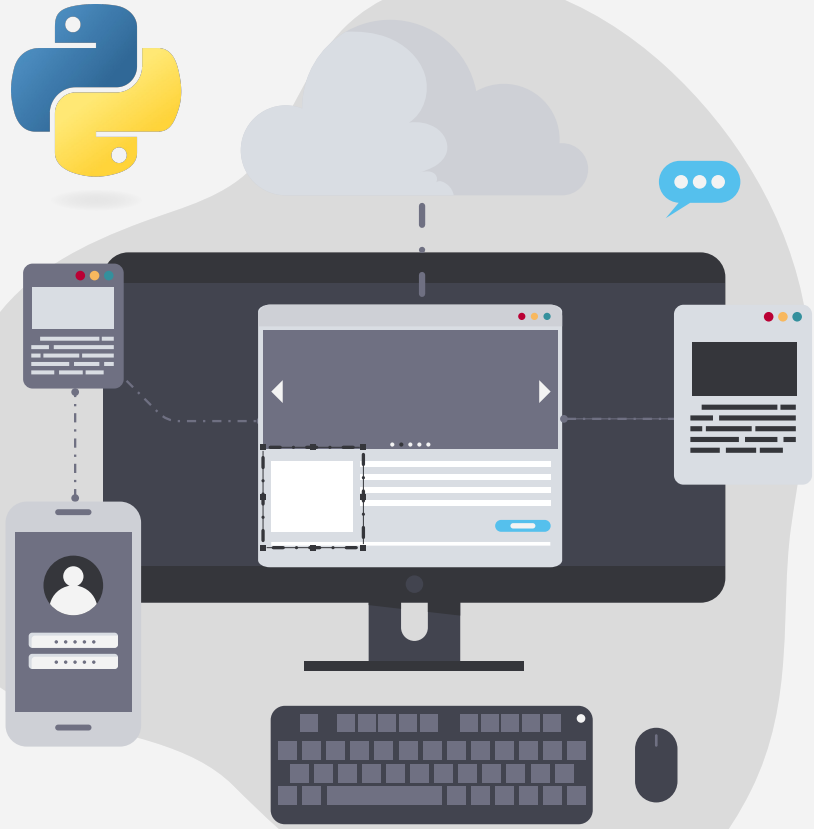
Socket Programming in Python

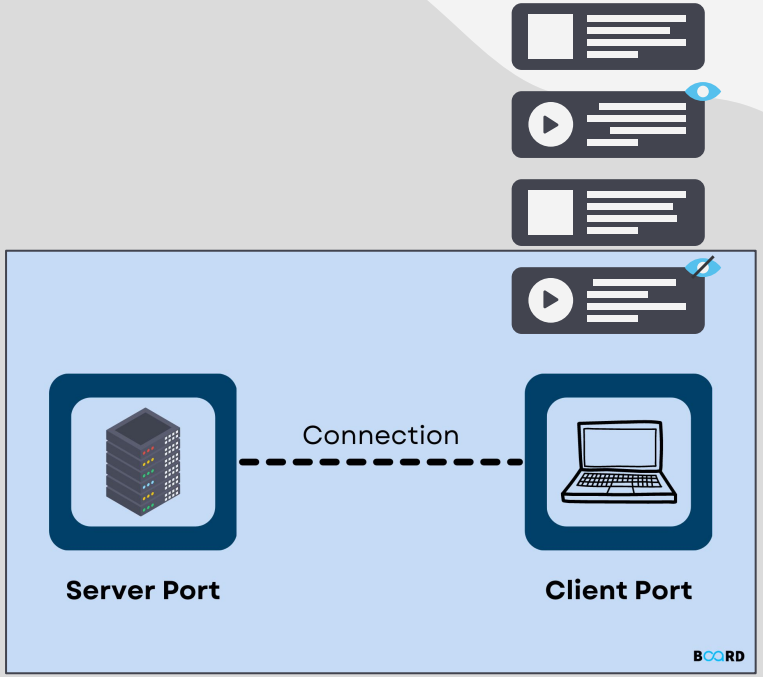
Echo Client and Server

Benjamin Taylor // ITCS-3166-001



Introduction

- **Purpose of the Project:**
  - We will explore the basics of socket programming, a key concept for enabling communication between applications on different devices over a network.
  - By creating a simple client-server application, we'll walk through the core steps of sending and receiving data through network sockets.
- **Key Concepts:**
  - Basic principles of network communication using sockets
  - Structure and coding of simple client and server applications in Python



Technical Setup

Tools Used:

Python

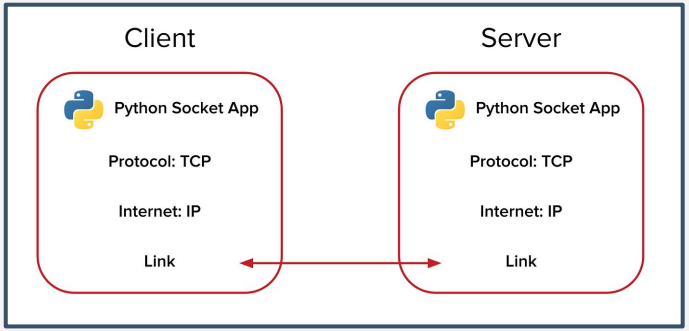
- A popular programming language offering comprehensive support for network communications through its built-in libraries

Socket Library

- This library is essential for creating and managing network connections. It provides the tools to send and receive data over the internet or between processes on the same machine.

Concept of Loopback Address

- The loopback address (127.0.0.1) is a unique IP address that allows a computer to communicate with itself.
- In this project, I'll be using it to test our client and server application on the same machine so we don't have to rely on any external network setup.



Client Program (client.py)

Import the Socket

Load the Python library needed to create network connections

Create a Socket Object

Initialize a new socket using IPv4 addressing (AF_INET) and TCP protocol (SOCK_STREAM)

Connect to Server

Establish a connection to the server located at '127.0.0.1' on port 12345

Send a Message

The message "Hello, server!" is encoded to bytes and sent over the network

Receive a Response

The client waits and receives a response from the server, decoding it back from bytes to string

Close Connection

Properly close the socket to free up the port and end the session

Client Program Code

Imports socket library needed for the network communications → `import socket`

Initializes new TCP socket using IPv4 addressing → `client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)`

Connects client to server using loopback address (port 12345) →

```
client_socket.connect(('127.0.0.1', 12345))
print("Connected to the server.")
```

Stores and sends encoded message to the server as bytes →

```
message = "Hello, server!"
client_socket.send(message.encode())
print(f"Sent to server: {message}")
```

Receives and decodes response from server (limited to 1024 bytes) →

```
response = client_socket.recv(1024).decode()
print(f"Received from server: {response}")
```

Closes socket, ending connection →

```
client_socket.close()
print("Client connection closed.")
```

Server Program (server.py)

Import the Socket

Import the necessary module for network communication

Create and Bind a Socket

Set up a new socket bound to '127.0.0.1' and port 12345, preparing it to accept connections

Listen for Connections

Configure the server to accept one connection at a time

Accept Connection

Establish a live connection with a client

Receive Message and Respond

After receiving a message from the client, the server sends back a confirmation message

Close Connection

Closing both client and server sockets ensures no resources are left hanging, which could lead to port and memory leaks

Server Program Code

Imports socket library needed for the network communications

```
import socket
```

Creates new socket object using IPv4 addressing and TCP protocol

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Binds server socket to loopback address and specifies port number

```
server_socket.bind(('127.0.0.1', 12345))
```

Configures server to accept connections (one at a time)

```
server_socket.listen(1)
print("Server is listening on port 12345...")
```

Outputs message to console, indicating server is listening

```
client_socket, client_address = server_socket.accept()
print(f"Connected to client at {client_address}")
```

Displays client information confirming a connection

```
message = client_socket.recv(1024).decode()
print(f"Received from client: {message}")
```

Receives data from client and decodes to string from bytes

```
response = "Hello, client! Your message was received."
client_socket.send(response.encode())
```

Sends encoded message back

```
client_socket.close()
server_socket.close()
```

Closes client socket and server socket, ending session with client and shutting down server

```
print("Server connection closed.")
```

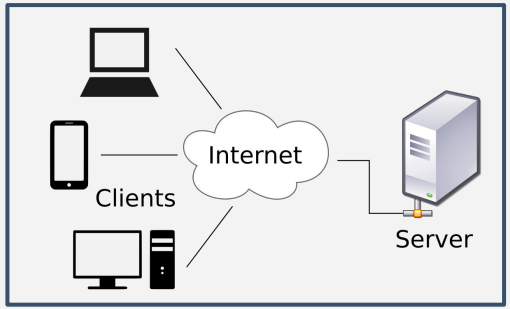
Conclusion

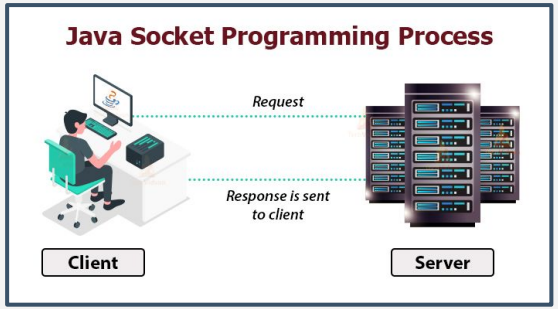
Overview

- Successfully implemented a client-server architecture using Python's socket programming abilities
- Demonstrated real-time communication between a client and server on a local machine using TCP/IP protocol

Takeaways

- Gained practical experience in network socket management, including opening, binding, listening, and closing sockets
- Enhanced understanding of the TCP/IP protocol suite, ensuring reliable data transmission and connection management





Resources

- https://en.wikipedia.org/wiki/Python_%28programming_language%29
- https://medium.com/@PubNub/python-socket-programming-client-server-peer-libraries-a61023e98e1f
- https://realpython.com/python-sockets/
- https://www.javatpoint.com/socket-programming-in-c-or-cpp
- https://www.datacamp.com/tutorial/a-complete-guide-to-socket-programming-in-python
- https://techvidvan.com/tutorials/java-socket-programming/
- https://en.wikipedia.org/wiki/Client%E2%80%93server_model