6 Python Sockets Example

Jacques Mock Schindler

03.09.2025

In this section, we will explore how to create a simple client-server application using Python sockets. This example will demonstrate the basic concepts of socket programming, including how to establish a connection, and send and receive data.

> **Network Socket**
>
> A network socket is a software structure within a network node of a computer network that serves as an endpoint for sending and receiving data across the network.

Communication Structure

The example will consist of a server that listens for incoming connections and a client that connects to the server. The communication will be modelled as a kind of chat service.

Server Code

To run the following example code there is no virtual environment required. The example uses just core Python libraries.
Below is the working code example for the server:

```
# socket_server.py

import socket

def server_program():
    # define host name and port
    host = 'localhost'    # for coummunication on the local machine
    port = 5000           # initiate port no above 1024

    # create socket
```

```
    server_socket = socket.socket()

    # bind the socket to the host and port
    server_socket.bind((host, port))

    # set the server to listen for connections
    server_socket.listen(2) # max 2 clients can connect

    # accept new connection from client
    conn, address = server_socket.accept()
    print("Connection from: " + str(address))

    while True:
        # receive data stream. it won't accept data packet greater than 1024 bytes
        data = conn.recv(1024).decode()
        if not data:
            # if data is not received break
            break
        print("Recived from connected client: " + str(data))
        # prompt the user to enter a message
        data = input(' -> ')
        # send data to the client as bytes
        conn.send(data.encode())

    # close the connection
    conn.close()

if __name__ == '__main__':
    server_program()
```

To run the script directly, open a terminal in the directory where the script is located and run the following command:

```
python socket_server.py
```

What happens if you do so will be explained in the following sections.

The code consist of two main parts: the `server_program` function and the `if __name__ == '__main__':` block. The `server_program` function contains the main logic for the server, while the `if` block is used to execute the server code when the script is run directly.

When the script is run, the first thing that happens is the import of the `socket` module, which provides the necessary functions and classes for working with sockets in Python.

Next, the `server_program` function is called. Inside this function, the first thing that happens is the definition of the host address and the port number. `localhost` is a special address that refers to the local machine. `localhost` is equivalent to the IPv4 address `127.0.0.1`. This means that the server will only accept connections from clients running on the same machine. This is a simulation of a network on the local machine.
On line #11, the `server_socket` is created using the `socket.socket()` function, which creates a new socket object. This socket will be used to listen for incoming connections from clients. To make the connection work the `server_socket` must be bound to the host and port using the `bind()` method (line #14). Next, the server is set to listen for incoming connections with the `listen()` method (line #17). The server can accept a maximum of 2 clients at a time. This limit is set to prevent the server from being overwhelmed by too many connections at once. The `accept()` method is used to accept a new connection from a client. It gets the tuple `(conn, address)` where `conn` is a new socket object that can be used to communicate with the client, and `address` is the address of the client. On line #21, a message is printed to show the address of the connected client.
On line #23, the server enters a loop where it waits for data from the client. The `recv()` method is used to receive data from the client. The length of the data that can be received is limited to 1024 bytes. If the client sends more data than this, it will be truncated. If no data is received, the server breaks out of the loop and closes the connection. If the server receives data, it prints the data to the console and then prompts the user to enter a response (line #31). The response is sent back to the client using the `send()` method.

Client Code

Below is the working code of the client example:

```
1  # socket_client.py
2
3  import socket
4
5  def client_program():
6      # define host name and port (of the server to connect to)
7      host = 'localhost'  # as both code is running on same pc
8      port = 5000         # socket server port number
9
10     # create socket
11     client_socket = socket.socket()
12
13     # connect to the server
```

```
    client_socket.connect((host, port))

    # prompt the user to enter a message
    message = input(" -> ")  # take input

    # message loop
    while message.lower().strip() != 'bye':
        # send message to the server as bytes
        client_socket.send(message.encode())

        # receive response from the server
        data = client_socket.recv(1024).decode()

        print('Received from server: ' + data)

        # prompt the user to enter a new message
        message = input(" -> ")  # take new input

    # close the connection
    client_socket.close()

if __name__ == '__main__':
    client_program()
```

To run this script directly, it is the same as for the server script. Open a terminal in the directory where the script is located and run the following command:

```
python socket_client.py
```

If the script runs, the first thing that happens is the import of the `socket` module.

Next, the `client_program` function is called. Inside this function, the first thing that happens is the definition of the host address and the port number. Here, the host address and the port number have to be the ones defined in the server script.

After defining the host and port, the client creates a socket object using the `socket.socket()` function. This socket will be used to connect to the server. The `connect()` method is called on the socket object to establish a connection to the server.

Once the connection is established, the client enters a loop where it can send messages to the server and receive responses. The client prompts the user to enter a message, which is then sent to the server using the

`send()` method. The client also waits for a response from the server using the `recv()` method.

If the user enters ‘bye’, the client will exit the loop and close the connection to the server.