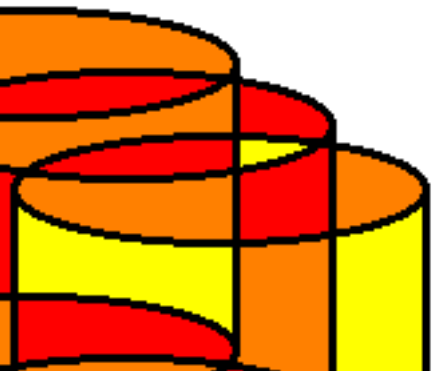
Funkcija map

```
map( kub, [ 0, 1, 2, 3, 4 ] )
```

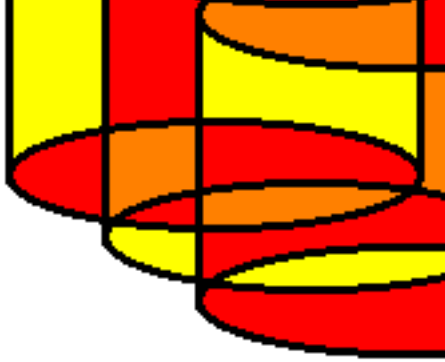


Funkcija range

```
map( kub, range( 5 ) )
```

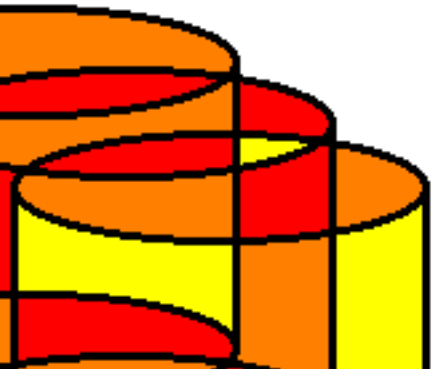


Lambda notacija



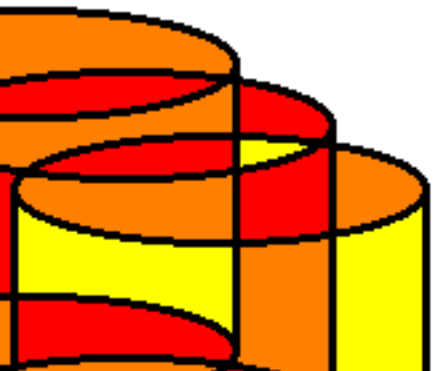
Lambda funkcije su anonimne funkcije koje se kreiraju prilikom izvođenja programa.

```
map( lambda x: x*x*x, range( 5 ) )
```



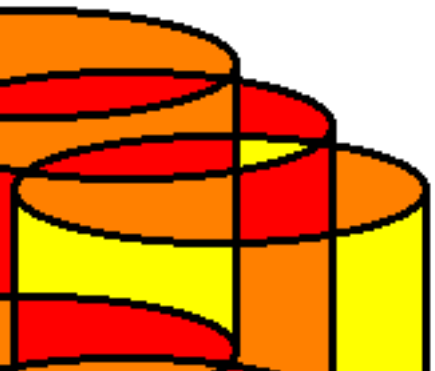
Primjer upita

```
map( lambda x: x.upper(),
     [k.nadimak for k in
                root['korisnici']])
```



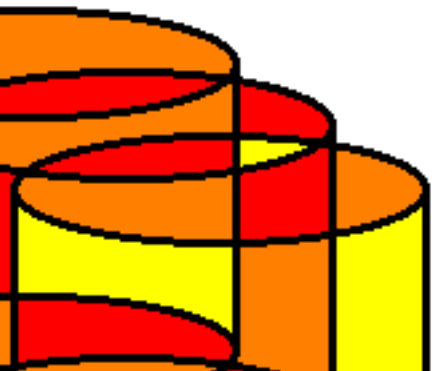
Primjer upita

```
map(lambda (x,y):x.title()+' '+y.upper(),
    [ ( k.nadimak, k.email )
            for k in root[ 'korisnici' ] ] )
```



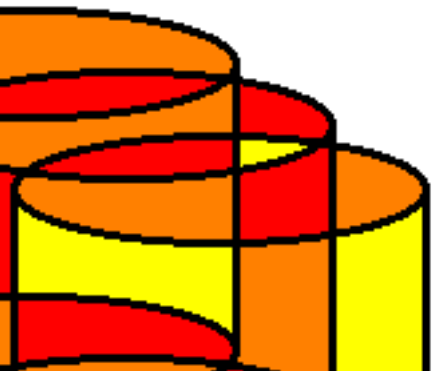
Funkcija reduce

```
reduce( pomnozi, [ 1, 2, 3 ] )
```

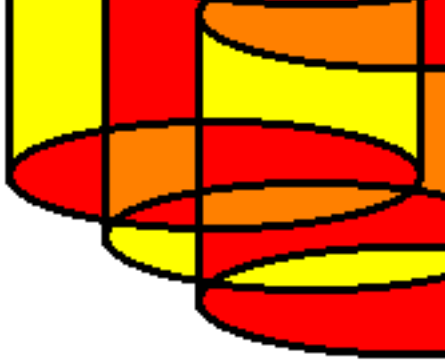


Funkcija reduce

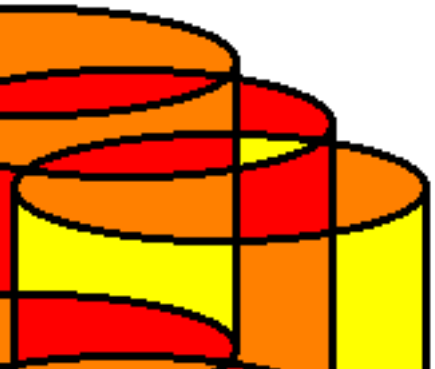
```
reduce( pomnozi, range( 1, 4 ) )
```



Funkcija reduce

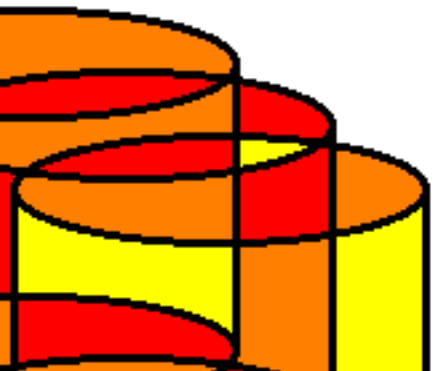


```
reduce(lambda x, y: x * y, range( 1, 4 ))
```



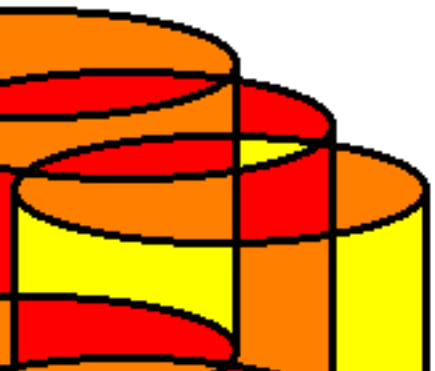
Funkcija reduce

```
reduce( lambda x, y: x + ' ' + y,
                root['zapisi'][3].oznake )
```



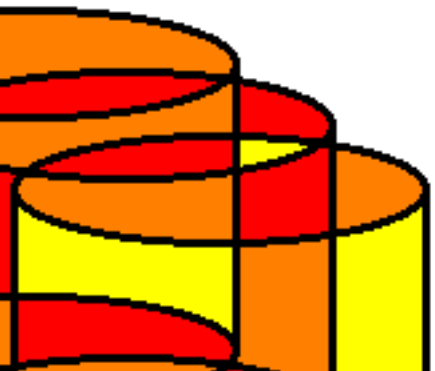
Funkcija reduce

```
reduce( lambda x, y: x + ', ' + y,
   [ z.naslov for z in root[ 'zapisi' ] ] )
```



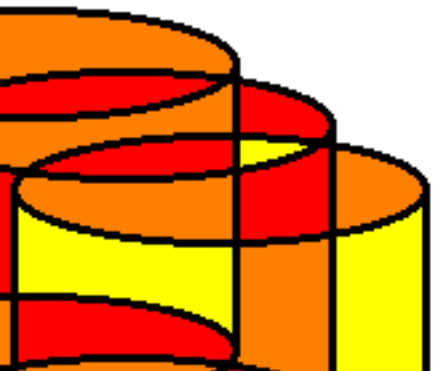
Funkcija enumerate

```
[ i for i in enumerate( ['a','b','c'] ) ]
```



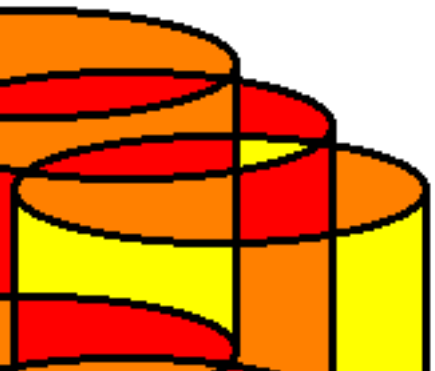
Funkcija enumerate

```
[ i for i in enumerate( [ k.email for k in
        root['korisnici'] ], start=100 ) ]
```



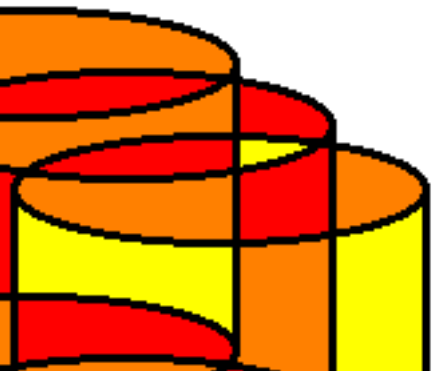
Funkcija reversed

```
[ i for i in reversed( [ 0, 1, 2, 3 ] ) ]
```



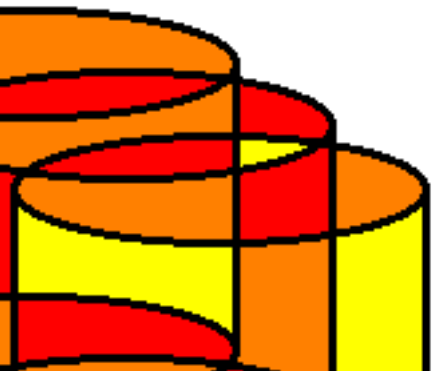
Funkcija zip

```
zip( [ 1, 2, 3 ], [ 'a', 'b', 'c' ] )
```



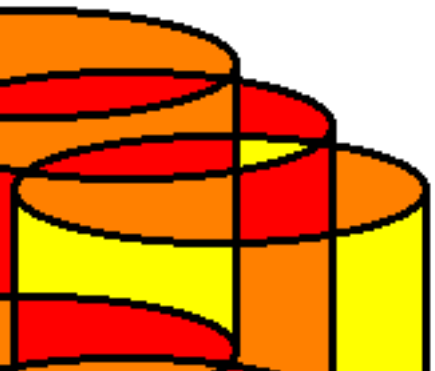
Funkcija dict

```
dict( zip( [ 1, 2, 3 ], ['a','b','c'] ) )
```



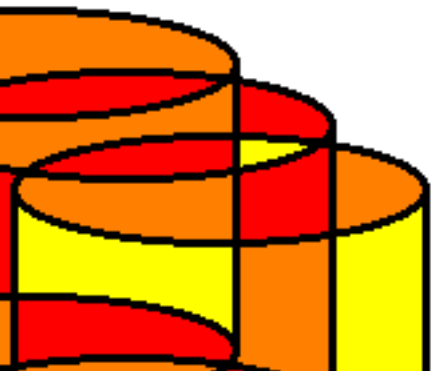
Funkcija dict

```
dict( zip(
   [k.nadimak for k in root['korisnici']],
   [k.email for k in root['korisnici']]) )
```



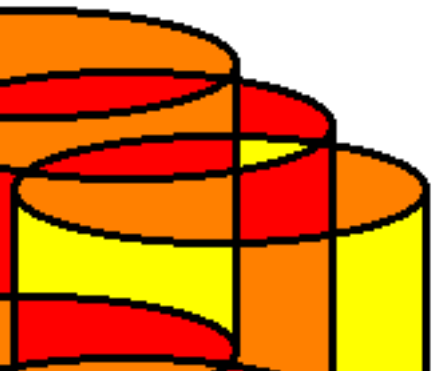
Funkcija dict

```
dict( [ ( k.nadimak, k.email ) for k in
                    root[ 'korisnici' ] ] )
```



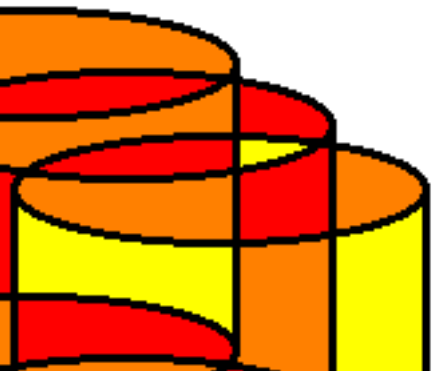
Funkcija filter

```
filter( neparan, range( 10 ) )
```



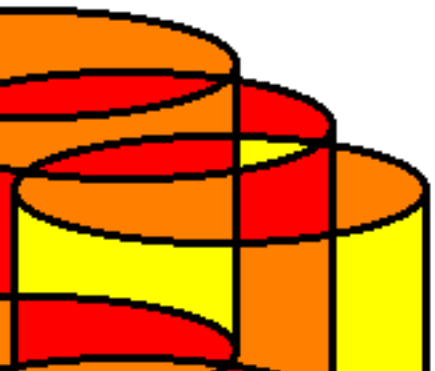
Funkcija filter

```
filter( lambda x: x % 2, range( 10 ) )
```

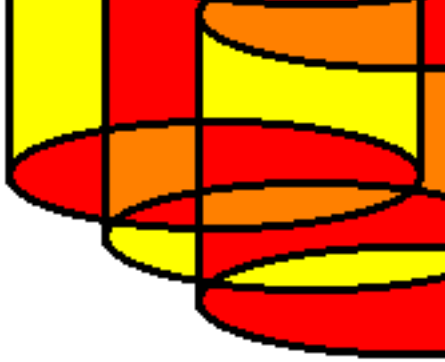


Funkcija filter

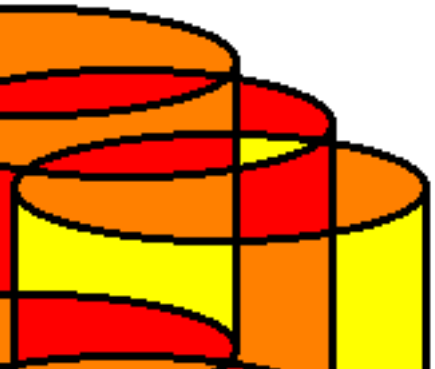
```
filter( lambda x:
 x.autor.nadimak == 'ivek',root['zapisi'])
```



Agregirajuće funkcije

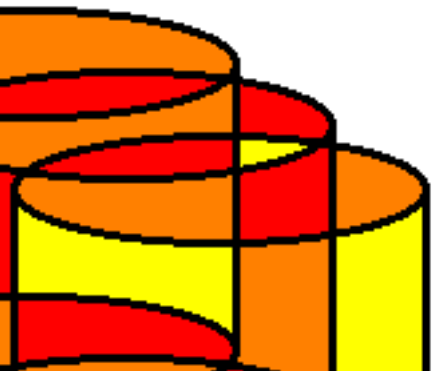


```
max( [ 4, 2, 1, 4, 5, 8, 2 ] )
min( [ 4, 2, 1, 4, 5, 8, 2 ] )
sum( [ 4, 2, 1, 4, 5, 8, 2 ] )
```



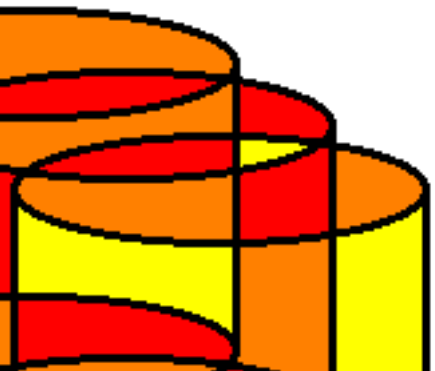
Još o listama

```
range(1,11)
```



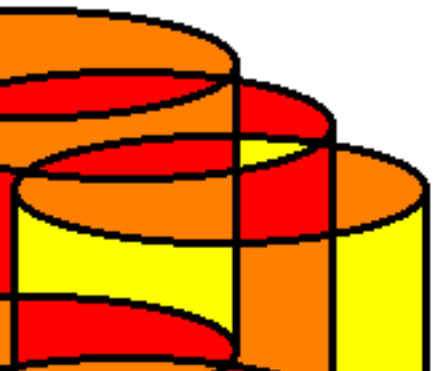
Još o listama

```
range(1,11)
range(1,11)[ 2 ]
```



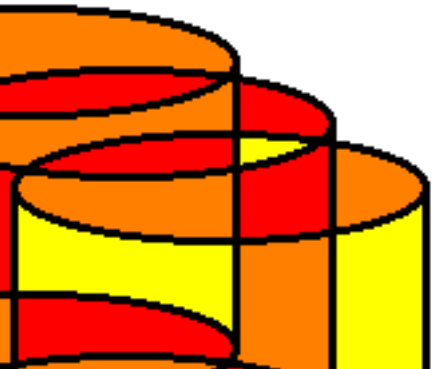
Još o listama

```
range(1,11)
range(1,11)[ 2 ]
range(1,11)[ -1 ]
```



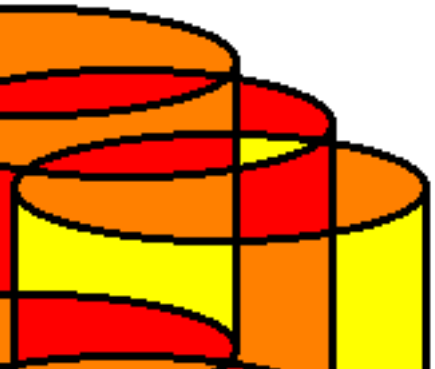
Još o listama

```
range(1,11)
range(1,11)[ 2 ]
range(1,11)[ -1 ]
range(1,11)[ 2:4 ]
```



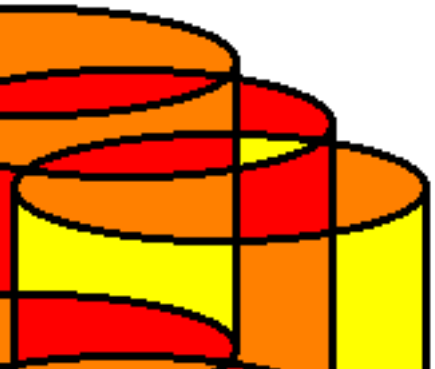
Još o listama

```
range(1,11)
range(1,11)[ 2 ]
range(1,11)[ -1 ]
range(1,11)[ 2:4 ]
range(1,11)[ :4 ]
```



Još o listama

```
range(1,11)
range(1,11)[ 2 ]
range(1,11)[ -1 ]
range(1,11)[ 2:4 ]
range(1,11)[ :4 ]
range(1,11)[ 2: ]
```

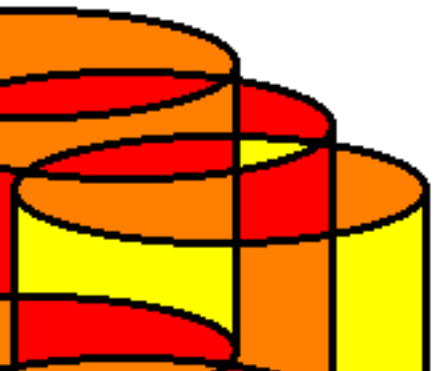


Još o listama

```
range(1,11)
range(1,11)[ 2 ]
range(1,11)[ -1 ]
range(1,11)[ 2:4 ]
range(1,11)[ :4 ]
range(1,11)[ 2: ]
range(1,11)[ :-3 ]
```

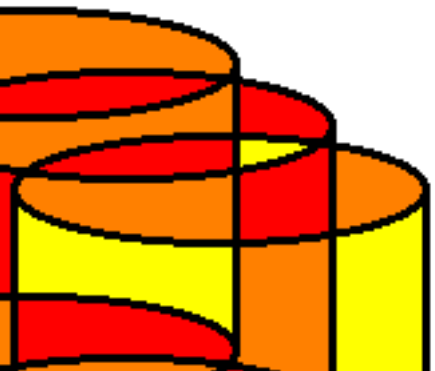
Funkcija sorted

```
sorted( [ 2, 5, 1, 4, 3 ] )
```

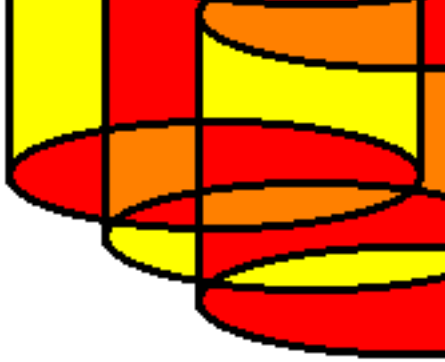


Funkcija sorted

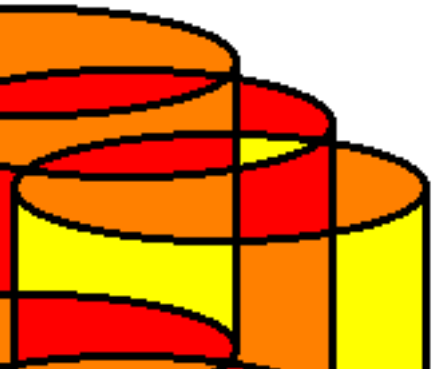
```
sorted( [ k.nadimak for k in
                    root[ 'korisnici' ] ] )
```



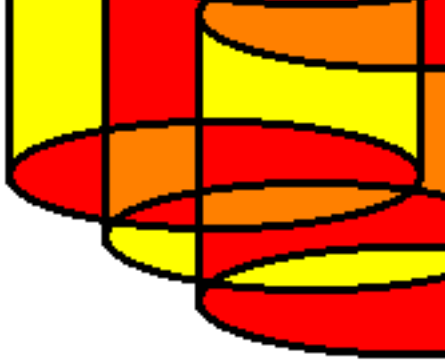
Funkcija sorted



```
sorted( [ k for k in
            root[ 'korisnici' ] ],
                key=lambda x:x.nadimak  )
```



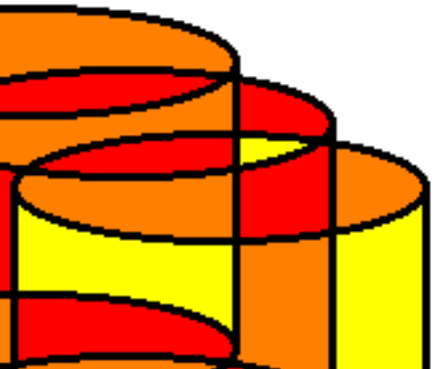
Zadatak



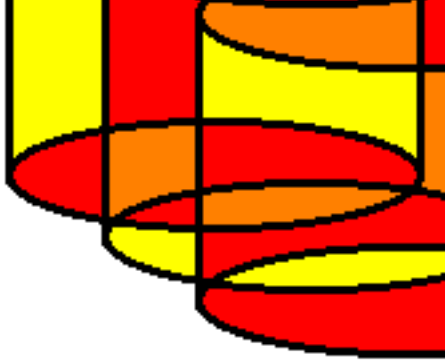
- Implementirajte perzistentnu klasu Trokut koja ima atribute a, b i c koje reprezentiraju stranice trokuta te metodu ispis koja ispisuje podatke o trokutu na sljedeći način:

  a = 3 b = 4 c = 5

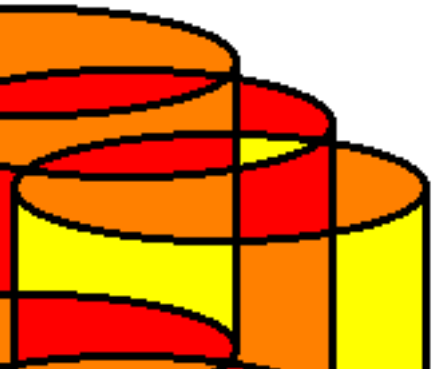
- Instancirajte 10 trokuta te ih pohranite u ZODB bazu podataka u perzistentnu listu root[‘trokuti’]
- Implementirajte upit (generator liste) koji će iz liste svih trokuta izvući one koji ne čine trokut (zbroj dviju kraćih stranica mora biti veći najdulje stranice)



Organizacija podataka



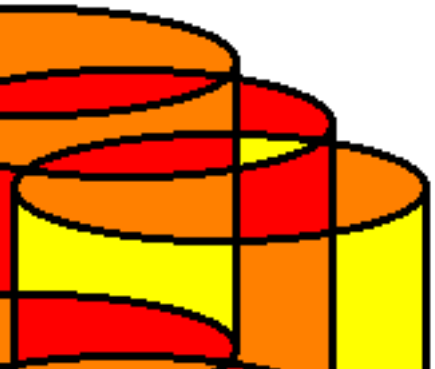
- U ZODB bazi podataka nema striktnih pravila o organizaciji podataka već je programeru dopušteno da organizira podatke po volji.



Organizacija podataka

- Moguće je u korijenu baze podataka pohraniti sve objekte bez obzira na tip:

root	ključ	objekt	tip
	k1	o1	osoba
	k2	o2	osoba
	k3	o3	integer
	k4	o4	lista
	k5	o5	grupa
	k6	o6	osoba

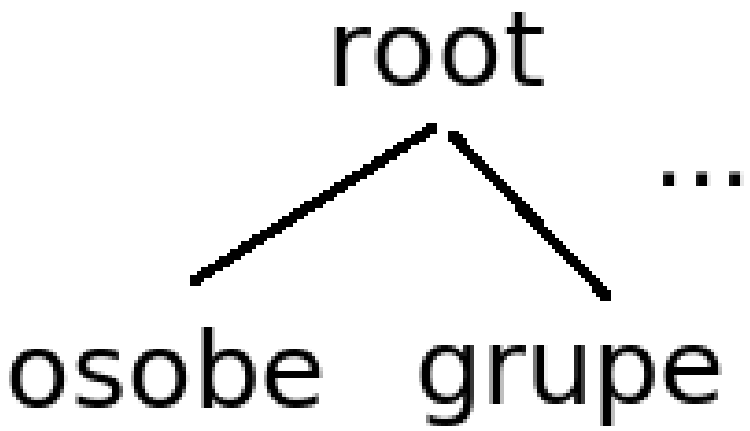


Organizacija podataka

- Efikasnije je organizirati podatke hijerarhijski

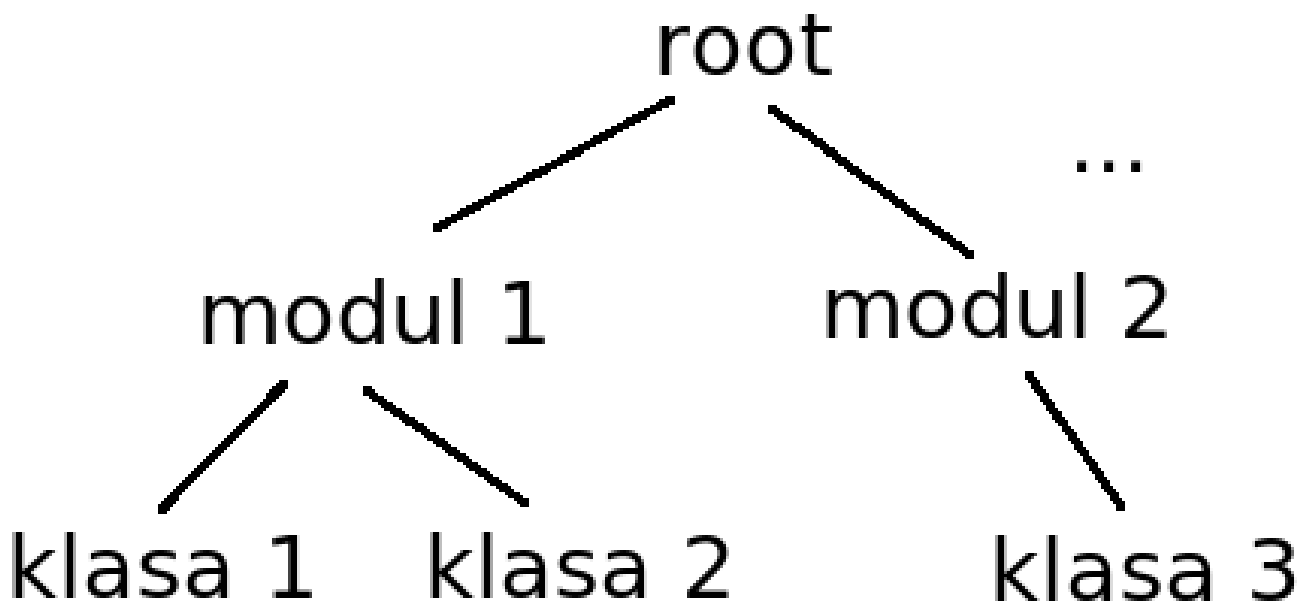
root	ključ	objekt	tip			
	ko	ro	rječnik			
				k1	o1	osoba
				k2	o2	osoba
				k6	o6	osoba
	ki	ri	rječnik			
				k3	o3	integer
	kg	rg	rječnik			
				k5	o5	grupa
	kl	rl	rječnik			
				k4	o4	lista

Organizacija podataka



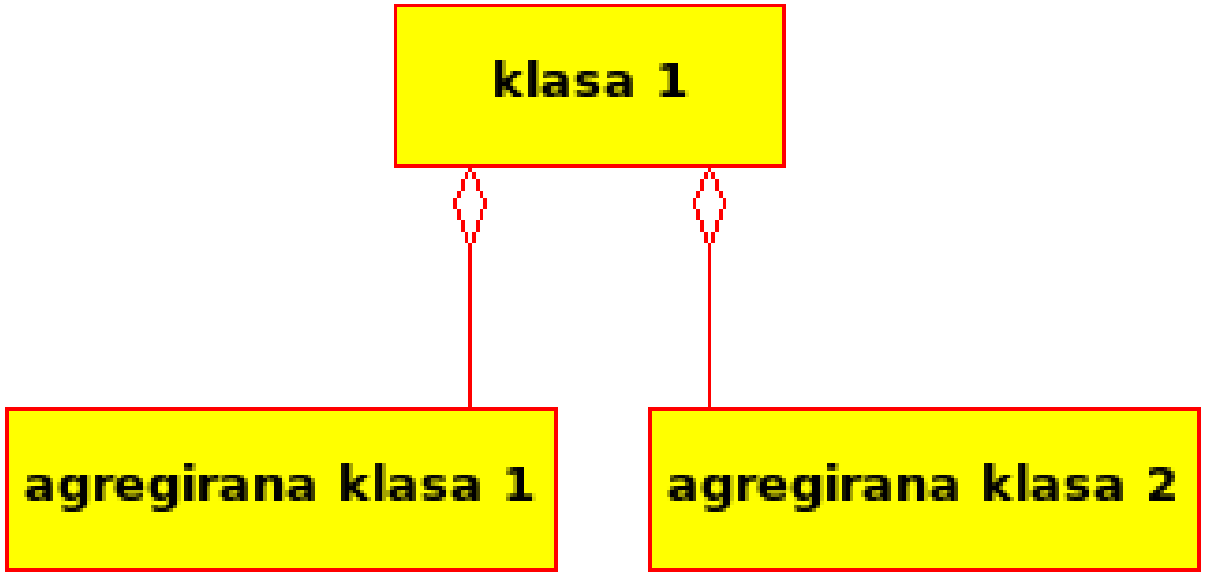
Organizacija podataka

- po modulima aplikacije

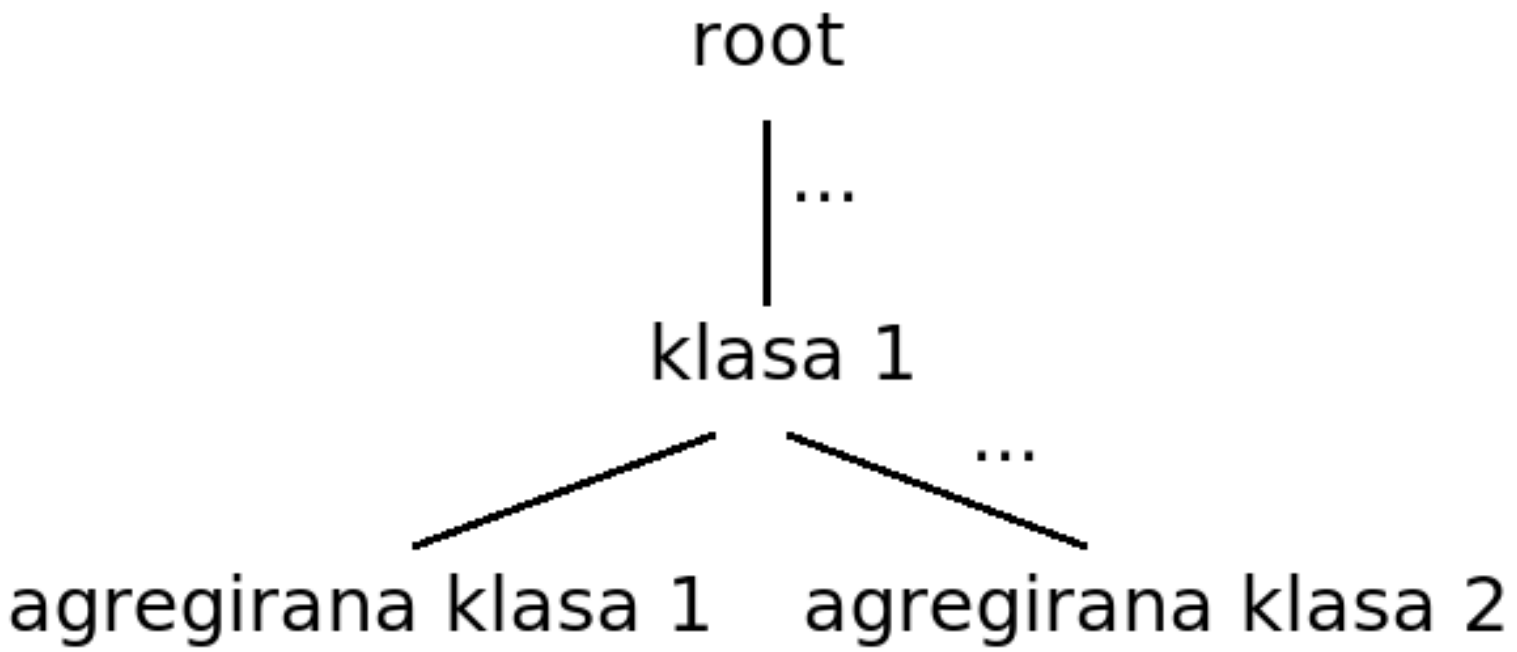


Organizacija podataka

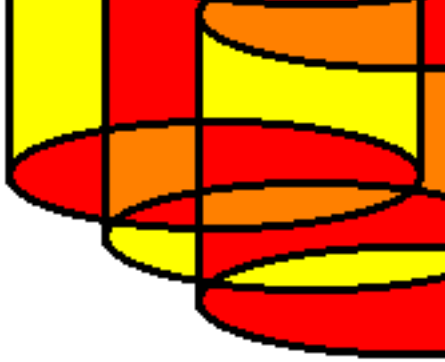
- u ovisnosti o konceptualnoj povezanosti (npr. agregacija)



Organizacija podataka

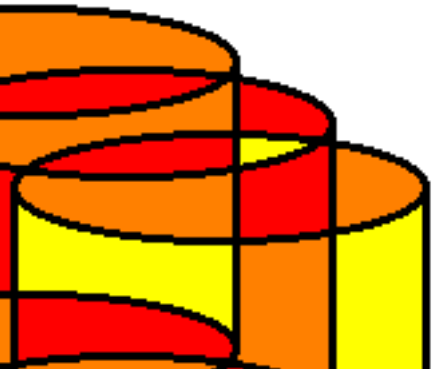


Priprema

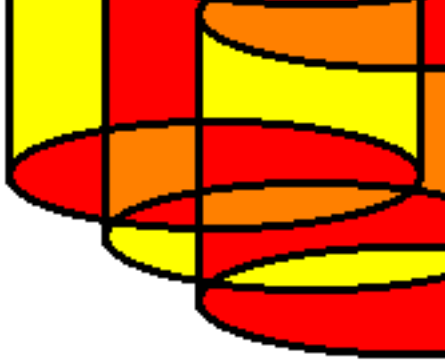


- U nastavku slijedi implementacija aplikacije s grafičkim sučeljem
- U slučaju da se javi pogreška o ne postojanju modula, moguće je da je potrebno instalirati TkInter, naredbnom:

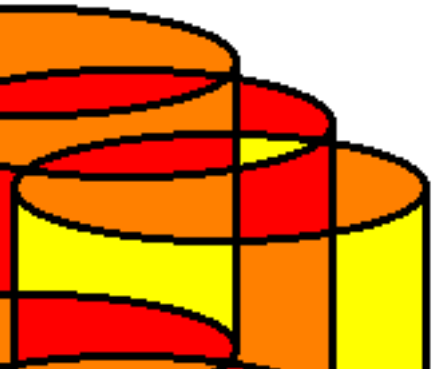
```
sudo apt install python3-tk # koristiti Py3
```



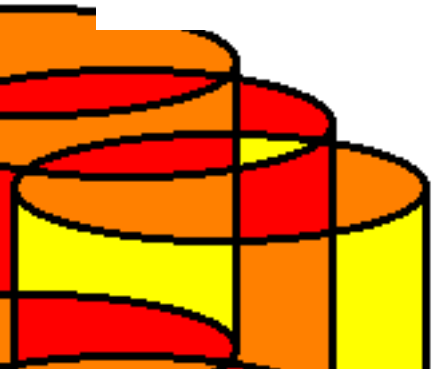
Primjer

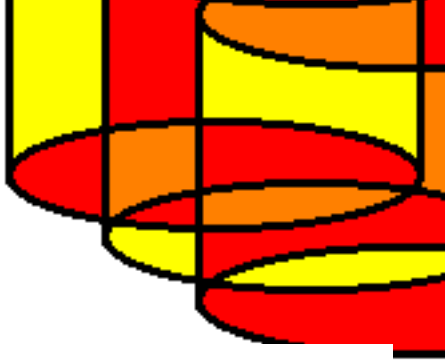


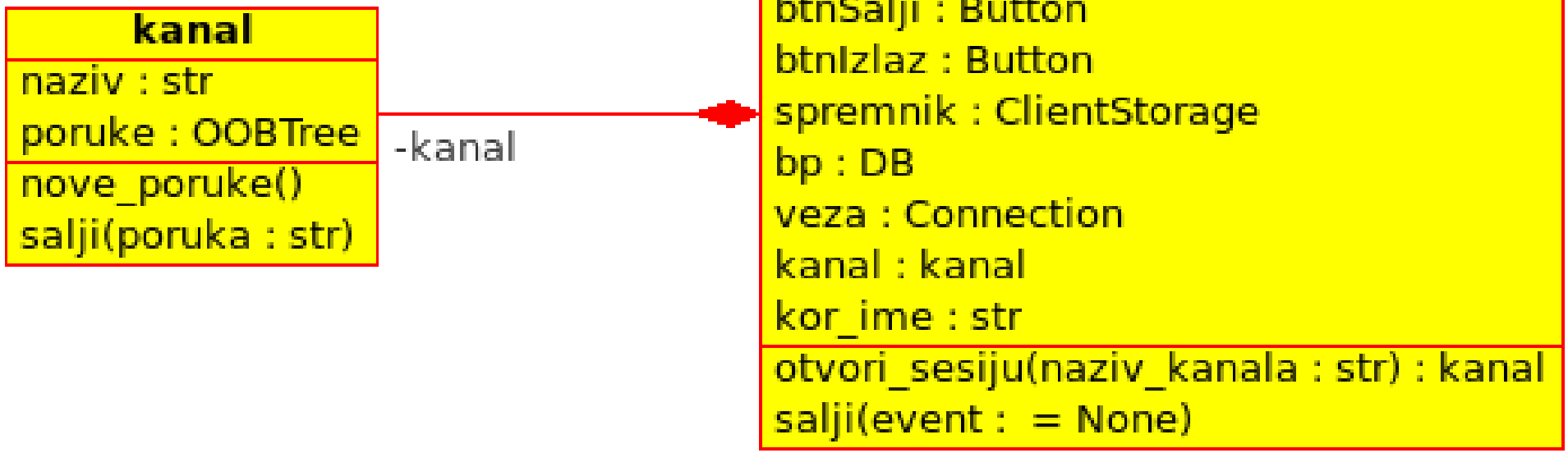
- Rad s mrežnim spremnikom upoznat ćemo na primjeru jednostavne aplikacije za brbljaonicu.
- Na sličan način moguće je ostvariti bilo kakvu razmjenu kompleksnih objekata putem mreže (npr. mrežni šah ili bilo kakva mrežna igra, mrežna oglasna ploča, mrežni repozitorij za praćenje poslovnih procesa, DMS ...)



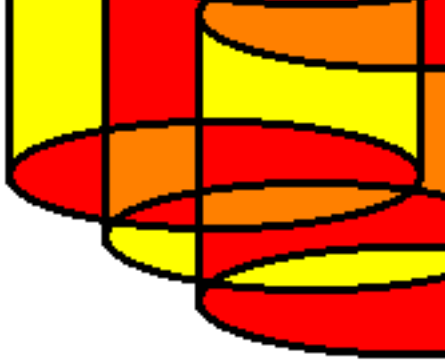
- Implementirat ćemo sljedeći UML dijagram klasa



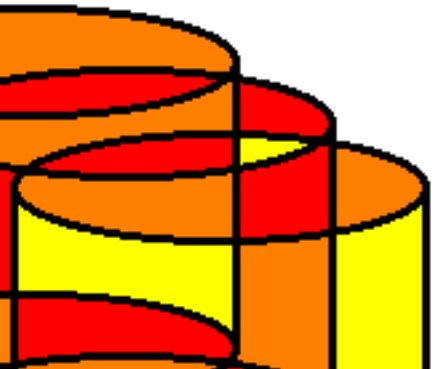




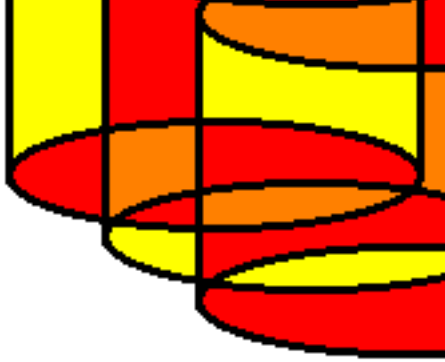
brblj.py



https://tinyurl.com/zodb-brblj



Pokretanje



Za pokretanje potrebno je pokrenuti ZEO poslužitelj, npr.

```
runzeo -a localhost:2709 -f /tmp/brblj.fs
```

Zatim podesiti prava:

```
chmod +x brblj.py
```

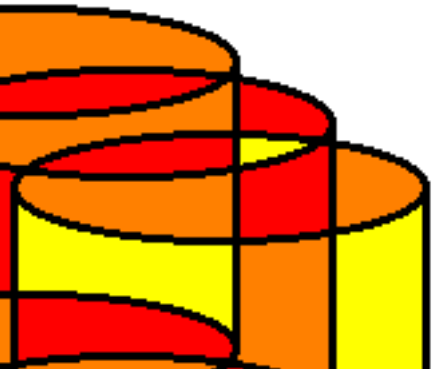
Opcionalno pogledati pomoć:

```
./brblj.py --help
```

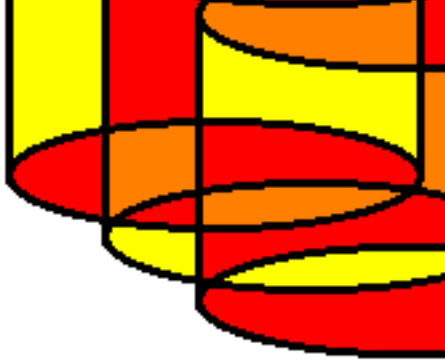
Zatim pokrenuti nekoliko instanci skripte (u više prozora konzole na primjer) da simulirate chat

```
./brblj.py --kanal nogomet --ime jože
```

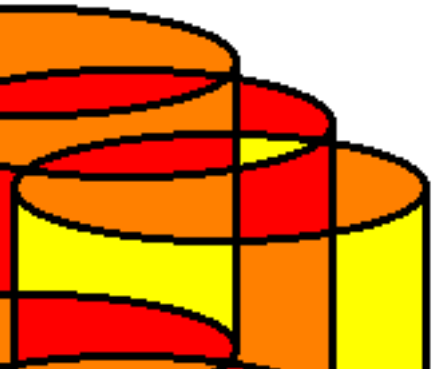
```
./brblj.py --kanal nogomet --ime ivek
```



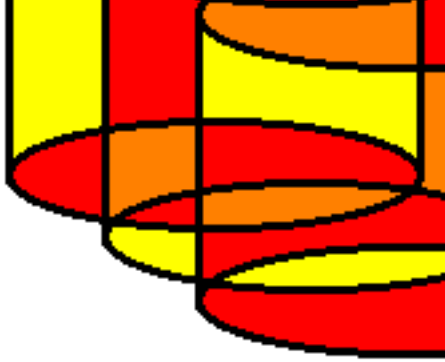
Zadatak



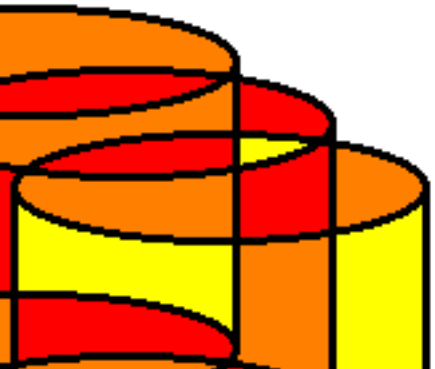
- Proširite aplikaciju **brblj.py** tako da se u objektu kanala osim poruka pohranjuju i korisnici koji su autori poruka te vremena slanja poruka.
- Implementirajte naredbu /izlistaj koja će ako se upiše u sučelje kanala ispisati sve korisnike koji su slali poruke na bilo koji kanal u posljednjih pola sata.



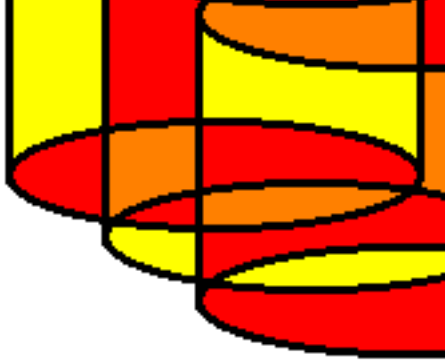
Diskusija



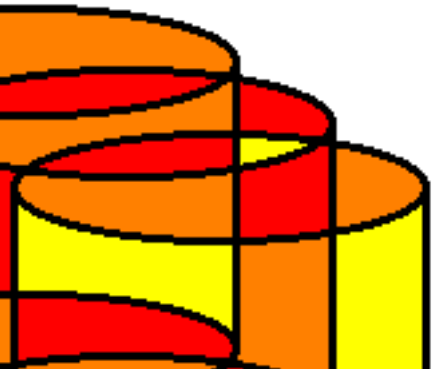
- Kada koristiti objektno-orijentirane baze podataka
  - Kod specifičnih aplikacija (CAD, CASE, mrežne aplikacije, grafika, računalne igre, mobilne aplikacije, višeagentni sustavi...) koje iziskuju kompleksne strukure podataka, a ne zahtijevaju kompleksne upite nad velikim količinama podataka (performanse!)



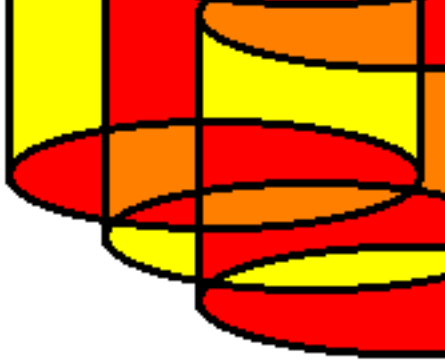
Diskusija



- Kada ne koristiti objektno-orijentirane baze podataka
  - Kod aplikacija koje se temelje u načelu na jednostavnim strukturama podataka i/ili iziskuju rad s velikim količinama podataka s kompleksnim upitima.



Diskusija



- Što ako imamo kompleksne strukture podataka i potrebu za dobrim performansama na velikim količinama podataka?
  - Preporuka: poopćene i objektno-relacijske baze podataka