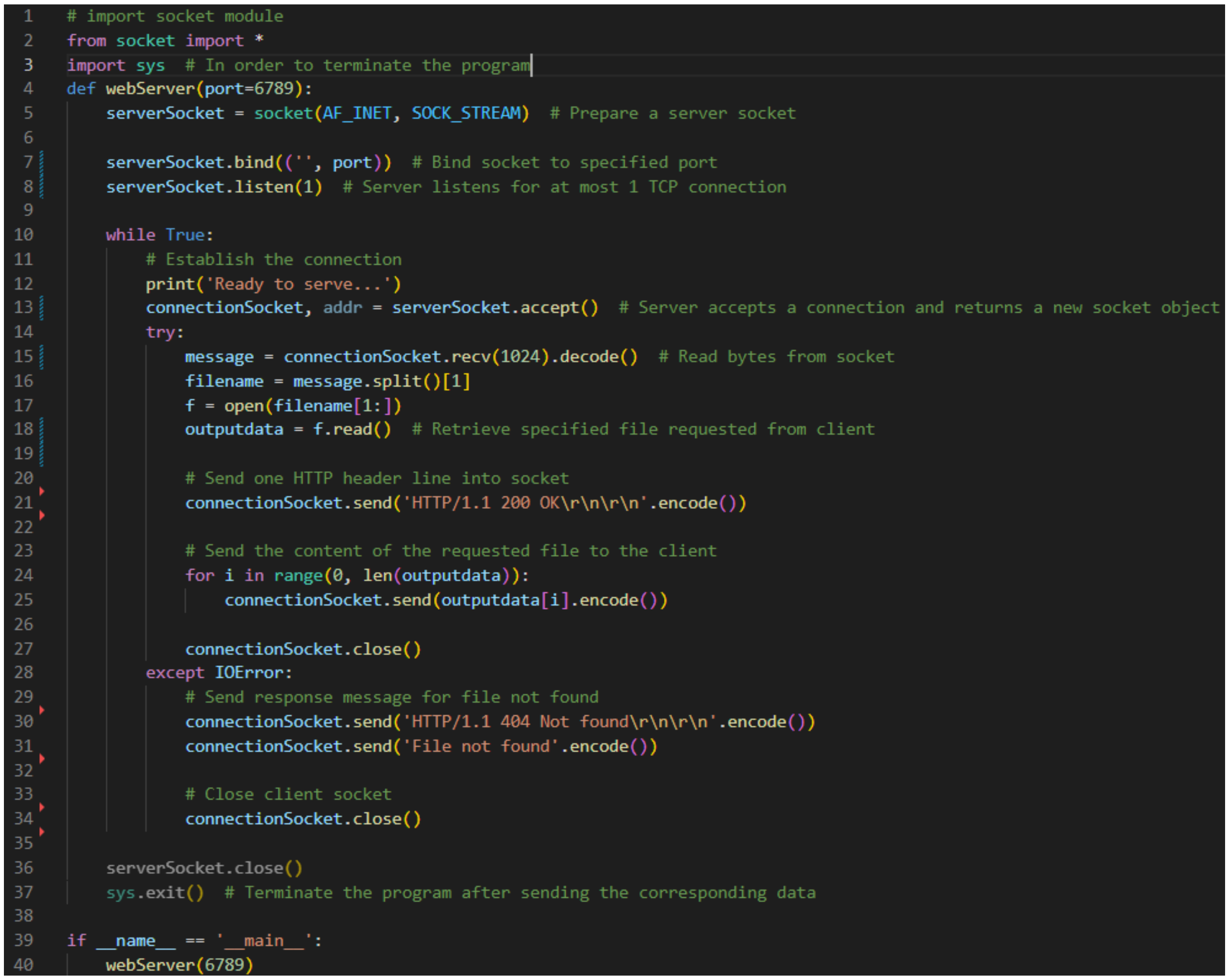
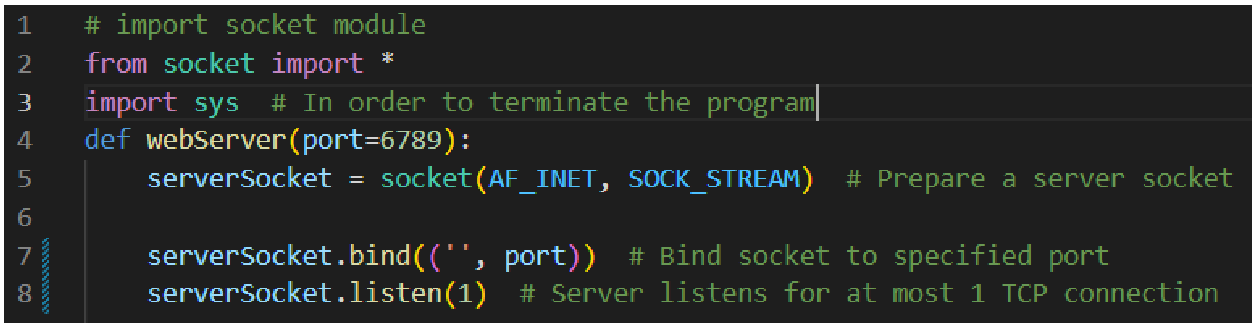
Socket Programming Lab #1: Web Server

Full program for *server.py* provided below:



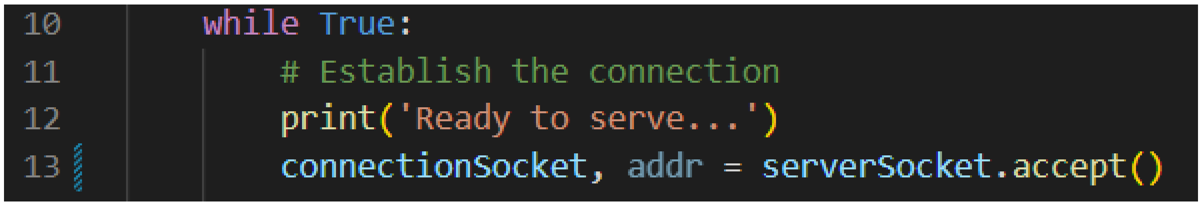
```python
# import socket module
from socket import *
import sys  # In order to terminate the program
def webServer(port=6789):
    serverSocket = socket(AF_INET, SOCK_STREAM)  # Prepare a server socket

    serverSocket.bind(('', port))  # Bind socket to specified port
    serverSocket.listen(1)  # Server listens for at most 1 TCP connection

    while True:
        # Establish the connection
        print('Ready to serve...')
        connectionSocket, addr = serverSocket.accept()  # Server accepts a connection and returns a new socket object
        try:
            message = connectionSocket.recv(1024).decode()  # Read bytes from socket
            filename = message.split()[1]
            f = open(filename[1:])
            outputdata = f.read()  # Retrieve specified file requested from client

            # Send one HTTP header line into socket
            connectionSocket.send('HTTP/1.1 200 OK\r\n\r\n'.encode())

            # Send the content of the requested file to the client
            for i in range(0, len(outputdata)):
                connectionSocket.send(outputdata[i].encode())

            connectionSocket.close()
        except IOError:
            # Send response message for file not found
            connectionSocket.send('HTTP/1.1 404 Not found\r\n\r\n'.encode())
            connectionSocket.send('File not found'.encode())

            # Close client socket
            connectionSocket.close()

    serverSocket.close()
    sys.exit()  # Terminate the program after sending the corresponding data

if __name__ == '__main__':
    webServer(6789)
```

After creating the server socket in line 5, we must set up the server socket to receive incoming TCP connections. In line 7, we bind the server socket to a specified port number, 6789. In line 8, we specify that the server will listen to at most 1 TCP connection before refusing any other incoming connections.



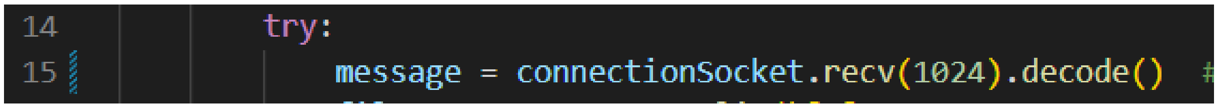
```
1   # import socket module
2   from socket import *
3   import sys  # In order to terminate the program
4   def webServer(port=6789):
5       serverSocket = socket(AF_INET, SOCK_STREAM)  # Prepare a server socket
6
7       serverSocket.bind(('', port))  # Bind socket to specified port
8       serverSocket.listen(1)  # Server listens for at most 1 TCP connection
```

In line 13, the server accepts a connection and, in return, creates a new socket object `connectionSocket` that can send and receive data on the connection. `addr` is the address bound to the socket on the other end of the connection.



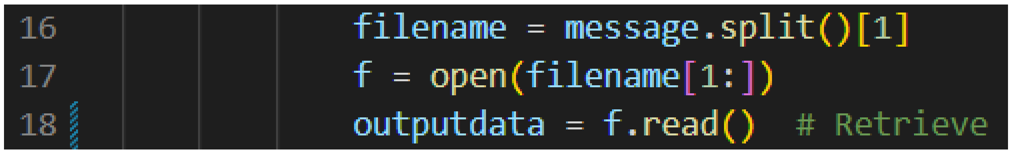
```
10      while True:
11          # Establish the connection
12          print('Ready to serve...')
13          connectionSocket, addr = serverSocket.accept()
```

In line 15, we create a variable named `message` that will read in bytes of data received by `connectionSocket`.



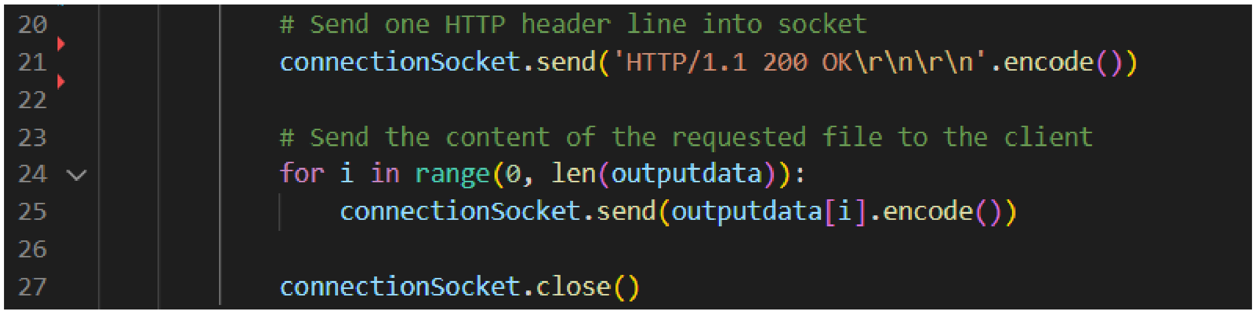
```
14          try:
15              message = connectionSocket.recv(1024).decode()  #
```

In line 18, we create a variable named outputData to act as a buffer that will contain the contents of the file specified in the client request (from line 15).



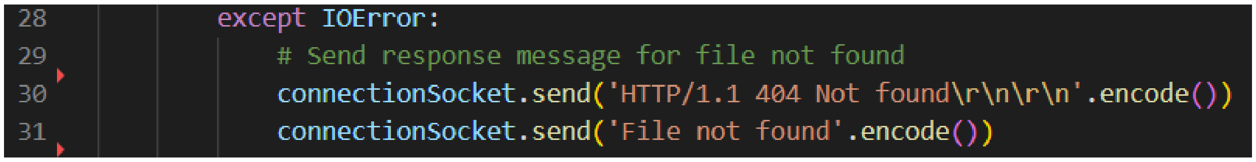
```
16              filename = message.split()[1]
17              f = open(filename[1:])
18              outputdata = f.read()  # Retrieve
```

In line 21, the program sends a HTTP 200 OK header encoded as bytes from `connectionSocket` to the client.



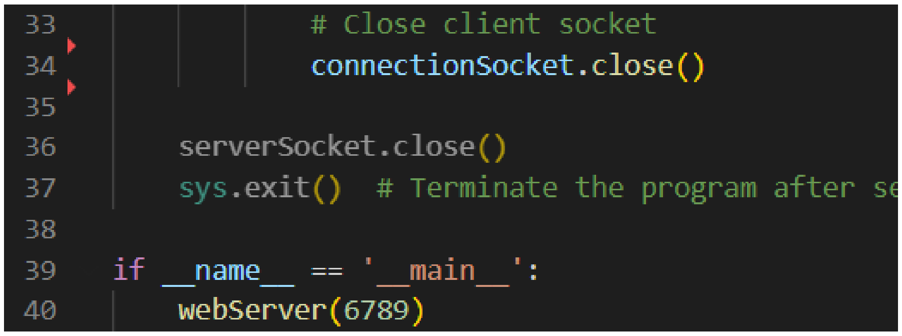
```
20              # Send one HTTP header line into socket
21              connectionSocket.send('HTTP/1.1 200 OK\r\n\r\n'.encode())
22
23              # Send the content of the requested file to the client
24              for i in range(0, len(outputdata)):
25                  connectionSocket.send(outputdata[i].encode())
26
27              connectionSocket.close()
```

If the client requests a file that does not exist in the server directory, then we want to return an error code. In line 30, the program sends a HTTP 404 header encoded as bytes from `connectionSocket` to the client. In line 31, the program sends a response message “File not found” from `connectionSocket` to the client.



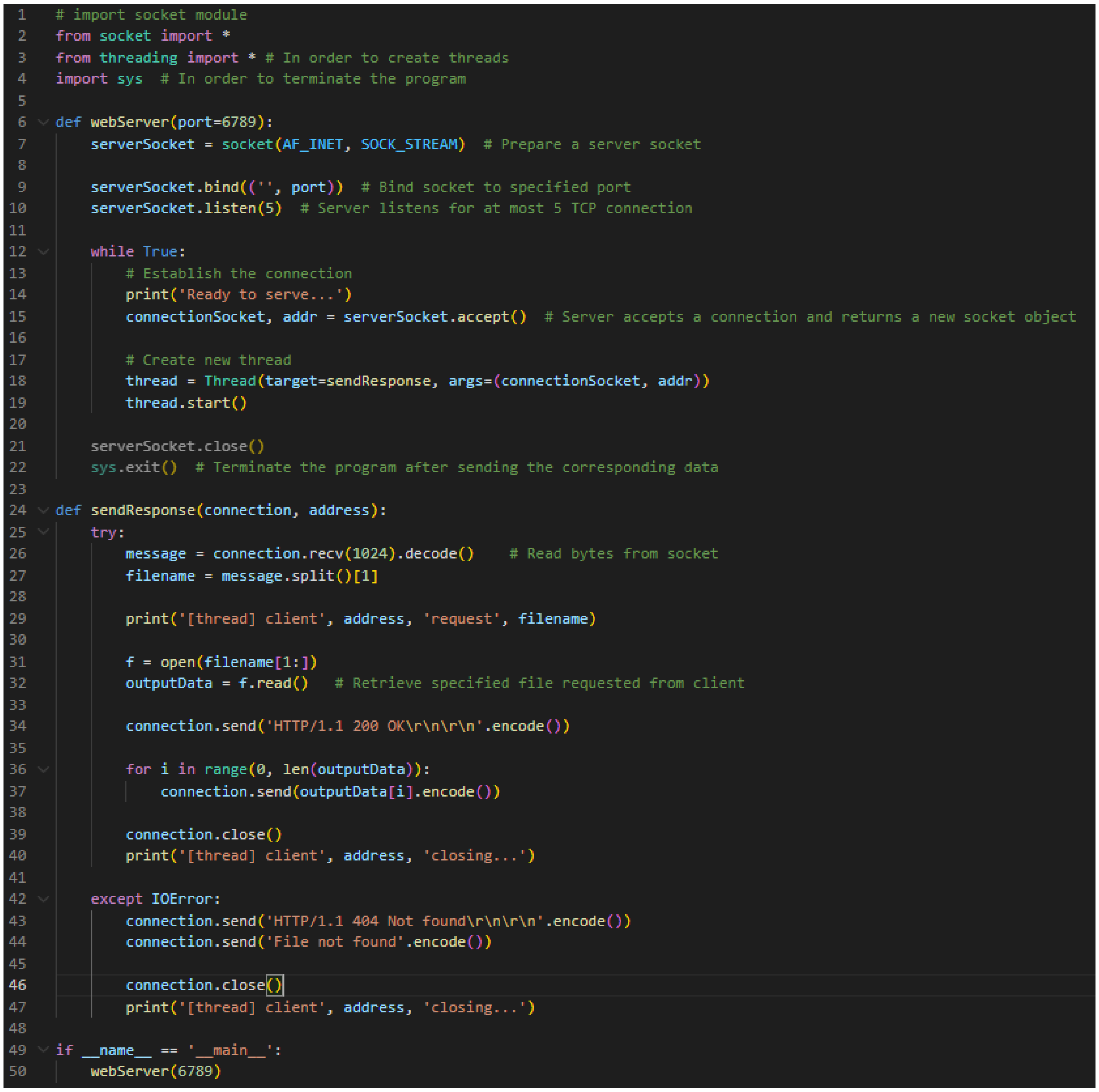
```
28          except IOError:
29              # Send response message for file not found
30              connectionSocket.send('HTTP/1.1 404 Not found\r\n\r\n'.encode())
31              connectionSocket.send('File not found'.encode())
```

In line 34, after sending an error message if the client requests a file that does not exist, the program closes out `connectionSocket`.



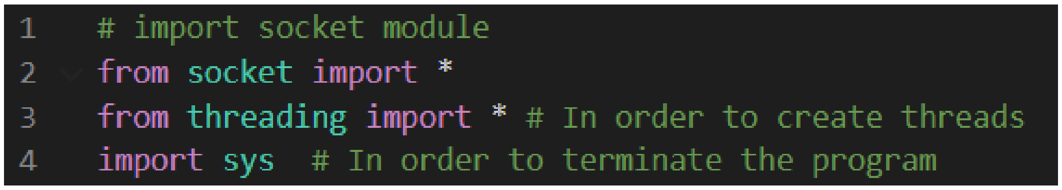
```
33              # Close client socket
34              connectionSocket.close()
35
36      serverSocket.close()
37      sys.exit()  # Terminate the program after se
38
39  if __name__ == '__main__':
40      webServer(6789)
```

Full program for *multiserver.py* provided below:



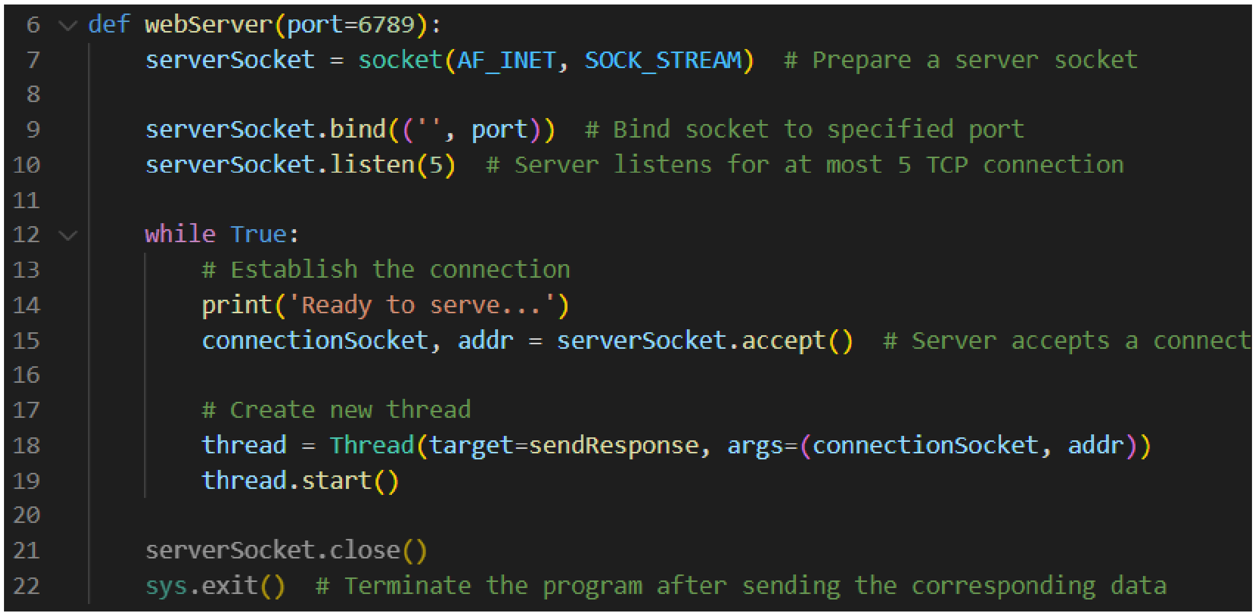
```python
# import socket module
from socket import *
from threading import * # In order to create threads
import sys  # In order to terminate the program

def webServer(port=6789):
    serverSocket = socket(AF_INET, SOCK_STREAM)  # Prepare a server socket

    serverSocket.bind(('', port))  # Bind socket to specified port
    serverSocket.listen(5)  # Server listens for at most 5 TCP connection

    while True:
        # Establish the connection
        print('Ready to serve...')
        connectionSocket, addr = serverSocket.accept()  # Server accepts a connection and returns a new socket object

        # Create new thread
        thread = Thread(target=sendResponse, args=(connectionSocket, addr))
        thread.start()

    serverSocket.close()
    sys.exit()  # Terminate the program after sending the corresponding data

def sendResponse(connection, address):
    try:
        message = connection.recv(1024).decode()    # Read bytes from socket
        filename = message.split()[1]

        print('[thread] client', address, 'request', filename)

        f = open(filename[1:])
        outputData = f.read()   # Retrieve specified file requested from client

        connection.send('HTTP/1.1 200 OK\r\n\r\n'.encode())

        for i in range(0, len(outputData)):
            connection.send(outputData[i].encode())

        connection.close()
        print('[thread] client', address, 'closing...')

    except IOError:
        connection.send('HTTP/1.1 404 Not found\r\n\r\n'.encode())
        connection.send('File not found'.encode())

        connection.close()
        print('[thread] client', address, 'closing...')

if __name__ == '__main__':
    webServer(6789)
```

Lines 1-4 import the necessary modules that we will use to create the multithreaded server. Line 2 imports the socket module, which we will use to create the server-side sockets that receive client requests. Line 3 imports the threading module, which we will use to create threads as multiple clients send requests to the server simultaneously. Line 4 imports the sys module, that we will use to terminate the program.



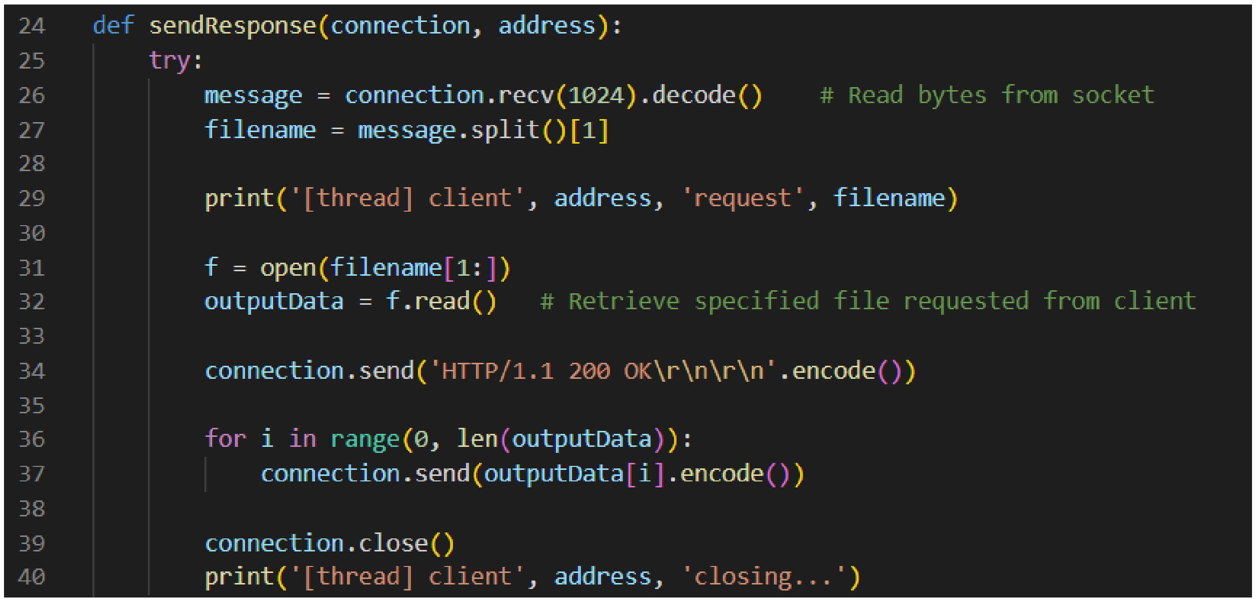
```
1    # import socket module
2    from socket import *
3    from threading import * # In order to create threads
4    import sys  # In order to terminate the program
```

The function *webServer* is similar to that of *server.py* with some modifications to support multiple threads. In line 10, we change the maximum amount of TCP connections the server will listen for to 5. After the server socket accepts a TCP connection and creates a new socket, the program creates a new thread (line 18) and passes the new TCP connection socket and the address of the client-side socket to the target function *sendResponse*. In line 19, the program then starts the newly created thread.



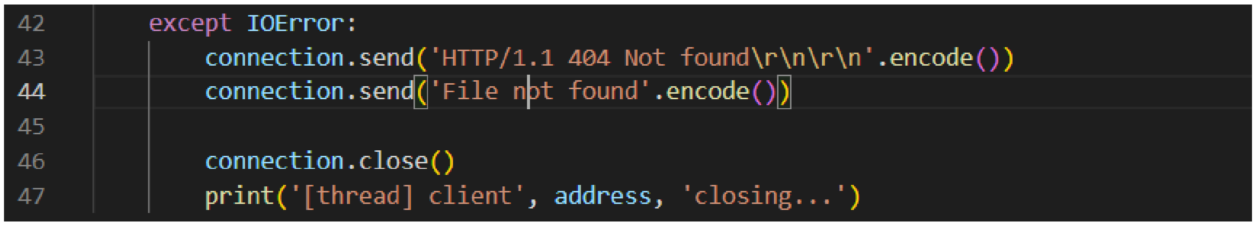
```
6   def webServer(port=6789):
7       serverSocket = socket(AF_INET, SOCK_STREAM)  # Prepare a server socket
8
9       serverSocket.bind(('', port))  # Bind socket to specified port
10      serverSocket.listen(5)  # Server listens for at most 5 TCP connection
11
12      while True:
13          # Establish the connection
14          print('Ready to serve...')
15          connectionSocket, addr = serverSocket.accept()  # Server accepts a connect
16
17          # Create new thread
18          thread = Thread(target=sendResponse, args=(connectionSocket, addr))
19          thread.start()
20
21      serverSocket.close()
22      sys.exit()  # Terminate the program after sending the corresponding data
```

Starting from line 24, we define the function *sendResponse*. *sendResponse* accepts a connection socket and address bound to the client socket as its parameters. You will notice that much of this code is similar to that of the *webServer* function in *server.py*. In line 26, the connection socket receives and decodes the message received from the client during the TCP connection. In line 27, the program extracts the filename from the message. In lines 31-32, the program retrieves the requested file from the server directory and reads its contents into a buffer, *outputData*. In line 34, the connection socket sends a HTTP 200 OK response header to the client socket. In lines 36-37, the connection socket sends the rest of the contents of the requested file to the client socket. In line 39, the connection socket is closed as the response is completed.



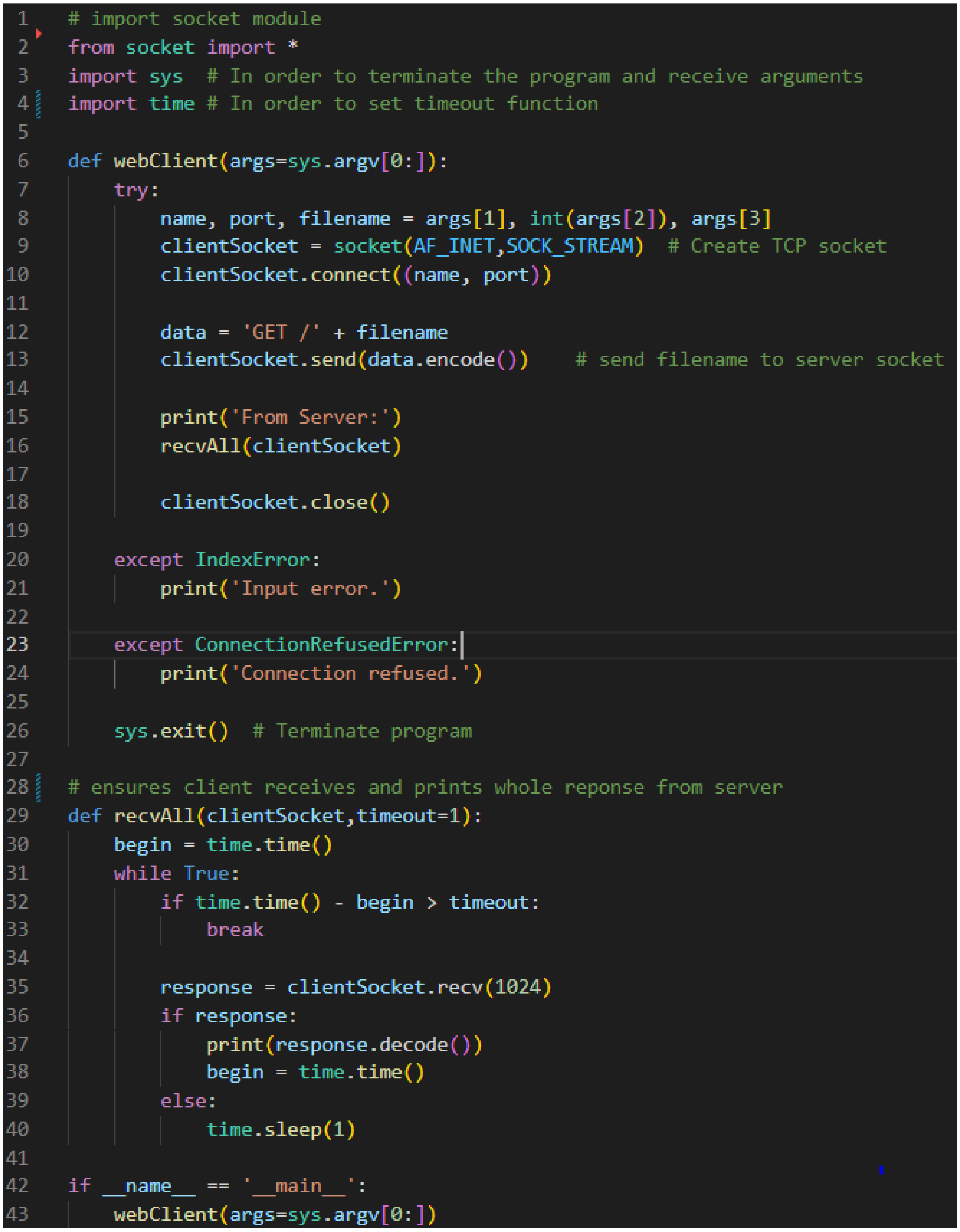
```
24  def sendResponse(connection, address):
25      try:
26          message = connection.recv(1024).decode()    # Read bytes from socket
27          filename = message.split()[1]
28
29          print('[thread] client', address, 'request', filename)
30
31          f = open(filename[1:])
32          outputData = f.read()   # Retrieve specified file requested from client
33
34          connection.send('HTTP/1.1 200 OK\r\n\r\n'.encode())
35
36          for i in range(0, len(outputData)):
37              connection.send(outputData[i].encode())
38
39          connection.close()
40          print('[thread] client', address, 'closing...')
```

In lines 42-47, the program handles any IOErrors in the event that the client requests a file that does not exist in the server directory. In line 43, the connection socket sends a HTTP 404 Not Found response header to the client socket, as well as a 'File not found' message in line 44. In line 46, the connection socket is closed as the response is complete.



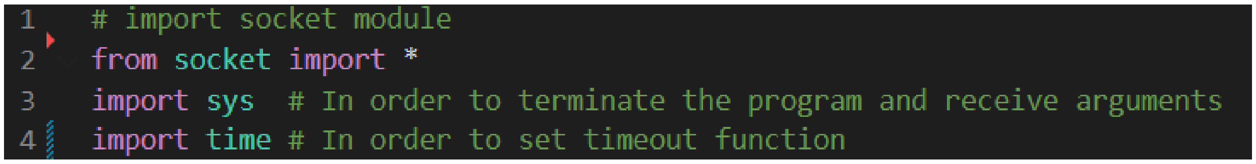
```
42      except IOError:
43          connection.send('HTTP/1.1 404 Not found\r\n\r\n'.encode())
44          connection.send('File not found'.encode())
45
46          connection.close()
47          print('[thread] client', address, 'closing...')
```

Full program for *client.py* provided below:



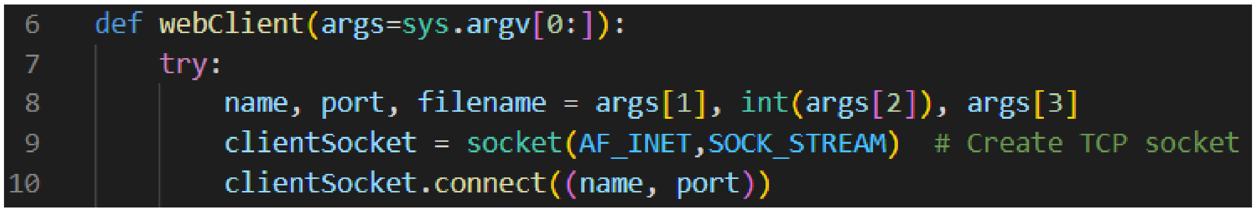
```python
# import socket module
from socket import *
import sys  # In order to terminate the program and receive arguments
import time # In order to set timeout function

def webClient(args=sys.argv[0:]):
    try:
        name, port, filename = args[1], int(args[2]), args[3]
        clientSocket = socket(AF_INET,SOCK_STREAM)  # Create TCP socket
        clientSocket.connect((name, port))

        data = 'GET /' + filename
        clientSocket.send(data.encode())    # send filename to server socket

        print('From Server:')
        recvAll(clientSocket)

        clientSocket.close()

    except IndexError:
        print('Input error.')

    except ConnectionRefusedError:
        print('Connection refused.')

    sys.exit()  # Terminate program

# ensures client receives and prints whole reponse from server
def recvAll(clientSocket,timeout=1):
    begin = time.time()
    while True:
        if time.time() - begin > timeout:
            break

        response = clientSocket.recv(1024)
        if response:
            print(response.decode())
            begin = time.time()
        else:
            time.sleep(1)

if __name__ == '__main__':
    webClient(args=sys.argv[0:])
```

Lines 1-4 import the necessary modules to successfully execute *client.py*. Line 2 imports the socket module we will use to create the client-side socket that will initiate a TCP connection to the server-side socket. Line 3 imports the *sys* module we will use to terminate the program when the client socket receives and prints the data from the server socket, as well as accept command line arguments when specifying the server name, port number, and filename to be requested. Line 4 imports the *time* module we will use in our timeout function when receiving data from the server socket.



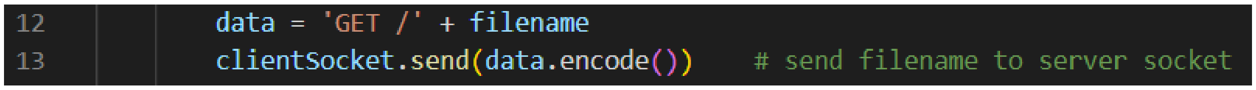
```
1   # import socket module
2   from socket import *
3   import sys  # In order to terminate the program and receive arguments
4   import time # In order to set timeout function
```

Line 6 defines the function *webClient* where we will set up the client socket, initiate a TCP connection to the server socket, send a client request to the server, and receive the server's response. *webClient* accepts the argument *args* which will be a list of all command line arguments. In line 8, we define the server name, port number, and filename from the list *args*. In line 9, the program creates the client TCP socket. In line 10, the program initiates the TCP connection to the server socket using the server name and port number defined previously in line 8.



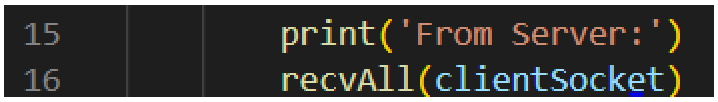
```
6   def webClient(args=sys.argv[0:]):
7       try:
8           name, port, filename = args[1], int(args[2]), args[3]
9           clientSocket = socket(AF_INET,SOCK_STREAM)  # Create TCP socket
10          clientSocket.connect((name, port))
```

In line 12, we format the client request in the form "GET /" + filename. This is because when the server receives a client request, it splits the request up by word and extracts the second word as the file to return. Please refer to line 16 in *server.py* above. In line 13, the program encodes the client request and sends it through the client socket to the server socket.



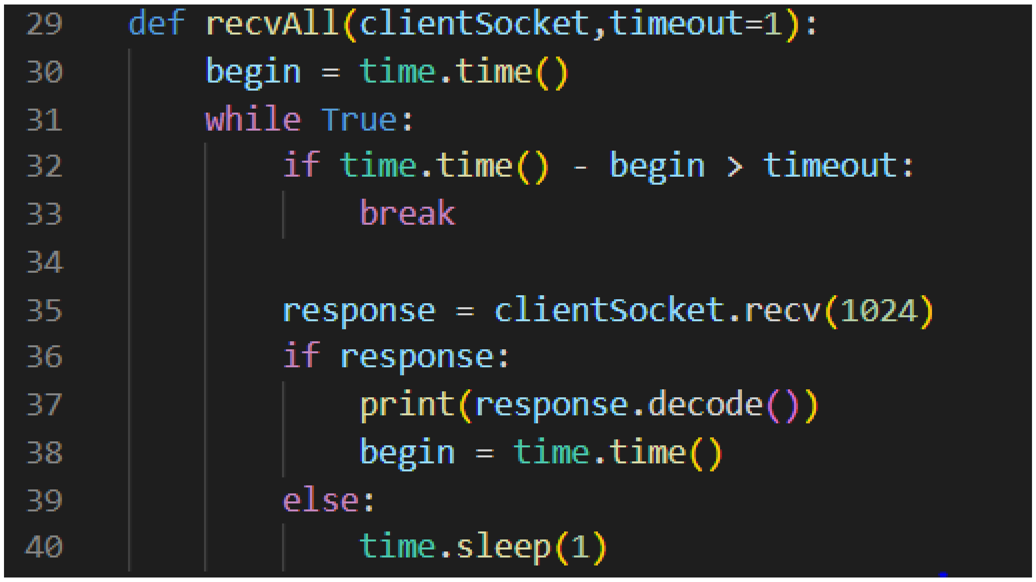
```
12          data = 'GET /' + filename
13          clientSocket.send(data.encode())    # send filename to server socket
```

In line 15, the program prints to the console “From Server:” and prepares to receive data from the server. In line 16, the program calls the function *recvAll* and passes the client socket as an argument to receive all of the data the server socket will send over and prints it to the console. The *recvAll* function is specified below.



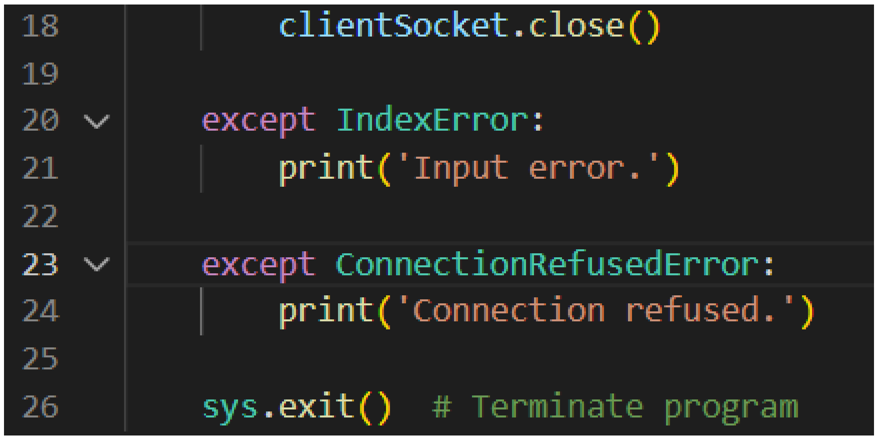
```
15          print('From Server:')
16          recvAll(clientSocket)
```

The *recvAll* function defined in lines 29-40 sets a timeout range of 1 second before the function exits after not receiving any more data from the server socket. If the client socket receives data from the server, the *recvAll* function will print the response to the console and reset the timeout. This ensures that the client socket has received all incoming data sent from the server socket.



```
29  def recvAll(clientSocket,timeout=1):
30      begin = time.time()
31      while True:
32          if time.time() - begin > timeout:
33              break
34  
35          response = clientSocket.recv(1024)
36          if response:
37              print(response.decode())
38              begin = time.time()
39          else:
40              time.sleep(1)
```

After receiving and printing the server response to the console, line 18 will close the client socket. In lines 20-21, the program catches any IndexErrors that would occur in the case that the user inputs arguments formatted differently from `python client.py servername port filename`. In lines 23-24, the program catches any ConnectionRefusedErrors that would occur in the case that the client socket attempts to connect to an offline server. In line 26, the program exits the program once the function *webClient* completes.



```
18          clientSocket.close()
19
20      except IndexError:
21          print('Input error.')
22
23      except ConnectionRefusedError:
24          print('Connection refused.')
25
26      sys.exit()  # Terminate program
```