

FUNKCIONALNO PROGRAMIRANJE

Oznaka predmeta: FPR

Predavanje broj: 07

Nastavna jedinica: PYTHON,

Nastavne teme:

Cookie (snimanje i uzimanje vrednosti). Baze podataka (konekcija, kreiranje baze, upiti, uzimanje rezultata upita, kreiranje, dodavanje, menjanje i brisanje zapisa). MySQLdb. sqlite3. Mrežno programiranje (socket, klijent-server arhitektura).

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 cgi
cgi.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 informacija o cookie-ima šalje uz HTTP zaglavlje pre *Content-type* polja.
- Postavljaju se tri cookie-a (1 pomoću modula `http.cookies` i 2 bez njega).

```
#!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 Mozilla
cookie["session"]["path"] = "/cgi-bin/" #kod Mozilla
cookie["session"]["expires"] = \
expiration.strftime("%a, %d-%b-%Y %H:%M:%S GMT")
print ("Set-Cookie: Elvis=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', ' Elvis=Presley', ' Ford=Mustang']`
 session --> 948797479
 Elvis --> Presley
 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 Elvis 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 ("Elvis = " + cookie["Elvis"].value)
```

```
except (http.cookies.CookieError, KeyError):
```

```
    print ("Elvis cookie not set!")
```

Elvis = Presley

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())
# visetruko izvršavanje
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 konekcije 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 konektovanje 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 pymysql (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 fetch data")
db.close()

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

Python, DB

- UPDATE operacija se odnosi 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 ne može se 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 beskonekcionne (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 implementirani 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_INET ili AF_UNIX.

socket_type: vrednosti su: ili je SOCK_STREAM ili SOCK_DGRAM.

protocol: Obično ostavljeno podrazumevano 0.

Server socket metodi:

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

Client socket metodi:

Metod	Opis
<code>s.connect()</code>	Inicira TCP konekciju sa serverom.

Python: mrežno programiranje

- Generalne metode socket-a:

Metod	Opis
<code>s.recv()</code>	Prima TCP poruke.
<code>s.send()</code>	Šalje TCP poruke.
<code>s.recvfrom()</code>	Prima UDP poruke.
<code>s.sendto()</code>	Šalje UDP poruke.
<code>s.close()</code>	Zatvara socket.
<code>socket.gethostname()</code>	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
import socket
s = socket.socket()
host = socket.gethostname()
port = 12345
s.connect((host, port))
print (s.recv(1024).decode())
s.close()
```

This is client.py file
Import socket module
Create a socket object
Get local machine name
Reserve a port for your service.
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	smtpplib
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:

Echo server program

```
import socket
```

```
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)
```

```
conn, addr = s.accept()
```

Python: mrežno programiranje

```
print ('Connected by', addr)
while True:
    data = conn.recv(1024)
    if not data: break
    conn.send(data)
conn.close() # izmenite kod eho servera
```

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')
data = s.recv(1024)
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).