FUNKCIONALNO PROGRAMIRANJE

Oznaka predmeta: FPR

Predavanje broj: 08

Nastavna jedinica: PYTHON,

Nastavne teme:

SMTP (slanje emaila, attachment-a). Multithreading (kreiranje, startovanje, sinhronizacija, korišćenje reda poslova). XML (parsiranje korišćenjem tehnologija: SAX, DOM i JSON).

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

Literatura:

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

Python: SMTP

- Simple Mail Transfer Protocol (SMTP) je protokol koji rukuje slanjem e-mail-a i rutiranjem e-mail-a između mail servera.
- Python obezbeđuje **smtplib** modul koji definiše SMTP klijent sesijski objekat koji se koristi za slanje email-a bilo kom računaru na Internetu koji ima SMTP ili ESMTP daemon osluškivač.

```
import smtplib
smtpObj = smtplib.SMTP( [host[,port[,local_hostname]]] )
```

  - **host**: host na kom radi SMTP server. Opcionalno može se specificira IP adresa hosta ili naziv domena.
  - **port**: ako je obezbeđen argument *host* onda je potrebno specificirati port na kom osluškuje SMTP server (obično 25).
  - **local_hostname**: ako SMTP server radi na lokalnoj mašini onda se specificira *localhost* kao opcija.
- SMTP objekat ima metod **sendmail** koji se koristi za poruke i ima sledeće parametre:
  - *sender* - string adrese pošiljaoca.
  - *receivers* - lista stringova (1 po primaocu).
  - *message* - string poruka.

Python: SMTP

- Sledi primer koji šalje e-mail u kome je poruka formatirana korišćenjem trostrukih navodnika i koja ima korektan format (From, To i Subject zaglavlje odvojeno od tela poruke jednom praznom linijom):

```
#!/usr/bin/python
import smtplib
sender = 'from@fromdomain.com'
receivers = ['to@todomain.com']
message = """From: From Person <from@fromdomain.com>
To: To Person <to@todomain.com>
Subject: SMTP e-mail test

This is a test e-mail message.
""" # moglo bi i programski da se pripremi
try:
   smtpObj = smtplib.SMTP('localhost')
   smtpObj.sendmail(sender, receivers, message)
   print ("Successfully sent email")
except smtplib.SMTPException:
   print ("Error: unable to send email")
```

- Za slanje e-mail-a u primeru korišćen je smtpObj koji je konektovan na SMTP server na lokalnom računaru.

Python: SMTP

- Ako se ne koristi lokalni SMTP server onda se koristi smtplib klijent za komunikaciju sa udaljenim SMTP serverom.

```
smtplib.SMTP('mail.your-domain.com', 25)
```

- Kada se e-mail-om šalje tekst poruka korišćenjem Python-a kompletan sadržaj se tretira kao tekst, čak i kada bi se u tekst poruku uključili HTML tagovi i dalje bi poruka bila tretirana kao običan tekst.
- Python obezbeđuje opciju da se HTML poruka zaista šalje kao takva i da se kao takva i tretira što se čini specificiranjem MIME verzije i tipa sadržaja.

```
#!/usr/bin/python
import smtplib
sender = 'from@fromdomain.com'
receivers = ['to@todomain.com']
message = """From: From Person <from@fromdomain.com>
To: To Person <to@todomain.com>
MIME-Version: 1.0
Content-type: text/html
Subject: SMTP HTML e-mail test

This is an e-mail message to be sent in HTML format
<b>This is HTML message.</b>
<h1>This is headline.</h1>
"""
```

Python: SMTP

```
try:
   smtpObj = smtplib.SMTP('localhost')
   smtpObj.sendmail(sender, receivers, message)
   print ("Successfully sent email")
except smtplib.SMTPException:
   print ("Error: unable to send email")
```

Slanje attachment-a e-mail-om

- Potrebno je postaviti da je **Content-type** zaglavlje **multipart/mixed**. Nakon ovoga može se specificirati attachment unutar **boundary** sekcije.
  - Boundary počinje sa jedinstvenim markerom, a završava sa tim markerom ispred koga stoje dve crtice.
- Prikačeni fajlovi trebalo bi da budu kodovani (npr. base64) pre slanja.

```
#!/usr/bin/python
import smtplib
import base64
filename = "test.txt"
# Read a file and encode it into base64 format
fo = open(filename, "rb")
filecontent = fo.read()
encodedcontent = base64.b64encode(filecontent) # base64
```

Python: SMTP

```
sender = 'me@fromdomain.net'
reciever = 'amrood.admin@gmail.com'
marker = "AUNIQUEMARKER"
body ="""  This is a test email to send an attachement.  """
# Define the main headers.
part1 = """From: From Person <me@fromdomain.net>
To: To Person <amrood.admin@gmail.com>
Subject: Sending Attachement
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=%s
--%s
""" % (marker, marker)
# Define the message action
part2 = """Content-Type: text/plain
Content-Transfer-Encoding:8bit

%s
--%s
""" % (body,marker)
# Define the attachment section
part3 = """Content-Type: multipart/mixed; name=\"%s\"
Content-Transfer-Encoding:base64
Content-Disposition: attachment; filename=%s
```

Python: multithreading

```
%s
--%s
""" %(filename, filename, encodedcontent, marker)
message = part1 + part2 + part3
try:
   smtpObj = smtplib.SMTP('localhost')
   smtpObj.sendmail(sender, reciever, message)
   print ("Successfully sent email")
except Exception:
   print ("Error: unable to send email")
```

Višenitno (Multithreading) programiranje u Python-u

- Izvršavanje nekoliko niti je slično paralelenom izvršavanju nekoliko različitih programa a dobici su:
  - Više niti unutar procesa deli isti prostor podataka sa glavnom niti čime je olakšana međusobna komunikacija nego u slučaju da su u pitanju odvojeni procesi.
  - Niti (Threads) se ponekad nazivaju light-weight procesi jer ne zahtevaju mnogo memorije (jeftinije su u tom smislu od procesa).

Python: multithreading

- Niti imaju početak, sekvencu izvršavanja i završetak.
  - Niti su pre-empted (interrupted, mogu biti prekinute)
  - Mogu se privremeno staviti u sleep stanje dok se ostale niti izvršavaju (yielding).

Startovanje nove niti

- Za startovanje nove niti potrebno je pozvati konstruktor `Thread` iz modula `threading`:

```
t = Thread ( target=function_name, args=(ar1,ar2,...) )
```

  - Ovaj metod omogućuje brz i efikasan način za kreiranje nove niti i u Linux-u i u Windows-u.
  - U primeru kreiranja niti `args` predstavlja n-torku argumenata.
  - Za poziv funkcije bez argumenata koristi se prazna n-torka.
  - Kreirana nit se startuje pozivom metode start.

```
t.start()
```

    koja startuje metodu `run()` koju treba redefinisati.

  - Kada se desi povratak iz funkcije niti onda ta nit završava svoj rad.

Python: multithreading

- Threading modul ima sve metode starog modula thread (zastareo ali se u 3.5 može koristiti kao modul _thread) i obezbeđuje neke dodatne metode:
  - threading.current_thread() : vraća tekuću nit (ili currentThread()).
  - threading.active_count() : vraća broj aktivnih niti (ili activeCount()).
  - threading._enumerate() : enumeracija niti (ili enumerate()).
- Threading modul ima klasu Thread čije su metode kao što sledi:
  - run(): predstavlja ulaznu tačku niti.
  - start(): startuje nit pozivanjem njene run metode.
  - join([time]): čeka da nit završi posao.
  - isAlive(): proverava da li se nit još izvršava.
  - getName(): vraća naziv niti.
  - setName(): postavlja naziv niti.
- Za kreiranje vlastite niti korišćenjem threading modula potrebno je uraditi kao što sledi:
  - Definisati novu klasu izvedenu iz klase Thread.
  - Nadjačati __init__(self [,args]) metod dodavanjem potrebnih argumenata
  - Nadjačati run(self) metod implementacijom šta bi nit trebalo da radi.

Python: multithreading

```
#!/usr/bin/python
import threading
import time
# Define a function for the thread
def print_time(threadName, delay):
   count = 0
   while count < 5:
      time.sleep(delay)
      count += 1
      print ("%s: %s" % ( threadName, time.ctime(time.time()) ) )
      print(threading.current_thread().getName())
# Create two threads as follows
try:
   t1 = threading.Thread( target=print_time, args=("Thread-1", 1 ) )
   t2 = threading.Thread( target=print_time, args=("Thread-2", 2 ) )
   t1.setName("Elvis")
   t1.start()
   t2.setName("Tom Jones")
   t2.start()
except:
   print ("Error: unable to start thread")
print(threading.active_count())
print(threading._enumerate())
```

Python: multithreading

```
3
[<Thread(Tom Jones, started 3240)>, <Thread(Elvis, started 4780)>,
<_MainThread(MainThread, started 4100)>]
Thread-1: Sun Dec  6 16:17:45 2015
Elvis
Thread-2: Sun Dec  6 16:17:46 2015
Tom Jones
Thread-1: Sun Dec  6 16:17:46 2015
Elvis
Thread-1: Sun Dec  6 16:17:47 2015
Elvis
Thread-2: Sun Dec  6 16:17:48 2015
Tom Jones
Thread-1: Sun Dec  6 16:17:48 2015
Elvis
Thread-1: Sun Dec  6 16:17:49 2015
Elvis
Thread-2: Sun Dec  6 16:17:50 2015
Tom Jones
Thread-2: Sun Dec  6 16:17:52 2015
Tom Jones
Thread-2: Sun Dec  6 16:17:54 2015
Tom Jones

Process finished with exit code 0
```

Python: multithreading

- Kreiranje klase koja nasleđuje klasu `threading.Thread` je kao što sledi:

```
import threading
import time

class myThread (threading.Thread):
  def __init__(self, threadID, name, delay):
    threading.Thread.__init__(self)
    self.threadID = threadID
    self.name = name
    self.delay = delay
  def run(self):
    print ("Starting " + self.name)
    print_time(self.name, self.delay, 5)
    print ("Exiting " + self.name)

def print_time(threadName, delay, counter):
  while counter:
     time.sleep(delay)
     print ("%s: %s" % (threadName, time.ctime(time.time())))
     counter -= 1
```

- Sada se kreira instanca ove klase i startuje nova nit start() metodom (a ona poziva run() metodu).

Python: multithreading

```
# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)
# Start new Threads
thread1.start()
thread2.start()
print ("Main Thread")
```

```
Starting Thread-1
Starting Thread-2
Main Thread
Thread-1: Sun Dec  6 16:45:36 2015
Thread-1: Sun Dec  6 16:45:37 2015
Thread-2: Sun Dec  6 16:45:37 2015
Thread-1: Sun Dec  6 16:45:38 2015
Thread-1: Sun Dec  6 16:45:39 2015
Thread-2: Sun Dec  6 16:45:39 2015
Thread-1: Sun Dec  6 16:45:40 2015
Exiting Thread-1
Thread-2: Sun Dec  6 16:45:41 2015
Thread-2: Sun Dec  6 16:45:43 2015
Thread-2: Sun Dec  6 16:45:45 2015
Exiting Thread-2
```

Python: multithreading

Sinhronizacija niti

- Threading modul obezbeđuje mehanizam zaključavanja koji omogućuje da se
  - nova brava kreira pozivom `threading.Lock()` metode koja vraća novu bravu.
- Brava ima metod `acquire(blocking=1)` koji forsira nit da radi sinhronizovano.
  - Opcionalni parametar blocking omogućuje da se kontroliše kada nit čeka da dobije bravu.
    - Ako je blocking == 1 nit se blokira i čeka se na oslobađanje brave.
    - Ako je blocking == 0 onda nit odmah vraća vrednost 0 ako se brava ne može dobiti, odnosno, vraća vrednost 1 ako je brava dobijena.
- Brava ima metod `release`() koji se koristi za oslobađanje brave kada nije potrebna.

```
#!/usr/bin/python
import threading
import time
class myThread (threading.Thread):
  def __init__(self, threadID, name, delay):
    threading.Thread.__init__(self)
    self.threadID = threadID
    self.name = name
```

Python: multithreading

```
    self.delay = delay
  def run(self):
    print ("Starting " + self.name)
    # Get lock to synchronize threads
    threadLock.acquire()
    print_time(self.name, self.delay, 3)
    # Free lock to release next thread
    threadLock.release()
def print_time(threadName, delay, counter):
  while counter:
    time.sleep(delay)
    print ("%s: %s" % (threadName, time.ctime(time.time())) )
    counter -= 1

threadLock = threading.Lock()
threads = []
# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)
# Start new Threads
thread1.start()
thread2.start()
# Add threads to thread list
```

Python: multithreading

```
threads.append(thread1)
threads.append(thread2)
# Wait for all threads to complete
for t in threads:
  t.join()
print ("Exiting Main Thread")
```

```
Starting Thread-1
Starting Thread-2
Thread-1: Sun Dec  6 17:21:00 2015
Thread-1: Sun Dec  6 17:21:01 2015
Thread-1: Sun Dec  6 17:21:02 2015
Thread-2: Sun Dec  6 17:21:04 2015
Thread-2: Sun Dec  6 17:21:06 2015
Thread-2: Sun Dec  6 17:21:08 2015
Exiting Main Thread
```

Niti i korišćenje reda

- Modul queue omogućuje da se kreira novi red koji sadrži određeni broj članova.
- Slede metodi koji kontrolišu navedeni red:
  - get(): uklanja i vraća taj uklonjeni član reda.
  - put(): dodaje član u red.

Python: multithreading

- qsize() : vraća trenutni broj članova u redu.
- empty(): vraća True ako je red prazan (u suprotnom vraća False).
- full(): vraća True ako je red pun (u suprotnom vraća False).

```
#!/usr/bin/python
import queue
import threading
import time
exitFlag = 0
class myThread (threading.Thread):
  def __init__(self, threadID, name, q):
    threading.Thread.__init__(self)
    self.threadID = threadID
    self.name = name
    self.q = q
  def run(self):
    print ("Starting " + self.name)
    process_data(self.name, self.q)
    print ("Exiting " + self.name)
def process_data(threadName, q):
  while not exitFlag:
    queueLock.acquire()
```

Python: multithreading

```
    if not q.empty():
      data = q.get()
      queueLock.release()
      print ("%s processing %s" % (threadName, data))
    else:
      queueLock.release()
    time.sleep(1)

queueLock = threading.Lock()
threadList = ["Thread-1", "Thread-2", "Thread-3"]
nameList = ["One", "Two", "Three", "Four", "Five"]
workQueue = queue.Queue(10)
threads = []
threadID = 1
# Create new threads
for tName in threadList:
  thread = myThread(threadID, tName, workQueue)
  thread.start()
  threads.append(thread)
  threadID += 1
# Fill the queue
for word in nameList:
  workQueue.put(word)
```

Python: multithreading

```
# Wait for queue to empty
while not workQueue.empty():
  pass
# Notify threads it's time to exit
exitFlag = 1
# Wait for all threads to complete
for t in threads:
  t.join()
print ("Exiting Main Thread")
```

```
Starting Thread-1
Starting Thread-2
Starting Thread-3
Thread-1 processing One
Thread-2 processing Two
Thread-3 processing Three
Thread-1 processing Four
Thread-2 processing Five
Exiting Thread-3
Exiting Thread-1
Exiting Thread-2
Exiting Main Thread
```

Python i XML

- XML je tehnologija koja je namenjena za standardizovanu razmenu podataka između aplikacija bez obzira na platformu i programski jezik koji je korišćen u razvoju pomenutih aplikacija.
- Extensible Markup Language (XML) je opisni jezik sličan HTMLu.
- XML je preporučen od strane World Wide Web Consortium-a i dostupan je kao otvoreni standard.
- XML je podesan za čuvanje male i srednje količine podataka.
- Python-ova xml biblioteka obezbeđuje minimalan skup interfejsa za rad sa XMLom.
- Dva najčešće korišćena API-ja za XML podatke su interfejsi SAX i DOM.
- Simple API for XML (SAX) : podesan je kada postoje memorijska ograničenja ili su veliki dokumenti u pitanju.
- Document Object Model (DOM) API : predstavlja preporuku World Wide Web Consortium-a gde se sadržaj fajla učita u memoriju u hijerarhiskom (tree-based) obliku.

Python: XML

- Korišćenje DOMa zauzima resurse čak i kada radi sa malim fajlovima.
- SAX ne može obraditi informacije brzo kao DOM ako su u pitanju veliki fajlovi.
- SAX je read-only, dok DOM dozvoljava promene u XML fajlu.
- Dat je fajl XML fajl *movies.xml* koji predstavlja ulaz za sve XML primere koji slede:

```
<collection shelf="New Arrivals">
<movie title="Enemy Behind">
   <type>War, Thriller</type>
   <format>DVD</format>
   <year>2003</year>
   <rating>PG</rating>
   <stars>10</stars>
   <description>Talk about a US-Japan war</description>
</movie>
<movie title="Transformers">
   <type>Anime, Science Fiction</type>
   <format>DVD</format>
   <year>1989</year>
   <rating>R</rating>
   <stars>8</stars>
   <description>A scientific fiction</description>
</movie>
```

Python: XML

```
<movie title="Trigun">
   <type>Anime, Action</type>
   <format>DVD</format>
   <episodes>4</episodes>
   <rating>PG</rating>
   <stars>10</stars>
   <description>Manga, western fiction</description>
</movie>
<movie title="Ishtar">
   <type>Comedy</type>
   <format>VHS</format>
   <rating>PG</rating>
   <stars>2</stars>
   <description>Viewable boredom</description>
</movie>
</collection>
```

Parsiranje XMLa korišćenjem SAX API-ja

- SAX je standardni interfejs za parsiranje XML pokrenut događajem (event-driven XML parsing).
- Parsiranje XMLa korišćenjem SAXa zahteva da se kreira korisnički ContentHandler koji nasleđuje `xml.sax.ContentHandler`.

Python: XML

- Korisnički ContentHandler rukuje tagovima i atributima datog XMLa.
  - ContentHandler obezbeđuje metode koje rukuju različitim događajima parsiranja.
  - Metode `startDocument` i `endDocument` se pozivaju na početku i na kraju XML fajla.
  - ContentHandler se zove na početku i na kraju svakog elementa.
    - Ako parser nije u modu namespace onda se pozivaju metode:
      `startElement(tag, attributes)` i `endElement(tag)`.
      U suprotnom se pozivaju odgovarajuće metode
      `startElementNS` i `endElementNS`.
      - » Parametar tag se odnosi na dati element
      - » Parametar attributes se donosi na pripadne attribute tag-a
- Metoda make_parser kreira objekat parser i vraća isti.
  - Parser objekat je tipa kog sistem prvog nađe.

```
xml.sax.make_parser( [parser_list] )
```

parser_list: opcionalni argument koji se sastoji od liste parsera koji mogu implementirati metodu make_parser.

Python: XML

- Metod parse kreira SAX parser i koristi ga za parsiranje dokumenta.

  `xml.sax.parse( xmlfile, contenthandler[, errorhandler])`

  - xmlfile: naziv XML fajla iz koga će se čitati.
  - contenthandler: ContentHandler objekat.
  - errorhandler: ako je specificiran mora biti `SAXErrorHandler` objekat.
- Metod parseString služi za kreiranje SAX parsera i da parsira XML string.

  `xml.sax.parseString(xmlstring, contenthandler[, errorhandler])`

  - xmlstring: naziv XML stringa iz koga se čita.
  - contenthandler: ContentHandler objekat.
  - errorhandler: ako je specificiran mora biti `SAXErrorHandler` objekat.
- Primer:

```
#!/usr/bin/python
import xml.sax
class MovieHandler( xml.sax.ContentHandler ):
  def __init__(self):
    self.CurrentData = ""
    self.type = ""
    self.format = ""
    self.year = ""
```

Python: XML

```
    self.rating = ""
    self.stars = ""
    self.description = ""
  # Call when an element starts
  def startElement(self, tag, attributes):
    self.CurrentData = tag
    if tag == "movie":
      print ("*****Movie*****")
      title = attributes["title"]
      print ("Title:", title)
  # Call when an elements ends
  def endElement(self, tag):
    if self.CurrentData == "type":
      print ("Type:", self.type)
    elif self.CurrentData == "format":
      print ("Format:", self.format )
    elif self.CurrentData == "year":
      print ("Year:", self.year )
    elif self.CurrentData == "rating":
      print ("Rating:", self.rating )
    elif self.CurrentData == "stars":
      print ("Stars:", self.stars)
```

Python: XML

```
    elif self.CurrentData == "description":
      print ("Description:", self.description)
    self.CurrentData = ""
  # Call when a characters is read
  def characters(self, content):
    if self.CurrentData == "type":
      self.type = content
    elif self.CurrentData == "format":
      self.format = content
    elif self.CurrentData == "year":
      self.year = content
    elif self.CurrentData == "rating":
      self.rating = content
    elif self.CurrentData == "stars":
      self.stars = content
    elif self.CurrentData == "description":
      self.description = content
if ( __name__ == "__main__"):
  # create an XMLReader
  parser = xml.sax.make_parser()
  # turn off namepsaces
  parser.setFeature(xml.sax.handler.feature_namespaces, 0)
```

Python: XML

```
# override the default ContextHandler
Handler = MovieHandler()
parser.setContentHandler( Handler )
parser.parse("movies.xml")
```

```
*****Movie*****
Title: Enemy Behind
Type: War, Thriller
Format: DVD
Year: 2003
Rating: PG
Stars: 10
Description: Talk about a US-Japan war
*****Movie*****
Title: Transformers
Type: Anime, Science Fiction
Format: DVD
Year: 1989
Rating: R
Stars: 8
Description: A scientific fiction
*****Movie*****
Title: Trigun
Type: Anime, Action
Format: DVD
Rating: PG
Stars: 10
Description: Manga, western fiction
*****Movie*****
Title: Ishtar
Type: Comedy
Format: VHS
Rating: PG
Stars: 2
Description: Viewable boredom
```

Parsiranje XMLa sa DOM API-jem

- Document Object Model ("DOM") je cross-language API iz World Wide Web Consortium-a (W3C) za pristup i modifikaciju XML dokumenata.
- DOM je ekstremno koristan za random-access.
  - SAX dozvoljava da se u jednom trenutku vidi samo deo dokumenta (ako se gleda jedan SAX element drugom se ne može pristupiti).
- Ideja je da se (brzo) učita XML dokument i da se onda kreira minidom objekat iz modula `xml.dom.minidom`.
  - Objekat minidom obezbeđuje jednostavnu metodu za parsiranje koja (brzo) kreira DOM stablo iz XML fajla.
    - Poziv `parse`(file [,parser]) funkcije iz minidom-a parsira se XML file u objekat DOM stablo.

```
#!/usr/bin/python
import xml.dom.minidom
# Open XML document using minidom parser
DOMTree = xml.dom.minidom.parse("movies.xml")
collection = DOMTree.documentElement
```

Python: XML

```
if collection.hasAttribute("shelf"):
   print ("Root element : %s" % collection.getAttribute("shelf") )
# Get all the movies in the collection
movies = collection.getElementsByTagName("movie")
# Print detail of each movie.
for movie in movies:
   print ("*****Movie*****" )
   if movie.hasAttribute("title"):
      print ("Title: %s" % movie.getAttribute("title") )

   type = movie.getElementsByTagName('type')[0]
   print ("Type: %s" % type.childNodes[0].data )
   format = movie.getElementsByTagName('format')[0]
   print ("Format: %s" % format.childNodes[0].data )
   rating = movie.getElementsByTagName('rating')[0]
   print ("Rating: %s" % rating.childNodes[0].data )
   description = movie.getElementsByTagName('description')[0]
   print ("Description: %s" % description.childNodes[0].data)
```

```
Root element : New Arrivals
*****Movie*****
Title: Enemy Behind
Type: War, Thriller
Format: DVD
```

Python: XML

```
Rating: PG
Description: Talk about a US-Japan war
*****Movie*****
Title: Transformers
Type: Anime, Science Fiction
Format: DVD
Rating: R
Description: A scientific fiction
*****Movie*****
Title: Trigun
Type: Anime, Action
Format: DVD
Rating: PG
Description: Manga, western fiction
*****Movie*****
Title: Ishtar
Type: Comedy
Format: VHS
Rating: PG
Description: Viewable boredom
```

Python: JSON

- JSON (JavaScript Object Notation) predstavlja format za razmenu podataka.
- Izveden je iz JavaScript-a i koristi se za prenos strukturiranih podataka preko mreže za razmenu podataka između servera i web aplikacije (zamena za XML).

```
import json
class User:
    staticni_podatak="podatak klase"
    def __init__(self, name, surname):
        self.name = name
        self.surname = surname
    def to_json(self):
        return json.dumps(self.__dict__)
    def __str__(self):
        return self.name+" "+self.surname
    @classmethod
    def from_json(cls, json_str):
        json_dict = json.loads(json_str)
        return cls(**json_dict)
    @staticmethod
    def f(arg):
        print(User.staticni_podatak+arg); #moze i ovo ali je vise za g.fun
        return
```

Python: JSON

```
dictUser = User("Elvis", "Presley").to_json()
print(dictUser)
```

```
{"name": "Elvis", "surname": "Presley"}
```

```
fp = open('jsonpodaci.txt','w')
json.dump(dictUser,fp)
fp.close()
```

```
objUser = User.from_json(dictUser);
print(objUser);
```

```
Elvis Presley
```

```
fp = open('jsonpodaci.txt','r')
print( User.from_json(json.load(fp)) );
```

```
Elvis Presley
```

```
fp.close()
```

```
User.f(" <--")
```

```
podatak klase <--
```

Sadržaj jsonpodaci.txt će biti:

```
"{\"name\": \"Elvis\", \"surname\": \"Presley\"}"
```

Za 3. čas

```
#mozda da urade da
#dobra (brza) nit povecava ocenu
#a losa nit (spora) smanjuje ocenu, cilj je 10
__author__ = 'Pero'
import threading
import time
ocena=5
def MenjajOcenu(menjaj,pauza):
    global ocena
    while(True):
        if(ocena==10):  return
        else:
            lock.acquire()
            ocena+=menjaj
            print(ocena)
            lock.release()
            time.sleep(pauza)
lock = threading.Lock()
t1=threading.Thread(target=MenjajOcenu,args=(1,0.5))
t2=threading.Thread(target=MenjajOcenu,args=(-1,1))
t1.start()
t2.start()
```