
Using TCP socket, implement HTTP server and client

▼ Creating HTTP SERVER

▼ 1. Importing Socket Library

```
import socket
from socket import AF_INET, SOCK_STREAM, SO_REUSEADDR, SOL_SOCKET
#SO_REUSEADDR is to reuse the socket
```

▼ 2. Create HTTP Server with response message in http format

```
HOST, PORT = 'localhost', 8081
response = b"HTTP/1.1 200 OK \n\nhello buddy welcome to server"
```

▼ 3. Create TCP Server Socket

```
server_socket = socket.socket(AF_INET, SOCK_STREAM)
#setsockopt makes socket to reuse address to read user
server_socket.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
# Binding the TCP server
server_socket.bind((HOST, PORT))
# Server can listen to max of 1 Client
server_socket.listen(1)
```

▼ 4. Server acts like a HTTP Server

HTTP is a stateless protocol

Tcp is a connection oriented protocol

```
#here we have connect many times to the server
#because server closes connection with client after responding to client
while True:
    try:
        client_socket, addr = server_socket.accept()
        print(client_socket.recv(1024).decode('utf-8'))
        client_socket.sendall(response)
        client_socket.close()
    except Exception as e:
        . . .
```

```
print(e)
socketserver.close()
```

▼ Creating HTTP client

▼ 1. Creating client socket to connect to the server

```
client_socket = socket.socket(AF_INET, SOCK_STREAM)
HOST, PORT = 'localhost', 8081 #connecting to host on port
```

▼ 2. Requesting server to connect using Http request

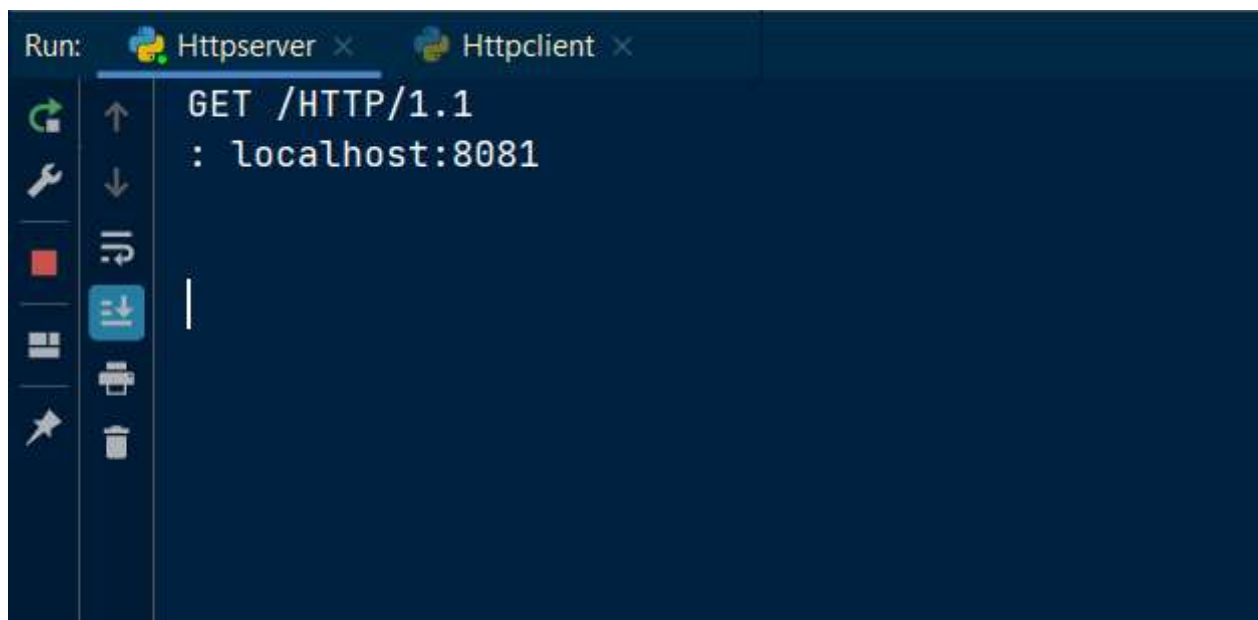
```
request = f"GET /HTTP/1.1\r\n: {HOST}:{PORT}\r\n\r\n".encode('utf8')
client_socket.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
client_socket.connect((HOST, PORT))
client_socket.sendall(request)
```

▼ 3. After getting the server response printing

```
response = ""
while True:
    data = client_socket.recv(1024)
    if data == b'':
        break
    print(data.decode())
client_socket.close()
```

▼ Output(Screen shots)

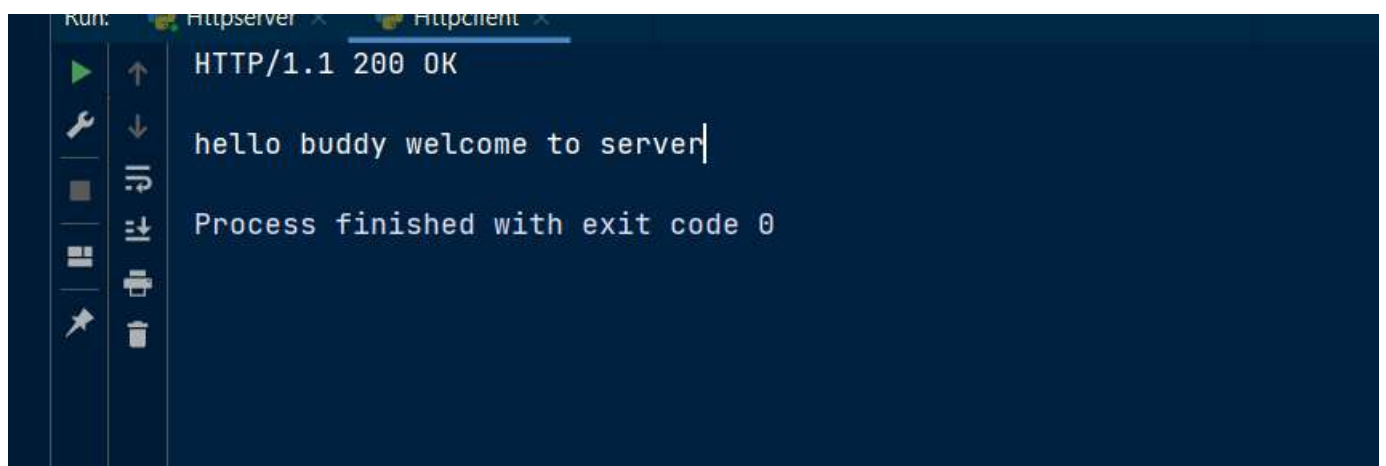
▼ 1. Server output



A screenshot of a terminal window with a dark blue background. At the top, there are two tabs: 'Httpserver' and 'HttpClient'. The 'HttpClient' tab is active. The terminal displays the text 'GET /HTTP/1.1' on the first line and ': localhost:8081' on the second line. A vertical cursor is positioned on the third line. On the left side of the terminal, there is a vertical toolbar with various icons for navigation and editing.

```
Run: Httpserver x HttpClient x
GET /HTTP/1.1
: localhost:8081
|
```

▼ 2. Client output



A screenshot of a terminal window with a dark blue background. At the top, there are two tabs: 'Httpserver' and 'HttpClient'. The 'HttpClient' tab is active. The terminal displays the text 'HTTP/1.1 200 OK' on the first line, 'hello buddy welcome to server' on the second line, and 'Process finished with exit code 0' on the third line. On the left side of the terminal, there is a vertical toolbar with various icons for navigation and editing.

```
Run: Httpserver x HttpClient x
HTTP/1.1 200 OK
hello buddy welcome to server
Process finished with exit code 0
```

