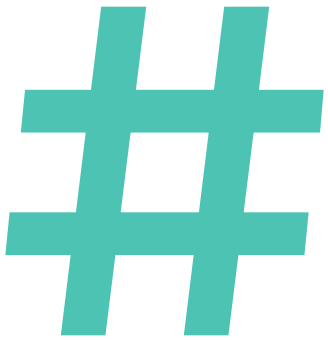


# Introduction to Socket

## Step 01 Socket

The Socket is a way to send messages across a network and establish a connection between two programs.

> **NOTE**
>
> Programs could be on the same computer or different computers. Socket uses TCP (Transmission Control Protocol)as the default protocol.

## Step 02 Client-Server Model

It is a model that depends on a server program and a client program and establishes the connection between them.

### 01 Client program

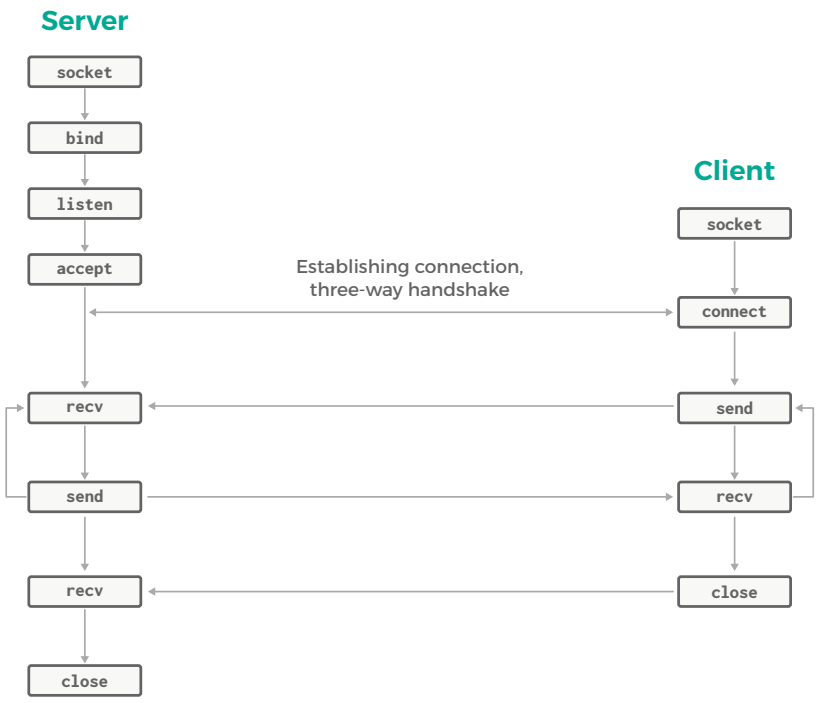
Connect to and interact with servers.

### 02 Server program

Listen to and accept network connections from client.

## Step 03 Socket API

It is a set of programming interfaces to perform network communication. It defines how applications can interact with the operating system's networking stack. It is an API that most programming language has built-in libraries that maps to this concept.



## Step 04 Socket API in Python

Python has built-in library that maps to the socket API. This library has multiple pre-built functions the programmers can use to tackle the network programming concept.

### 01 Socket module

- Is a built-in library that provides an interface for network communication, mapping directly to the socket API. It offers a collection of pre-built functions that simplify the creation and management of sockets, allowing developers to easily implement networking features in their applications.

```
socket()
bind()
listen()
accept()
connect()
send()
recv()
close()
```

## Step 05 Establishing The Socket Steps

There are 4 main steps to use the socket for establishing the connection between the Server program and the Client program.

### 01 Server

Preparing the server to listen for a request.

### 02 Client

Initiates a connection with the server.

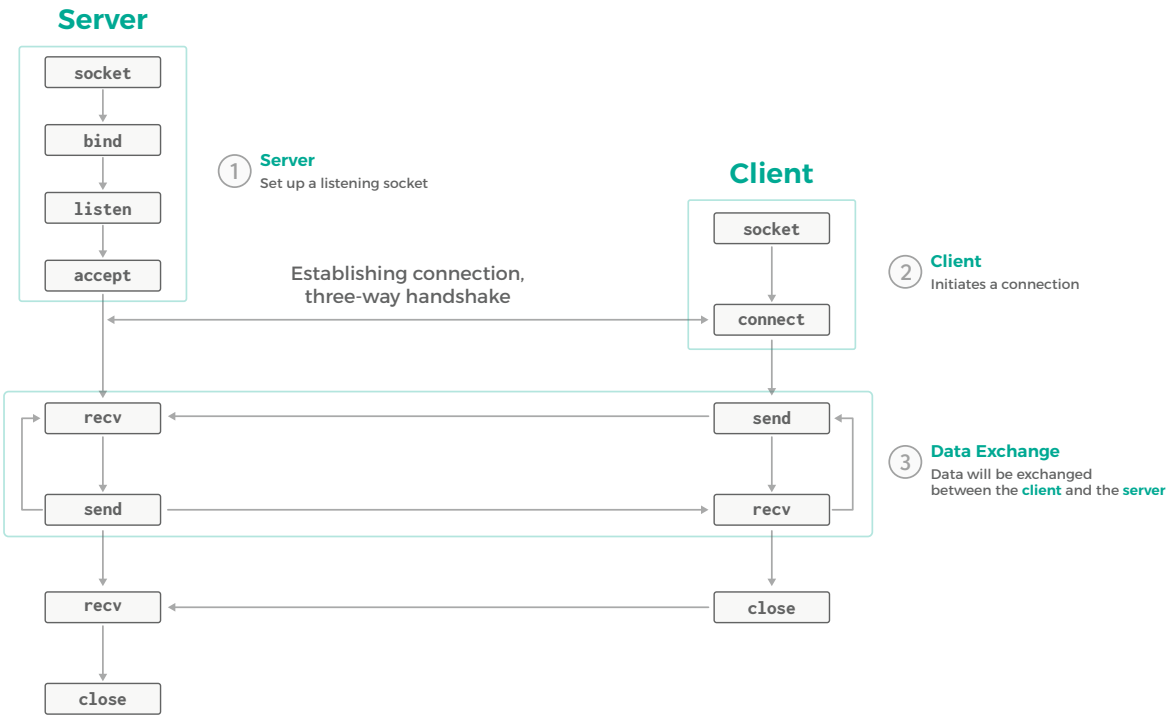
### 03 Data Exchange

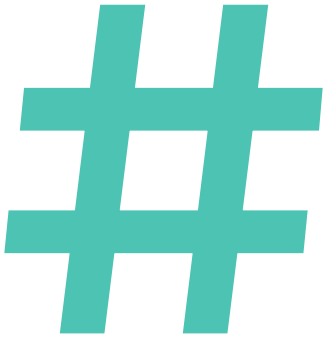
After connecting the server and the client, they can now send and receive messages (exchanging data).

### 04 Close the connection

The connection will be closed after establishing it and exchanging data between the server and the client.

In the following topics, we will use the above steps to build an echo Server-Client program. The server receives a message from the client, reads it, and sends it back to the client as it is (echo).





# Server

## Step 01 Set up a listening Socket

```
01  import socket
02  HOST = "127.0.0.1"
03  PORT = 65432
04  with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
05      s.bind((HOST, PORT))
06      s.listen()
07      addr, conn = s.accept()
08      print("connected")
```

CODE 1

This program represents the server program setting up a listening socket, every line and function in this program is important and it will be explained in the following steps.

## Step 02 HOST

The address that this socket accepts the connection from.

### 01 Specified connection

The Server program can specify the host it will accept the conneection from. If the Server program is accepting the connection from the same (local) computer, then it will use the local host "127.0.0.1".

```
HOST = "127.0.0.1"
```

If the server program is accepting a connection from another host, we can change the host to that one.

### 02 Un-specified connection

The Server program can accept a connection from any host by setting the host as empty.

```
HOST = ""
```

## Step 03 PORT

A port number specifies a running program that uses the network services (ex. sending or receiving data through the network).

> **NOTE**
>
> Ports less than 1024 reserved for specific services.

```
PORT = 65432
```

## Step 04 Create a Socket

```
socket.socket()
```

### 01 Function

Creates the socke.

### 02 Parameters

The first parameter is a constants represent the address and protocol families.

The second parameter is a constants represent the socket data exchange types.

### 03 Return

Returns a socket object whose methods implement the various socket system calls. methods like: socket.accept(), socket.bind(), and more we will use and explain in the following.

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

CODE 2

socket.AF_INET: indicates that this socket is using IPv4.
socket.SOCK_STREAM: indicates that this socket is exchanging data as stream of data.

## Step 05 Bind

Binds the socket to a specific host and port.

```
s.bind()
```

### 01 Function

One of the socket methods, it binds this socket to a specific host/port.

### 02 Parameters

A pair (host, port) is used for the AF_INET address family, where host is a string representing either a hostname in internet domain notation like 'daring.cwi.nl' or an IPv4 address like '100.50.200.5'.

### 03 Return

Returns None.

```
s.bind((HOST, PORT))
```

CODE 3

Binds this socket to the HOST = "127.0.0.1", and PORT = 65432

## Step 06 Listen for a Connection

```
s.listen()
```

### 01 Function

listen() is one of the socket methods. **Enable a server to accept connections.**

### 02 Parameters

No parameter needed.

### 03 Return

Returns None.

```
s.listen()
```

CODE 4

This socket now, listens for a connection.

## Step 07 Accept a Connection

```
s.accept()
```

### 01 Function

Accept a connection when one is initiated.

> **NOTE**
>
> Blocks remaining action until a connection is requested.

### 02 Parameters

No parameter needed.

### 03 Return

A pair (conn, address). conn is a new socket object usable to send and receive data on the connection. address is the address bound to the socket on the other end of the connection (Client).

```
conn, addr = s.accept()
```

CODE 5

conn is a new socket object used to send/receive data on this connection. addr is internet address of the client.

## Step 08 Run the Server Program

### 01 Run

After running the server program we built, your terminal will appear to hang. That's because the server is blocked because accept() method is blocking actions till a connection is requested.

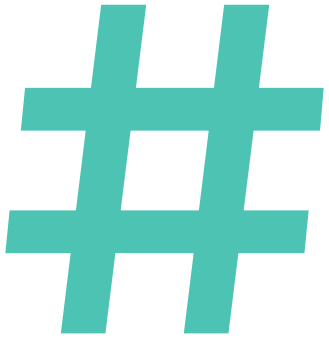
### 02 Connection request

This point will be implemented when we build the Client program in followin lectures. When a connection is requested to this server, it will print the below output.

```
connected
```

CODE 6

The Server program output's "connected" when a connection request is successfully initiated.



# Client

## Step 01 Initiates a Connection

```
01  import socket
02  HOST = "127.0.0.1"
03  PORT = 65432
04  with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
05      s.connect((HOST, PORT))
06      print("connection requested")
```

CODE 7

This program represent the client initating the connection to the server we built in the previous lecture, every line and function in this program is important and it will explained in the followig steps.

## Step 02 HOST

The server's hostname or IP address.

```
HOST = "127.0.0.1"
```

CODE 8

This socket will initiate a connection to specific host (127.0.0.1).

## Step 03 PORT

The port that the server is listening to requests on.

```
PORT = 65432
```

CODE 9

This socket will initiate a connection to specific port (65432).

## Step 04 Create a Socket

```
socket.socket()
```

### 01 Function

Creates the socket.

### 02 Parameters

The first parameter is a constants represent the address and protocol families.

The second parameter is a constants represent the socket types.

### 03 Return

Returns a socket object whose methods implement the various socket system calls.

methods like: socket.accept(), socket.bind(),socket.connect() and more.

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

CODE 10

socket.AF_INET: indicates that this socket is using IPv4
socket.SOCK_STREAM: indicates that this socket is using TCP.

## Step 05 Request a Connection

```
s.connect()
```

### 01 Function

One of the socket methods, it requests a connection to a server program.

### 02 Parameters

A pair (host, port) is accepted by this method, request a connection using the server's host and port.

### 03 Return

Returns None.

```
s.connect((HOST, PORT))
```

CODE 11

This socket is requesting to connect to a specific server, where the server's host and port are (127.0.0.1, 65432).

## Step 06 Run the Client Program

### 01 Run

After running the server, run the client program in a new and different terminal. In the client terminal you will see the below output.

**OUTPUT**

```
connection requested
```

This ouput indicates that the client connection request has been initiated.

Go back to the server terminal, If the connection is initiated successfully the server will output the below.

**OUTPUT**

```
connected
```

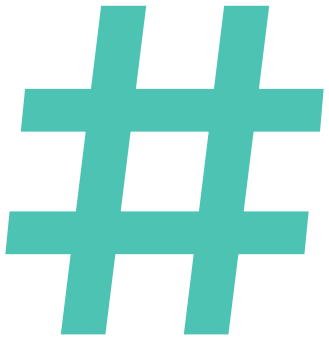
This is the output you should see in the server terminal, it indicates the connection is initiated.

### 02 Error

When running the client program before running the server first, it will cause to the below error.

**OUTPUT**

```
ConnectionRefusedError: [Error 61] Connection refused
```



# Data Exchange

## Step 01 Data Exchange within the Server Program

```
01  import socket
02
03  HOST = "127.0.0.1"
04
05  PORT = 65432
06
07  with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
08      s.bind((HOST, PORT))
09      s.listen()
10      conn, addr = s.accept()
11
```

```
12      with conn:
13          print(f"Connected by {addr}")
14          while True:
15              data = conn.recv(1024)
16              if not data:
17                  break
18              conn.sendall(data)
```

CODE 12

After accept() gives the client socket object conn, an infinite while loop is employed to repeatedly call conn.recv(), which reads any data sent by the client. This data is then echoed back using conn.sendall(). If conn.recv() returns an empty bytes object, b'', it indicates that the client has closed the connection, causing the loop to terminate. The with statement is used with conn to ensure that the socket is automatically closed at the end of the block.

## Step 02 Server Program Main Functionality

### 01 Input

Receive data from the client (Request).

```
data = conn.recv(1024)
```

CODE 13

The data variable will receive and hold the data from the client. conn is the new socket we initiated to handle the connection between the server and the client. recv() is one of the socket methods that allows the socket to receive data, and the parameter (1024) indicates that the server will only receive data =< 1024 bytes.

### 02 Process

The client's data usually includes a request for the server's resources (REQUEST), and the server should process the requested service. The server we built offers one service only, sending back the data it receives as it is.

### 03 Output

After receiving the data from the client and processing the requested service, the server will send a (RESPONSE) to the client with the requested service.

```
conn.sendall(data)
```

CODE 14

sendall() is one of the socket methods, it accepts a string representing the data that will be sent to the client (RESPONSE).

## Step 03 Data Exchange within the Client Program.

```
01  import socket
02
03  HOST = "127.0.0.1"  # The server's hostname or IP address
04  PORT = 65432  # The port used by the server
05
06  with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
07      s.connect((HOST, PORT))
08      s.sendall(b"Hello, world")
09      data = s.recv(1024)
10
11  print(f"Received {data!r}")
```

CODE 15

In comparison to the server, the client is pretty simple. It creates a socket object, uses connect() to connect to the server and calls s.sendall() to send its message. Lastly, it calls s.recv() to read the server's reply and then prints it.

## Step 04 Client Program Main Functionality

### 01 Input

The client will receive the RESPONSE from the server

```
data = s.recv(1024)
```

CODE 16

The client will receive the RESPONE from the Client using the method recv() and store the RESPONSE to the variable data. The 1024 parameter passed to the recv(1024) method, indicates that the client will only receive data =< 1024 bytes.

### 02 Process

After receiveing the response from the server the client usually do some operations on it. In our case the clinet will only receive the response and print it.

### 03 Output

The client will output (print) the received RESPONSE.

```
print(f"Received {data!r}")
```

OUTPUT

```
Received Hello, world
```

## Step 05 Close Connection

The socket() we used closes the connection after exchanging the data, no need for the s.close() function.

# Step 06 Socket Run

## 01 Start the server

```
python echo-server.py
```

> **NOTE**
>
> Server program pauses because the s.accept() blocks the execution. Program is still running, and execution will continue when a connection is requested.

## 02 Start the client in a diffrent terminal session.

```
python echo-client.py
```

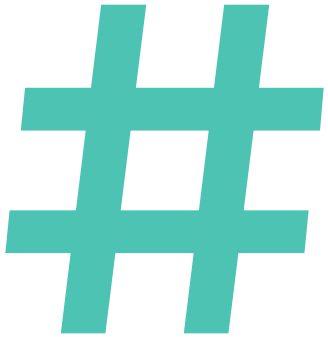
After running the client program, it should output the below

```
Received b'Hello, world'
```

On the server program terminal, after running the client, you should see the below output

```
Connected by ('127.0.0.1', client port number)
```

> **NOTE client port number**
>
> The client port number displayed in the server program terminal (output) is auto-generated and not constant. This means it will be different every time you run the program.



# Socket State

## Step 01 Socket state

Sockets have multiple and different states to be in. Like CLOSED indicating the socket has been closed, and LISTEN indicating the socket is listening (waiting for the client to send a message). We can indicate the current state of the running socket using terminal commands.

### 01 Run

After running the server, we can view the Server socket state by running the below command on the terminal.

```
lsof -i -n
```

CODE 17
lsof stands for list of files.

### 02 Output

Running the lsof -i -n command, will result the below output.

**OUTPUT**

| COMMAND | PID | USER | FD | TYPE | DEVICE | SIZE/OFF | NODE | NAME |
|---|---|---|---|---|---|---|---|---|
| Python | 28004 | **** | 3u | IPv4 | 0xa2ba90238d4e2435 | 0t0 | TCP | 127.0.0.1:65432 (LISTEN). |

COMMAND, PID (process ID), and USER (user ID) of open Internet sockets