PROGRAMIRANJE U INTEGRISANIM TEHNOLOGIJAMA

Oznaka predmeta: PIT

Predavanje broj: 04

Nastavna jedinica: PYTHON,

Nastavne teme:

Cookie (snimanje i uzimanje vrednosti). Baze podataka sqlite3, pymysql (konekcija, kreiranje baze, upiti, uzimanje rezultata upita, kreiranje, dodavanje, menjanje i brisanje zapisa). Mrežno programiranje (socket, klijent-server arhitektura). SMTP (slanje emaila, attachment-a). Multithreading (kreiranje, startovanje, sinhronizacija, korišćenje reda poslova). JSON.

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

Literatura:

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

Python: CGI, cookie

- HTTP protokol je protokol koji ne pamti stanje (stateless) tako da se koriste cookie-i.
- Server šalje podatke ka browser-u klijenta u obliku cookie-a.
  - Ako browser prihvati cookie isti će biti snimljen na hard disku klijenta.
  - Ovim će posetilac pristupajući sledeći put ovom sajtu dati informaciju serveru o navedenom cookie-u.
- Cookie se smešta kao običan nekriptovan tekst (plain text) i može da sadrži sledeća polja:
  - Name=Value : cookie se postavlja i dobija u formi para key-value.
  - Expires : vreme isteka cookie-a. Ako je polje prazno onda će cookie isteći kada posetilac zatvori browser.
  - Domain : domen sajta.
  - Path: staza do direktorijuma ili web stranice koja postavlja cookie. Ako je polje prazno onda se cookie može dobiti iz bilo kog direktorijuma ili stranice.
  - Secure: ako ovo polje sadrži sigurnosnu reč onda se cookie može dobiti samo preko sigurnosnog servera (secure server). Ako je polje prazno nema navedenih restrikcija.

Python: cookie

- Dobijanje cookie-a (Retrieving Cookies)
  - Cookie-i su smešteni u os.environ asocijativnom ulazu HTTP_COOKIE u formi kao što sledi:

```
key1=value1; key2=value2; key3=value3; ....
```

```
#!C:/Users/Pero/AppData/Local/Programs/Python/Python35/python.exe
import os
import cgitb
cgitb.enable()
print ("Content-type: text/plain\r\n\r\n")
if "HTTP_COOKIE" in os.environ:
    print(os.environ['HTTP_COOKIE'].split(';'))
    for cookie in os.environ['HTTP_COOKIE'].split(';'):
      (key, value) = str(cookie).split('=')
      print(key.lstrip(),"-->",value)
else:
    print("HTTP_COOKIE not set!")
```

- Dati skript `prikazi_cookie.py` prikazuje sve cookie u formi naziv --> vrednost.
- Ako nema postavljenih cookie-a (npr. obrisani, nisu ni postavljeni ili im je vreme života isteklo) onda skript `prikazi_cookie.py` daje poruku: `HTTP_COOKIE not set!`

Python: cookie

- Postavljanje cookie-a se izvodi tako da se informacija o cookie-ima šalje uz HTTP zaglavlje pre *Content-type* polja.
- U sledećem primeru postavljaju se tri cookie-a, prvi pomoću modula http.cookies a druga dva bez korišćenja navedenog modula.

```
#!C:/Users/Pero/AppData/Local/Programs/Python/Python35/python.exe
    import http.cookies
    import datetime
    import random
    import cgitb
    cgitb.enable()
    expiration = datetime.datetime.now() + datetime.timedelta(days=30)
    cookie = http.cookies.SimpleCookie()
    cookie["session"] = random.randint(1,1000000000)
    cookie["session"]["domain"] = ".localhost" #kod Mozille
    cookie["session"]["path"] = "/cgi-bin/"    #kod Mozille
    cookie["session"]["expires"] = \
    expiration.strftime("%a, %d-%b-%Y %H:%M:%S GMT")
    print ("Set-Cookie: Name=Presley; expires=Tuesday, 29-Dec-2020 23:12:40 GMT;" )
    print ("Set-Cookie: Ford=Mustang; expires=Tuesday, 29-Dec-2020 23:12:40 GMT;" )
    print (cookie.output())
    print ("Content-type: text/plain\r\n\r\n")
    print ("Cookie preko http.cookies.SimpleCookie jeste:\n"+cookie.output())
```

Python: cookie

```
Cookie preko http.cookies.SimpleCookie jeste:

Set-Cookie: session=948797479; Domain=.localhost; expires=Wed, 30-Dec-
2015 20:32:18 GMT; Path=/cgi-bin/
```

- Ako se nakon ovoga pozove prikazi_cookie.py

```
['session=948797479', ' Name=Presley', ' Ford=Mustang']
session --> 948797479
Name -- > Elvis
Ford --> Mustang
```

- U navedenom primeru koristi se Set-Cookie HTTP zaglavlje da bi se postavio cookie. Atributi cookie-a: Expires, Domain i Path su opcioni.
- Ako se želi uzeti vrednost cookie-a Name onda se to može uraditi kao što sledi:

```
#!C:/Users/Pero/AppData/Local/Programs/Python/Python35/python.exe
import http.cookies
import os
print ("Content-type: text/plain\n")
try:
    cookie = http.cookies.SimpleCookie(os.environ["HTTP_COOKIE"])
    print ("Name = " + cookie["Name"].value)
except (http.cookies.CookieError, KeyError):
    print ("Name cookie not set!")
```

```
Name = Elvis
```

Python: DB, sqlite3

- SQLite je softverska biblioteka koja implementira *self-contained, serverless, zero-configuration, transactional SQL database engine*.
  - Izvorni kod SQLite-a je u javnom domenu.
- SQLite3 se uključuje u Python 3.5 korišćenjem modula sqlite3.

```
import sqlite3
conn = sqlite3.connect('test.db') #ako je ":memory:" onda je sve u RAMu
print ("Opened database successfully")
conn.execute('''CREATE TABLE COMPANY
       (ID              INT PRIMARY KEY     NOT NULL,
       NAME             TEXT                NOT NULL,
       AGE              INT                 NOT NULL,
       ADDRESS          CHAR(50),
       SALARY           REAL);''')
print ("Table created successfully")
conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (1, 'Paul', 32, 'California', 20000.00 )");
conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (2, 'Allen', 25, 'Texas', 15000.00 )");
conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (3, 'Teddy', 23, 'Norway', 20000.00 )");
conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 )");
```

Python: DB, sqlite3

```
conn.commit()
print ("Records created successfully")
conn.close()
```

```
Opened database successfully
Table created successfully
Records created successfully
```

- Ostvarena je konekcija sa bazom, kreirana je tabela koja je potom popunjena podacima i na kraju je zatvorena konekcija sa bazom.
- Sledeći kod vrši selekciju zapisa i prikazuje dobijene rezultate te selekcije.

```
import sqlite3
conn = sqlite3.connect('test.db')
print ("Opened database successfully")
cursor = conn.execute("SELECT id, name, address, salary  from COMPANY")
for row in cursor:
   print ("ID = ", row[0])
   print ("NAME = ", row[1])
   print ("ADDRESS = ", row[2])
   print ("SALARY = ", row[3], "\n")
print ("Operation done successfully")
conn.close()
```

Python: DB, sqlite3

```
Opened database successfully
ID =  1
NAME =  Paul
ADDRESS =  California
SALARY =  20000.0

ID =  2
NAME =  Allen
ADDRESS =  Texas
SALARY =  15000.0

ID =  3
NAME =  Teddy
ADDRESS =  Norway
SALARY =  20000.0

ID =  4
NAME =  Mark
ADDRESS =  Rich-Mond
SALARY =  65000.0

Operation done successfully
```

- U sledećem primeru vrši se ažuriranje zapisa. Ostvarena je konekcija sa bazom, kreirana je tabela koja je potom popunjena podacima i na kraju je zatvorena konekcija sa bazom.

Python: DB, sqlite3

```
import sqlite3
conn = sqlite3.connect('test.db')
print ("Opened database successfully")
conn.execute("UPDATE COMPANY set SALARY = 25000.00 where ID=1")
conn.commit()
print ("Total number of rows updated :", conn.total_changes)
cursor = conn.execute("SELECT id, name, address, salary  from COMPANY")
for row in cursor:
   print ("ID = ", row[0])
   print ("NAME = ", row[1])
   print ("ADDRESS = ", row[2])
   print ("SALARY = ", row[3], "\n")
print ("Operation done successfully")
conn.close()
```

```
Opened database successfully
Total number of rows updated : 1
ID =  1
NAME =  Paul
ADDRESS =  California
SALARY =  25000.0
...
Operation done successfully
```

Python: DB, sqlite3

- Brisanje podataka ja dato u sledećem primeru gde će se obrisati svi zapisi kod kojih je ID različit od 2.

```
import sqlite3
conn = sqlite3.connect('test.db')
print ("Opened database successfully")
conn.execute("DELETE from COMPANY where ID<>2;")
conn.commit()
print ("Total number of rows deleted :", conn.total_changes)
cursor = conn.execute("SELECT id, name, address, salary  from COMPANY")
for row in cursor:
   print ("ID = ", row[0])
   print ("NAME = ", row[1])
   print ("ADDRESS = ", row[2])
   print ("SALARY = ", row[3], "\n")
print ("Operation done successfully")
conn.close()
```

```
Opened database successfully
Total number of rows deleted : 3
ID =  2
NAME =  Allen
ADDRESS =  Texas
SALARY =  15000.0
Operation done successfully
```

Python: DB, sqlite3

- Korišćenje objekta cursor koji se dobija preko otvorene konekcije.

```
import sqlite3
conn = sqlite3.connect('example.db')
c = conn.cursor()
c.execute('''CREATE TABLE stocks(date text, trans text, symbol text, qty
real, price real)''')
c.execute("INSERT INTO stocks VALUES ('2006-01-05', 'BUY', 'RHAT' ,100
,35.14)")
conn.commit()
conn.close()
conn = sqlite3.connect('example.db')
c = conn.cursor()
t = ('RHAT',)
c.execute('SELECT * FROM stocks WHERE symbol=?', t) #promenljiva t
print (c.fetchone())
# visestruko izvrsavanje
purchases = [('2006-03-28', 'BUY', 'IBM', 1000, 45.00),
             ('2006-04-05', 'BUY', 'MSFT', 1000, 72.00),
             ('2006-04-06', 'SELL', 'IBM', 500, 53.00),
            ]
c.executemany('INSERT INTO stocks VALUES (?,?,?,?,?)', purchases)
for row in c.execute('SELECT * FROM stocks ORDER BY price'):
    print (row)
```

Python: DB, sqlite3

- Korišćenje :memory:

```
import sqlite3

class Point(object):                # izvedena iz object, u 3.5 može bez
        def __init__(self, x, y):
            self.x, self.y = x, y

def adapt_point(point):
        return "%f;%f" % (point.x, point.y)

sqlite3.register_adapter(Point, adapt_point)

con = sqlite3.connect(":memory:")
    cur = con.cursor()
    p = Point(4.0, -3.2)
    cur.execute("select ?", (p,))
    print (cur.fetchone()[0])
                          4.000000;-3.200000
```

Python, pymysql

- API za baze podataka omogućuje:
  - Importovanje API modula.
  - Zahtevanje konekecije sa bazom podataka.
  - Baratanje SQL naredbama i uskladištenim procedurama.
  - Zatvaranje konekcije.
- pymysql je interfejs za konekciju sa MySQL database server-om iz Python-a.
- Da bi se proverilo da li postoji instalacija pymysql na datom računaru trebalo bi pokrenuti sledeći skript:

```
#!/usr/bin/python
import pymysql
```

ako je rezultat kao što sledi onda modul pymysql nije instaliran:

```
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    import pymysql
ImportError: No module named pymysql
```

u ovom slučaju instalirati odgovarajuću verziju na dati računar

```
pip install pymysql
```

Python, DB

- Za konketovanje na bazu podataka neka je zadovoljeno sledeće:
  - Kreirana je baza podataka TESTDB, koja sadrži tabelu EMPLOYEE koju čine sledeća polja: FIRST_NAME, LAST_NAME, AGE, SEX i INCOME.
  - Neka su user "testuser" i passwd "test123" postavljeni za pristup TESTDB-u.
  - Instalirati modul pymsql (pip install pymysql)
- Primer konekcije na MySQL bazu podatka TESTDB:

```
#!/usr/bin/python
import pymysql
# Open database connection
db =
 pymysql.connect(host="localhost",port=3306,user="testuser",
 passwd="test123",db="TESTDB")
# prepare a cursor object using cursor() method
cursor = db.cursor()
# execute SQL query using execute() method.
cursor.execute("SELECT VERSION()")
# Fetch a single row using fetchone() method.
data = cursor.fetchone()
print ("Database version : %s " % data)
# disconnect from server
db.close()
```

```
Database version : 10.1.26-MariaDB
```

Python, DB

- Kada je uspostavljena veza sa bazom podataka ide se na sledeći korak a to je kreiranje tabele EMPLOYEE:

```
#!/usr/bin/python
import pymysql
# Open database connection
db = pymysql.connect (host="localhost",port=3306,user="testuser",
        passwd="test123",db="TESTDB")
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Drop table if it already exist using execute() method.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")
# Create table as per requirement
sql = """CREATE TABLE EMPLOYEE (
                FIRST_NAME CHAR(20) NOT NULL,
                LAST_NAME  CHAR(20),
                AGE INT,
                SEX CHAR(1),
                INCOME FLOAT )"""
cursor.execute(sql)
# disconnect from server
db.close()
```

Python, DB

- Operacija INSERT koristi se za kreiranje zapisa u tabeli baze podataka:

```
#!/usr/bin/python
import pymysql
# Open database connection
db = pymysql.connect (host="localhost",port=3306,user="testuser",
        passwd="test123",db="TESTDB")
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to INSERT a record into the database.
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
                 LAST_NAME, AGE, SEX, INCOME)
                VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Commit your changes in the database
   db.commit()
except:
   # Rollback in case there is any error
   db.rollback()
# disconnect from server
db.close()
```

Python, DB

- Prethodni primer može se napisati tako da se kreiraju dinamički SQL upiti:

```
#!/usr/bin/python
import pymysql
db = pymysql.connect (host="localhost",port=3306,user="testuser",
        passwd="test123",db="TESTDB")
cursor = db.cursor()
# Prepare SQL query to INSERT a record into the database.
sql = "INSERT INTO EMPLOYEE(FIRST_NAME, \
       LAST_NAME, AGE, SEX, INCOME) \
       VALUES ('%s', '%s', '%d', '%c', '%d' )" % \
       ('Mac', 'Mohan', 20, 'M', 2000)
try:
   cursor.execute(sql)
   db.commit()
except:
   db.rollback()
db.close()
```

- Sledi segment koda gde se parametri korisnika i lozinke prosleđuju direktno

```
user_id = "Elvis"
password = "Presley"
con.execute('insert into Login values("%s", "%s")' % \
                           (user_id, password))
```

Python, DB

- READ operacija se koristi za uzimanje informacija iz baze podataka.
- Kada je jednom uspostavljena konekcija sa bazom podataka moguće je izvršiti upit ka bazi.
- Koristi se metod fetchone() za uzimanje jednog zapisa ili fetchall() metod za uzimanje više zapisa iz tabele baze podataka.
  - fetchone(): uzima sledeći red iz skupa rezultata dobijenog upitom. Skup rezultata je objekat koji je vraćen kada se koristi objekat cursor da se postavi upit.
  - fetchall(): uzima sve redove skupa rezultata. Ako su neki redovi već ekstrahovani iz skupa rezultata onda ova metoda vraća preostale redove u skupu rezultata.
  - rowcount: read-only atribut koji vraća broj redova koji su afektirani metodom execute().
- Primer: prikazivanje svih zapisa tabele EMPLOYEE kod kojih je plata veća od 1000.

```
#!/usr/bin/python
import pymysql
db = pymysql.connect (host="localhost",port=3306,user="testuser",
        passwd="test123",db="TESTDB")
```

Python, DB

```
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to INSERT a record into the database.
sql = "SELECT * FROM EMPLOYEE WHERE INCOME > '%d'" % (1000)
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Fetch all the rows in a list of lists.
   results = cursor.fetchall()
   for row in results:
      fname = row[0]
      lname = row[1]
      age = row[2]
      sex = row[3]
      income = row[4]
      # Now print fetched result
      print ("fname=%s,lname=%s,age=%d,sex=%s,income=%d" % \
            (fname, lname, age, sex, income ) )
except:
   print ("Error: unable to fecth data")
db.close()
```

```
fname=Mac, lname=Mohan, age=20, sex=M, income=2000
```

Python, DB

- UPDATE operacija odnosi se na ažuriranje jednog ili više zapisa u tabelama baze podataka. Npr. ažuriraju se svi zapisi koji imaju atribut SEX postavljen na vrednost 'M' tako da im se poveća atribut AGE za 1.

```
#!/usr/bin/python
import pymysql
db = pymysql.connect(host="localhost",port=3306,user="testuser",
        passwd="test123",db="TESTDB")
cursor = db.cursor()
# Prepare SQL query to UPDATE required records
sql = "UPDATE EMPLOYEE SET AGE = AGE + 1 WHERE SEX = '%c'" % ('M')
try:
   cursor.execute(sql)
   db.commit()
except:
   db.rollback()
db.close()
```

- DELETE operacija se koristi za brisanje zapisa:
- U primeru koji sledi brišu se zapisi zaposlenih starijih od 20 godina:

```
#!/usr/bin/python
import pymysql
db = pymysql.connect (host="localhost",port=3306,user="testuser",
        passwd="test123",db="TESTDB")
```

Python, DB

```
cursor = db.cursor()
# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
try:
   cursor.execute(sql)
   db.commit()
except:
   db.rollback()
db.close() #sve preostale nezavršene transakcije ce biti opozvane
```

- Transakcija predstavlja mehanizam koji obezbeđuje konzistentnost podataka i ima sledeće osobine:
  - **Atomicity (nedeljivost):** ili se cela transakcija kompletira ili ništa.
  - **Consistency (konzistentnost):** transakcija mora početi u konzistentnom stanju i mora ostaviti sistem u konzistentnom stanju.
  - **Isolation (izolacija):** međurezultati transakcije nisu vidljivi izvan tekuće transakcije.
  - **Durability (trajnost):** kada je transakcija jednom potvrđena (committed) efekti su trajni čak i nakon sistemskog otkaza.

Python, DB

- Python DB API 2.0 omogućuje dve metode ili potvrde (*commit)* ili opoziva *(rollback)* transakcije.

```
# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Commit your changes in the database
   db.commit()
except:
   # Rollback in case there is any error
   db.rollback()
```

- COMMIT operacija daje zeleno svetlo bazi podataka da finalizira promene i nakon toga se ne može vratiti prethodno stanje.
- ROLLBACK operacija služi za opoziv transakcije čime će se baza podataka vratiti u idenično stanje kao pre opozvane transakcije.
  - npr. ako je došlo do greške u izvršavanju transakcije

MREŽNO PROGRAMIRANJE

- Python omogućuje dva nivoa pristupa mrežnim servisima.
- Na niskom nivou koriste se socket-i čime se mogu realizovati klijent-server arhitektura za konekciono orijentisane (connection-oriented) i beskonekcione (connectionless) protokole.
- Python ima biblioteke koje omogućuju visoki nivo pristupa ka specifičnim mrežnim protokolima na aplikacionom nivou (FTP, HTTP, ...).
- Socket-i su krajnje tačke bidirekcionalnog komunikacionog kanala.
- Socket-i mogu komunicirati unutar procesa, između procesa na istoj mašini ili između procesa na različitim mašinama (kontinentima).
- Socket-i mogu biti implementirni nad različitim tipovima kanala (npr. TCP, UDP).

Python: mrežno programiranje

Termin	Opis
`domain (family)`	Porodica protokola koji se koriste za transportni mehanizam (konstante: AF_INET, PF_INET, PF_UNIX, PF_X25, ...).
`type`	Tip komunikacije između dve krajnje tačke (SOCK_STREAM za connection-oriented, SOCK_DGRAM za connectionless).
`protocol`	Tipično 0 (za identifikaciju varijante protokola).
`hostname`	Identifikator mrežnog interfejsa (string koji je naziv host-a, IP4 adresa sa tačka delimiterima 4 bajta, ili IPV6 adresa. String "<broadcast>", specificira INADDR_BROADCAST adresu. Zero-length string koji specificira INADDR_ANY adresu. Ceo broj interpretiran kao binarna adresa.
`port`	Svaki server osluškuje klijentske pozive na jednom ili više portova. Port može biti: broj, string koji sadrži broj porta ili naziv servisa.

Python: mrežno programiranje

- Za kreiranje socket-a koristi se modul socket:

  `s = socket.socket (socket_family, socket_type, protocol=0)`

  parametri su kao što sledi:

  socket_family: vrednosti su: ili AF_UNIX ili AF_INET.

  socket_type: vrednosti su: ili je SOCK_STREAM ili SOCK_DGRAM.

  protocol: Obično ostavljeno podrazumevano 0.

Server socket metodi:

Metod	Opis
s.bind()	Veže adresu (uređen par: hostname, port) i socket.
s.listen()	Postavlja i startuje TCP osluškivač.
s.accept()	Pasivno prihvatanje TCP klijentske konekcije (sinhrona)

Client socket metodi:

Metod	Opis
s.connect()	Inicira TCP konekciju sa serverom.

Python: mrežno programiranje

- Generalne metode socket-a:

Metod	Opis
s.recv()	Prima TCP poruke.
s.send()	Šalje TCP poruke.
s.recvfrom()	Prima UDP poruke.
s.sendto()	Šalje UDP poruke.
s.close()	Zatvara socket.
socket.gethostname()	Vraća naziv hosta (hostname).

- Kreiranje servera: koristi se funkcija socket modula socket za kreiranje socket-a.
- Kada je socket kreiran koriste se funkcije socketa da bi se postavio socket server.
- Sada se poziva bind((hostname, port)) metoda socket-a da bi se specificirao port na hostu.
- Nakon ovoga poziva se metoda *accept ()*
  - Ova metoda je sinhrona, odnosno, čeka dok se klijent ne konektuje na specificirani port, a onda vraća konketovani objekat koji predstavlja vezu ka klijentu.

Python: mrežno programiranje

- Sledi program koji radi na strani servera u socket komunikaciji:

```
#!/usr/bin/python                    # This is server.py file
import socket                        # Import socket module
s = socket.socket()                  # Create a socket object
host = socket.gethostname()          # Get local machine name
port = 12345                         # Reserve a port for your service.
s.bind((host, port))                 # Bind to the port
s.listen(5)                          # Now wait for client connection.
while True:
   conn, addr = s.accept()           # Establish connection with client.
   print ('Got connection from', addr)
   poruka = 'Thank you for connecting'
   conn.send(poruka.encode())
   conn.close()                      # Close the connection
```

- Klijent sada ima zadatak da otvori konekciju ka datom portu 12345 i datom host-u (da inicira komunikaciju).
  - Koristi se socket.connect((hostname, port)) metoda da bi se otvorila TCP konekcija ka navedenom hostu i portu.
  - Kada je socket otvoren, potrebno je pročitati i ispisati poruku koju šalje server a potom zatvoriti socket.

Python: mrežno programiranje

```
#!/usr/bin/python                    # This is client.py file
import socket                        # Import socket module
s = socket.socket()                  # Create a socket object
host = socket.gethostname()          # Get local machine name
port = 12345                         # Reserve a port for your service.
s.connect((host, port))
print (s.recv(1024).decode())
s.close()                            # Close the socket when done
```

- Sada je potrebno pokrenuti server u pozadini a onda pokrenuti klijent i videti rezultat:

```
# 1. start a server in background:
$ python server.py
# 2. run client:
$ python client.py
```

```
•                        Got connection from ('127.0.0.1', 3209)  #server
                         Thank you for connecting                 #client
```

Python Internet moduli

- Standardni protokoli imaju svoje predviđene rezervisane portove.
- Sledi lista standardnih protokola, pripadnih portova i odgovarajućih Python modula za navedene protokole.

Python: mrežno programiranje

Protokol	Namena	Port	Python modul
HTTP	Web pages	80	httplib, urllib, xmlrpclib
NNTP	Usenet news	119	nntplib
FTP	File transfers	20	ftplib, urllib
SMTP	Sending email	25	smtplib
POP3	Fetching email	110	poplib
IMAP4	Fetching email	143	imaplib
Telnet	Command lines	23	telnetlib
Gopher	Document transfers	70	gopherlib, urllib

- Još jedan primer komunikacije socket-ima:

```
import socket    # Echo server program
HOST = ''        # Symbolic name meaning all available interfaces
PORT = 50007
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
while True:
   conn, addr = s.accept()
```

Python: mrežno programiranje

```
 print ('Connected by', addr)
   data = conn.recv(1024)
   if not data: break
   conn.send(data)
conn.close()

# Echo client program
import socket
HOST = '127.0.0.1'       # The remote host
PORT = 50007             # The same port as used by the server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
s.send(b'Hello, world') # razmislite o s.send('Hello, world'.encode())
data = s.recv(1024)     # razmislite o data = s.recv(1024).decode()
s.close()
print ('Received', (data))
```

- U prethodnom programu klijent se konektuje na server kome šalje poruku.
- Server osluškuje port i kada se klijent javi uspostavi se veza sa klijentom od koga server prima poruku a onda tu istu poruku vraća klijentu (eho server).

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.
"""
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.
- Za eksperimente koristiti program fakeSMTP.

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 jedinstvenim markerom, a završava 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.
  - threading.activeCount() : vraća broj aktivnih niti (ili active_count).
  - threading.enumerate() : enumeracija niti.
- Threading modul ima klasu Thread čije su neke 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()` i koristi se 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: 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)
fp = open('jsonpodaci.txt','w')
json.dump(dictUser,fp)
fp.close()
objUser = User.from_json(dictUser);
print(objUser);

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

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

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