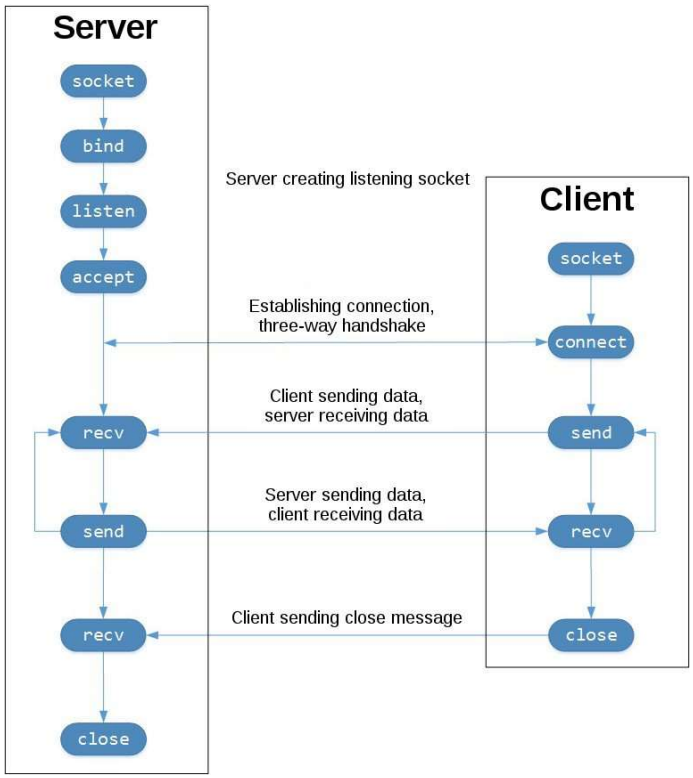
Practical Course "Web Technologies"

Exercises 1

Exercise 1.1.

Implement a simple client-server-communication in Python, in which the server (address: 127.0.0.1) will accept via port 5001 a connection of a single client, receive messages from the client, capitalize the message, and send it back to the client.

Calling sequence:



Class methods for the Socket module:

Class method	Description
`Socket`	Low-level networking interface (per the BSD API)
`socket.socket(family, type)`	Create and return a new socket object
`socket.getfqdn(name)`	Convert a string quad dotted IP address to a fully qualified domain name
`socket.gethostbyname(hostname)`	Resolve a hostname to a string quad dotted IP address
`socket.fromfd(fd, family, type)`	Create a socket object from an existing file descriptor

Instance methods for the Socket module:

Instance method	Description
`sock.bind( (adrs, port) )`	Bind the socket to the address and port
`sock.accept()`	Return a client socket (with peer address information)
`sock.listen(backlog)`	Place the socket into the listening state, able to pend *backlog* outstanding connection requests
`sock.connect( (adrs, port) )`	Connect the socket to the defined host and port
`sock.recv( buflen[, flags] )`	Receive data from the socket, up to `buflen` bytes
`sock.recvfrom( buflen[, flags] )`	Receive data from the socket, up to `buflen` bytes, returning also the remote host and port from which the data came
`sock.send( data[, flags] )`	Send the data through the socket
`sock.sendto( data[, flags], addr )`	Send the data through the socket
`sock.close()`	Close the socket

Exercise 1.2.

Implement a simple chat-room in Python, in which the server will accept multiple clients' connections, receive messages from clients, and broadcast them to the other clients

1. the server
   - import socket and threading python libraries
   - define the IP address and port on which the server will listen, and create a socket object
   - define a dictionary, that take the client's identification as key as its connection as value
   - create a function `broadcast(msg, cid)` that takes the message in the parameter and sends it to all connected clients except the client with the identification `cid`
   - create a function `handle_clients(conn, cid)`

     In this function,

     1. send a welcome-message to the client
     2. save the client's connection `conn` in the dictionary using the client identification as `key`
     3. create an information message about the newly added client and broadcasts the message to all currently connected clients

In last, read messages from the client in a loop and broadcast the message.

- create a function `accept_client_connection()`

  In this function accept a client requests using `sock.accept()` and print out a corresponding information (as server log). After that, send a greeting-message to the client and ask the client for its identification. If the client's identification is not empty, print out a log-message and start the `handle_clients(conn)` function in a thread.

  A thread is created by instantiating an instance of the Thread class:

  ```
  new_thread = threading.Thread(target=fn, args=args_tuple)
  ```

  `Thread`() accepts many parameters. The main ones are:

  `target`: specifies a function `fn` to run in the new thread

  `args`: specifies the arguments of the function `fn`.

  The `args` argument must be a tuple!

  Third, start the thread by calling the `start`() method of the Thread instance.

- in `main` resp as main-statements make the server listen for client requests and start the accept function `accept_client_connection()` in a thread for handling multiple requests at once.

2. the client
   - import socket and threading python libraries
   - define the IP address and port on which the server will listen
   - ask for the client's identification
   - create a socket on client side and connect to the server
   - receive the identification request from the server and return the identification
   - accepts a welcome message
   - start the function `receive()` in a thread for handling messages from server in a loop
   - enter a loop in which user is asked for a message and the message is send to the server