Socket Programming.

server needs a socket address (ip + port).

↓

host

why localhost ? because client & server both are running locally.

127.0.0.1 ↔ localhost means you are trying to access a process on the same machine.

python gives us a library called socket.

a plain socket object needs to be bound to the host & port to create a TCP connection.

if your server and client are on different machine you have to be on the same network.

1. create socket object on server
2. bind to ip & port
3. start listening for connections.

```python
while True:
    # wait for new connection acceptance
    server_socket.accept()
```

```python
import socket

HOST = "localhost"

PORT = 3000

# created a new socket object
server_socket = socket.socket()

# bind the socket object to the host and port

server_socket.bind((HOST, PORT))

# start listening for new connection
server_socket.listen()

print("server is listening on", HOST, PORT)
```

our process ends because we have to wait for new connection acceptance. we do that via an infinite loop. & our server is alive.

Since python supports multithreading. We can use a library called threading. We can create a new thread & on this thread we want to run a func^n.

server.py > ...

```python
import socket
import threading


def connect_a_client(conn, addr):
    print("New client has been connected")


HOST = "localhost"

PORT = 3000

# created a new socket object
server_socket = socket.socket()

# bind the socket object to the host and port

server_socket.bind((HOST, PORT))

# start listening for new connection
server_socket.listen()

print("server is listening on", HOST, PORT)

while True:
    conn, addr = server_socket.accept()
    t = threading.Thread()
```

accepting a new connection

```python
while True:
    conn, addr = server_socket.accept()
    t = threading.Thread(target=connect_a_client, args=(conn, addr))
    t.start() # it starts running the thread
```

multi threading

preparing

Client.

Welcome | server.py | client.py

client.py > ...

```python
import socket
print("Starting a client : Client 1")

HOST = "localhost"
PORT = 3000 # this is the port for the server to connect

client_socket = socket.socket()

client_socket.connect((HOST, PORT)) # client needs to connect to the server socket

client_socket.sendall(b"Hello from the client 1")

```

```python
def connect_a_client(conn, addr):
    print("New client has been connected")
    data = conn.recv(2048)
    print("Data recevied from client is:", data)
```

client socket. connect is where the 3 way handshake is done.

now server wants to send some data to client

Client.

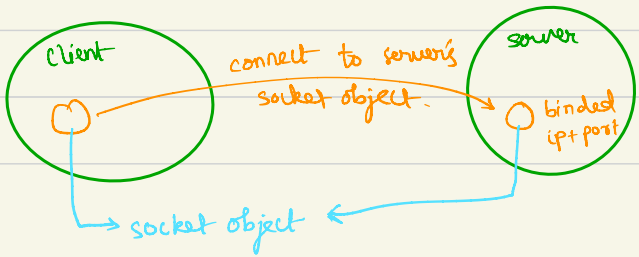
client.py

client.py > ...

```python
import socket
print("Starting a client : Client 1")

HOST = "localhost"
PORT = 3000 # this is the port for the server to

client_socket = socket.socket()

client_socket.connect((HOST, PORT)) # client need

client_socket.sendall(b"Hello from the client 1")


response_from_server = client_socket.recv(2048)

print(response_from_server)
```

```python
def connect_a_client(conn, addr):
    print("New client has been connected")
    data = conn.recv(2048)
    print("Data recevied from client is:", data)
    conn.sendall(b"Server has received your data thanks")
```

Server.



After the client process ends that connection is terminated.

client doesn't need dedicated port until any other client wants to connect to it.

pure client needs to have a process, port not necessary.

---

in nodejs net module automatically picks up the machine's ip address.

nodeserver.js

```
const net = require('net');

const server = net.createServer((socket) => {
    socket.on('data', (clientData) => {
        console.log("Data recevied from client", clientData);
        socket.write("received on server thank you");
    });
});

server.listen(8080, () => {
    console.log("New server up on port 8080");
});
```

→ binding + listening happens in same fn + accept.

Client

nodeclient.js

```
const net = require('net');

const client = net.createConnection({port: 8080}, () => {
    console.log("Client connected");
    client.write("hello from node client");
});

client.on('data', (serverData) => {
    console.log("Data received from server is", serverData.toString());
})
```

These were examples of TCP servers.

To create http servers in nodeJS you can use http module. Socket programming is abstracted by the library.

http.createServer() returns a new server object

→ in cb you have access to incoming req & outgoing response

→ when this server is called by client the cb is executed.

http runs on application layer
TCP runs on transport layer.

http relies on TCP connection. NO TCP, NO Http.

```
http-server-node.js > server.listen() callback
1   const http = require('http');
2
3   const server = http.createServer((req, res) => {
4       console.log("New connection was created");
5   });
6
7
8   server.listen(3000, () => {
9       console.log("server started at port 3000")
10  })
```

by default node gives http 1.1.

→ Since http is app layer, that is why we can view this in the browser.

Even on different route you get same response.

```
http-server-node.js > server > http.createServer() callback
1   const http = require('http');
2
3   const server = http.createServer((req, response) => {
4       console.log("New connection was created");
5       if(req.url === '/home') {
6           return response.end('welcome home')
7       } else {
8           return response.end("Something something !!!!")
9       }
10  });
11
12
13  server.listen(3000, () => {
14      console.log("server started at port 3000")
15  });
```

→ routing.

express app.listen.

```
app.listen = function listen() {
  var server = http.createServer(this);
  return server.listen.apply(server, arguments);
};
```

https://github.com/singhsanket143/SocketProgrammingDemo