Scientific Programming – Network programming in Python

Exercises:

1) The first Python server presented in the first lecture on network programming does actually not do anything else but to thank the client for the connection. Implement this simple server, plus the simple client from the lecture, and test that they are working. For testing purpose, you can do this all on your local host (no need for an internet connection).

Server ("12-Network-simple-server-lecture.py"):

```
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)
   conn.sendall(b'Thank you for connecting')
   conn.close()                                 # Close the connection
```

Client ("12-Network-simple-client-lecture.py"):

```
import socket                                   # Import socket module

s = socket.socket()                             # Create a socket object
host = socket.gethostname()                     # Get local machine name
port = 12345                                    # Port of the server to be contacted

s.connect((host, port))                         # Establish a connection to the server

print(s.recv(1024))                             # Receive the response and print it
s.close()                                       # Close the socket when done
```

Testing (run from the command line):

```
python 12-Network-simple-server-lecture.py &
# [3] 8995

python 12-Network-simple-client-lecture.py
# ('Got connection from', ('127.0.0.1', 47524))
# Thank you for connecting

python 12-Network-simple-client-lecture.py
# ('Got connection from', ('127.0.0.1', 47526))
# Thank you for connecting

jobs
# [...]
# [3]+  Running                 python 12-Network-simple-server-lecture.py &
```

```
kill %3
# [3]+  Terminated              python 12-Network-simple-server-lecture.py
```

Note: "`kill %3`" terminates the third job of this shell session (see list with "`jobs`"). Alternatively you can terminate the job using it's process ID (see solution of the next exercise as an example).

2) Let's make this a little bit more complicated: Write a server whose response depends on a specific request of the client:

- If the client sends the request string "time", the server sends as a response its own local time.
- If the client sends the string "nuqneH?", the server responds with "Qapla'!"
- If the client sends anything else, the server works like an echo server, it simply sends the received data back to the client.

The client itself should print out what response it received from the server. Test the different answers from the server.
[Note: "nuqneH?" ... "Qapla'!" is actually a complete conversation in Klingon :-)]

Server ("12-Network-simple-server.py"): [major changes in red]

```
import socket                                   # Import socket module

import time                                     # for sleep (wait)
from datetime import datetime                   # For local time

# We need a function to safely receive the client request!
End=b'\0'
def recv_end(conn):
    total_data=[]; data=''
    while True:
        data=conn.recv(1024)
        if End in data:
            total_data.append(data[:data.find(End)])
            break
        total_data.append(data)
    return b''.join(total_data)

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)

   data = recv_end(conn)

   print('Received request: ', data)

   time.sleep(5)

   if data == b'time':
       now = datetime.now()
       current_time = now.strftime("%H:%M:%S")
       conn.sendall(current_time.encode('utf-8') + b'\0')
```

```
   elif data == b'nuqneH?':
       conn.sendall(b'Qapla\'!\0')

   else:
       conn.sendall(data + b'\0')

   conn.close()                                # Close the connection
```

Client ("12-Network-simple-client.py"): [major changes in red]

```
import socket                                  # Import socket module
import sys                                     # For command line arguments

End=b'\0'
def recv_end(conn):
    total_data=[]; data=''
    while True:
        data=conn.recv(1024)
        if End in data:
            total_data.append(data[:data.find(End)])
            break
        total_data.append(data)
    return b''.join(total_data)

message = b''

n = len(sys.argv)
if (n == 1):
    print("No message to server provided on the command line; ending program\n")
    exit(0)
else:
    for i in range(1, n):
        message += sys.argv[i].encode('UTF-8')
        if (i != n-1):
            message += b' '

message += End

print ("Sending message to server: " + message.decode('UTF-8') + "\n")

s = socket.socket()                            # Create a socket object
host = socket.gethostname()                    # Get local machine name
port = 12345                                   # Port of the server to be contacted

s.connect((host, port))                        # Establish a connection to the server

s.sendall(message)

print(recv_end(s))                             # Receive the response and print it
s.close()                                      # Close the socket when done
```

Testing:

```
python 12-Network-simple-server.py &
# [4] 9749

python 12-Network-simple-client.py nuqneH?
# Sending message to server: nuqneH?
```

```
#
# ('Got connection from', ('127.0.0.1', 49424))
# ('Received request: ', 'nuqneH?')
# Qapla'!

python 12-Network-simple-client.py time
# Sending message to server: time
#
# ('Got connection from', ('127.0.0.1', 49554))
# ('Received request: ', 'time')
# 11:19:13

python 12-Network-simple-client.py time is precious
# Sending message to server: time is precious
#
# ('Got connection from', ('127.0.0.1', 49280))
# ('Received request: ', 'time is precious')
# time is precious

kill 9749
# [4]+  Terminated              python 12-Network-simple-server.py
```

3) Implement both the servers and the clients from exercise 1 and 2 in such a way, that they know when they have received the full message from the remote connection.

Already shown in the examples above (using `b'\0'` as termination character!)

4) Implement the server from exercise 1 in such a way that it can handle multiple requests at the same time! Test it by contacting it by three clients at the same time (one asking for the time, one speaking Klingon and one just saying “Hello world!” or something else).

Server (“12-Network-simple-server-multiproc.py”): [major changes in red]

```
import socket                                   # Import socket module

import multiprocessing, os                      # For multiprocessing and process ID
import time                                     # for sleep (wait)
from datetime import datetime                   # For local time

# We need a function to safely receive the client request!
End=b'\0'
def recv_end(conn):
    total_data=[]; data=''
    while True:
        data=conn.recv(1024)
        if End in data:
            total_data.append(data[:data.find(End)])
            break
        total_data.append(data)
    return b''.join(total_data)

def handleRequest(conn, addr):
   data = recv_end(conn)
   print('Process', os.getpid(), 'handling request from', addr, ':', data)
   time.sleep(10)

   if data == b'time':
```

```
        now = datetime.now()
        current_time = now.strftime("%H:%M:%S")
        conn.sendall(current_time.encode('utf-8') + b'\0')

    elif data == b'nuqneH?':
        conn.sendall(b'Qapla\'!\0')

    else:
        conn.sendall(data + b'\0')

    conn.close()

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('Process', os.getpid(),'got connection from', addr)
    process = multiprocessing.Process(target=handleRequest, args=(conn, addr))
    process.daemon = True
    process.start()
```

Testing (using and switching between 3 open shell sessions!):

```
python 12-Network-simple-server-multiproc.py &
# [5] 10960

# in shell session 1:
python 12-Network-simple-client.py time
Sending message to server: time

('Process', 10960, 'got connection from', ('127.0.0.1', 50544))
('Process', 11049, 'handling request from', ('127.0.0.1', 50544), ':', 'time')
('Process', 10960, 'got connection from', ('127.0.0.1', 50546))
('Process', 11051, 'handling request from', ('127.0.0.1', 50546), ':', 'nuqneH?')
('Process', 10960, 'got connection from', ('127.0.0.1', 50548))
('Process', 11053, 'handling request from', ('127.0.0.1', 50548), ':', 'Hello
World!')
11:45:58

# in shell session 2:
python 12-Network-simple-client.py nuqneH?
Sending message to server: nuqneH?

Qapla'!

# in shell session 3:
python 12-Network-simple-client.py Hello World!
Sending message to server: Hello World!

Hello World!

# in shell session 1:
kill 10960
# [5]+  Terminated              python 12-Network-simple-server-multiproc.py
```

Note: you can see that the server was running in the first shell session (that's why the server side print commands ended up there)! You can also see (from the process IDs) that the three requests where not handled by the server process, but by new processes spawned for handling the individual requests!

5) [Optional; results not shown here] Using a termination character (e.g., '\0') for letting the receiver know when a message ends has an important drawback: *The message itself must not contain any termination character*, otherwise the rest of the message will be ignored by the receiver. If the message contains some arbitrary, user-defined text, then it is in principle possible that the message itself does somewhere contain the termination character.
An alternative (and maybe better) approach would be: instead of *ending* the message with a termination character, it can be *started* with an indication of how long the message is (e.g., number of bytes for generic data or number of characters in text messages).
Example: instead of

"Hello world!" (12 characters)

the sender might actually transmit something like

"12;Hello world!"

to indicate that the message itself (after the semicolon) has 12 characters.

Try to implement this approach (defining a generic function which takes a message and adds its length at the beginning); both the client and the server need to be able to send such messages, receive then and be able to verify that they got the entire message.

6) RESTful web service: Implement the web service presented in the lectures

a) First create a simple web server (Hallo World!) and test it with your web browser at:
http://localhost:5000/

Server ("12-Network-flask-simple-webservice.py"):

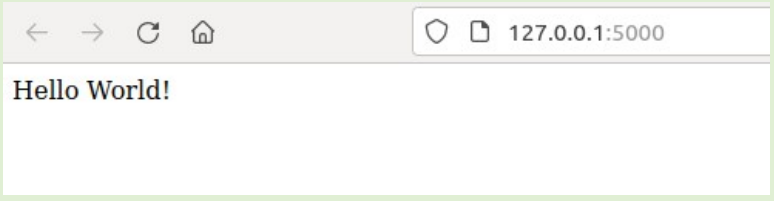
```
from flask import Flask, jsonify, request

app = Flask(__name__)

geneDB=[
  {
    'id':'1',
    'symbol':'A1BG',
    'name':'alpha-1-B glycoprotein'
  },
  {
    'id':'2',
    'symbol':'A2M',
    'name':'alpha-2-macroglobulin'
  }
]

@app.route("/")
def hello():
    return "Hello World!"

@app.route('/genedb/gene',methods=['GET'])
```

```
def getAllGenes():
    return jsonify({'genes':geneDB})

@app.route('/genedb/gene/<geneId>',methods=['GET'])
def getGene(geneId):
    usr = [ gene for gene in geneDB if (gene['id'] == geneId) ]
    return jsonify({'gene':usr})

@app.route('/genedb/gene/<geneId>',methods=['PUT'])
def updateGene(geneId):
    g = [ gene for gene in geneDB if (gene['id'] == geneId) ]
    if 'symbol' in request.json :
        g[0]['symbol'] = request.json['symbol']
    if 'name' in request.json:
        g[0]['name'] = request.json['name']
    return jsonify({'gene':g[0]})

@app.route('/genedb/gene',methods=['POST'])
def createGene():
    dat = {
        'id':request.json['id'],
        'symbol':request.json['symbol'],
        'name':request.json['name']
    }
    geneDB.append(dat)
    return jsonify(dat)

@app.route('/genedb/gene/<geneId>',methods=['DELETE'])
def deleteGene(geneId):
    g = [ gene for gene in geneDB if (gene['id'] == geneId) ]
    if len(g) == 0:
        abort(404)
    geneDB.remove(g[0])
    return jsonify({'response':'Success'})


if __name__ == "__main__":
    app.run()
```

Running the webservice (from the shell):

```
python 12-Network-flask-simple-webservice.py &
# [5] 11717
#  * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Checking with a web browser:



b) Test also it's status with CURL:

```
curl -i http://localhost:5000/
```

```
curl -i http://localhost:5000/
# HTTP/1.0 200 OK
# Content-Type: application/json
# Content-Length: 12
# Server: Werkzeug/0.14.1 Python/2.7.17
# Date: Fri, 14 May 2021 10:07:08 GMT
#
# Hello World!
```

c) Implement the rest of the web service for managing gene information. Test all functions and check the status (content) of the entire gene database after each update.

A) Getting info for all genes:

```
curl -i http://localhost:5000/genedb/gene
# HTTP/1.0 200 OK
# Content-Type: application/json
# Content-Length: 128
# Server: Werkzeug/0.14.1 Python/2.7.17
# Date: Fri, 14 May 2021 10:10:36 GMT
#
# {"genes":[{"id":"1","name":"alpha-1-B glycoprotein","symbol":"A1BG"},
# {"id":"2","name":"alpha-2-macroglobulin","symbol":"A2M"}]}
```

B) Get info for a single gene:

```
curl -i http://localhost:5000/genedb/gene/1
# HTTP/1.0 200 OK
# Content-Type: application/json
# Content-Length: 70
# Server: Werkzeug/0.14.1 Python/2.7.17
# Date: Fri, 14 May 2021 10:21:08 GMT
#
# {"gene":[{"id":"1","name":"alpha-1-B glycoprotein","symbol":"A1BG"}]}
```

C) Change/update info for a gene:

```
curl -i -H "Content-type: application/json" -X PUT -d "{\"symbol\":\"HYST2477\"}" \
      http://localhost:5000/genedb/gene/1
# HTTP/1.0 200 OK
# Content-Type: application/json
# Content-Length: 72
# Server: Werkzeug/0.14.1 Python/2.7.17
# Date: Fri, 14 May 2021 10:23:10 GMT
#
# {"gene":{"id":"1","name":"alpha-1-B glycoprotein","symbol":"HYST2477"}}
```

Check how the full DB looks like now:

```
curl -i http://localhost:5000/genedb/gene
# […]
# {"genes":[{"id":"1","name":"alpha-1-B glycoprotein","symbol":"HYST2477"},
# {"id":"2","name":"alpha-2-macroglobulin","symbol":"A2M"}]}
```

d) Add a new gene:

```
curl -i -H "Content-type: application/json" -X POST \
      -d "{\"id\":\"3\", \"symbol\":\"A2MP1\", \
      \"name\":\"alpha-2-macroglobulin pseudogene 1\"}" \
      http://localhost:5000/genedb/gene
# HTTP/1.0 200 OK
# Content-Type: application/json
# Content-Length: 72
# Server: Werkzeug/0.14.1 Python/2.7.17
# Date: Fri, 14 May 2021 11:25:35 GMT
#
# {"id":"3","name":"alpha-2-macroglobulin pseudogene 1","symbol":"A2MP1"}
```

Check how the full DB looks like now:

```
curl -i http://localhost:5000/genedb/gene
# […]
# {"genes":[{"id":"1","name":"alpha-1-B glycoprotein","symbol":"HYST2477"},
# {"id":"2","name":"alpha-2-macroglobulin","symbol":"A2M"},
# {"id":"3","name":"alpha-2-macroglobulin pseudogene 1","symbol":"A2MP1"}]}
```

E) Delete a gene from the DB:

```
curl -i -X DELETE http://localhost:5000/genedb/gene/2
# HTTP/1.0 200 OK
# Content-Type: application/json
# Content-Length: 23
# Server: Werkzeug/0.14.1 Python/2.7.17
# Date: Fri, 14 May 2021 11:27:09 GMT
#
# {"response":"Success"}
```

Check how the full DB looks like now:

```
curl -i http://localhost:5000/genedb/gene
# […]
# {"genes":[{"id":"1","name":"alpha-1-B glycoprotein","symbol":"HYST2477"},
# {"id":"3","name":"alpha-2-macroglobulin pseudogene 1","symbol":"A2MP1"}]}
```

d) Implement an additional function which empties the database in one go (deletes all gene entries at once) and test it.

Add to the server:

```
@app.route('/genedb/gene',methods=['DELETE'])
def deleteAllGenes():
    geneDB.clear()
    return jsonify({'response':'Success'})
```

Testing:

```
curl -i -X DELETE http://localhost:5000/genedb/gene
# HTTP/1.0 200 OK
# Content-Type: application/json
# Content-Length: 23
```

```
# Server: Werkzeug/0.14.1 Python/3.6.9
# Date: Fri, 14 May 2021 11:38:29 GMT
#
# {"response":"Success"}
```

Checking:

```
curl -i http://localhost:5000/genedb/gene
# HTTP/1.0 200 OK
# Content-Type: application/json
# Content-Length: 13
# Server: Werkzeug/0.14.1 Python/3.6.9
# Date: Fri, 14 May 2021 11:38:50 GMT
#
# {"genes":[]}
```

Note: this works only if the server is running with Python 3 because in Python 2 there is no function "`clear()`" for lists. For Python 2, instead of "`geneDB.clear()`" you can use "`del geneDB[:]`".

e) Finally, replace the "Hello World" function by a function which, for the same end point (GET and same URL) provides a list all available routes (URLs defined in `@app.route`).
[Hint: try `app.url_map.iter_rules()`]

Make the following change to the server (replace the function for "/"):

```
#@app.route("/")
#def hello():
#    return "Hello World!"

@app.route("/")
def listRoutes():
    routes = []
    for rule in app.url_map.iter_rules():
        routes.append('%s' % rule)
        routes = list(set(routes))
    return jsonify(routes)
```

Test it:

```
curl -i http://localhost:5000/
# HTTP/1.0 200 OK
# Content-Type: application/json
# Content-Length: 71
# Server: Werkzeug/0.14.1 Python/3.6.9
# Date: Fri, 14 May 2021 11:46:03 GMT
#
# ["/static/<path:filename>","/genedb/gene/<geneId>","/genedb/gene","/"]
```