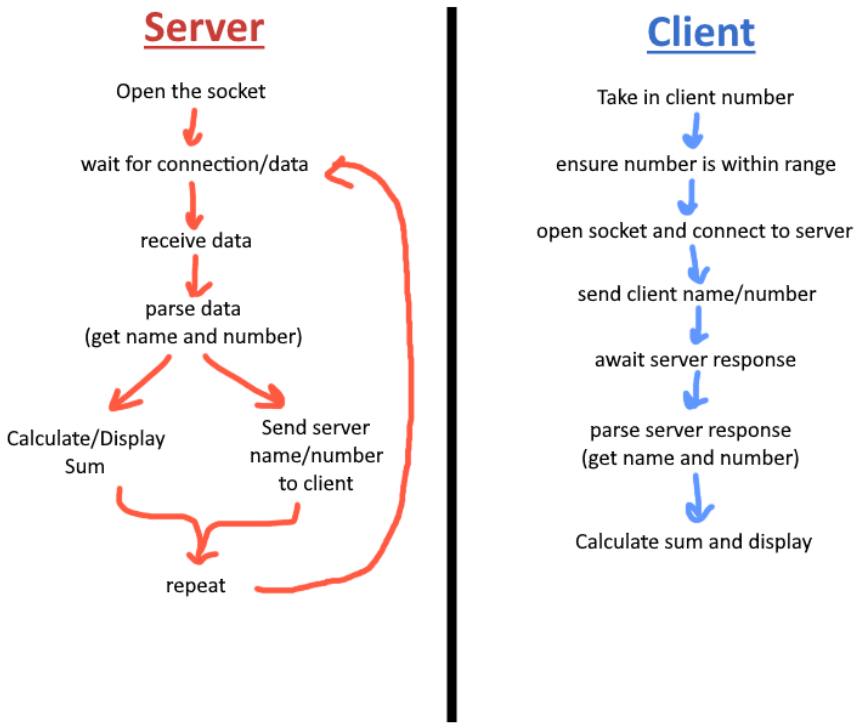
Networking Homework 4 Documentation

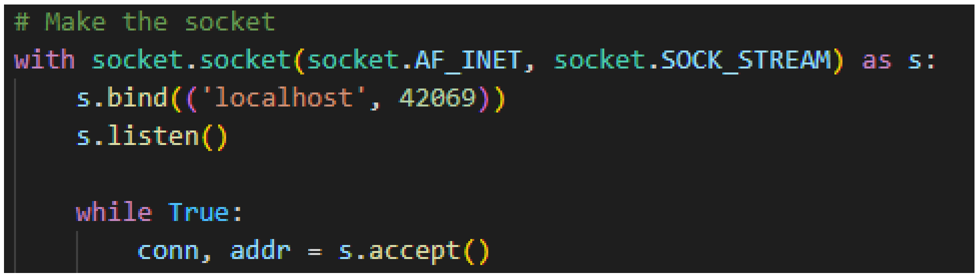
Problem: The goal here is to create a client-server application using sockets, where the client sends their name and a chosen number from 1 to 100 to the server, which then responds back to the client with its own name and number. Both the client and the server will display the received information and calculate/display the sum of the two integers. If the chosen number is out of range, the client will deny the input, and if the server gets an out-of-range number it will close itself.

Example Flowchart Structure:


Server
Open the socket
wait for connection/data
receive data
parse data (get name and number)
Calculate/Display Sum
Send server name/number to client
repeat
Client
Take in client number
ensure number is within range
open socket and connect to server
send client name/number
await server response
parse server response (get name and number)
Calculate sum and display

Code Breakdown:

On the server side, we need to be able to receive data from clients.



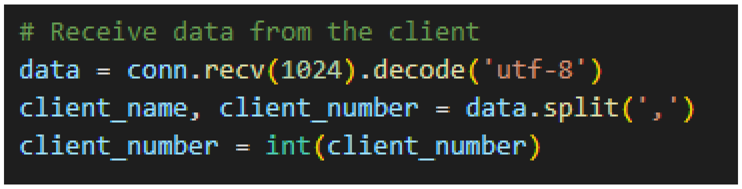
```
# Make the socket
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind(('localhost', 42069))
    s.listen()

    while True:
        conn, addr = s.accept()
```

The above code opens a socket, binds it to localhost with port 42069, and begins listening for connections. After that, a persistent while loop runs, which is halted by s.accept(). This halts execution until a connection has been established with a client, after which the socket object (conn) and the address connecting (addr) get set.

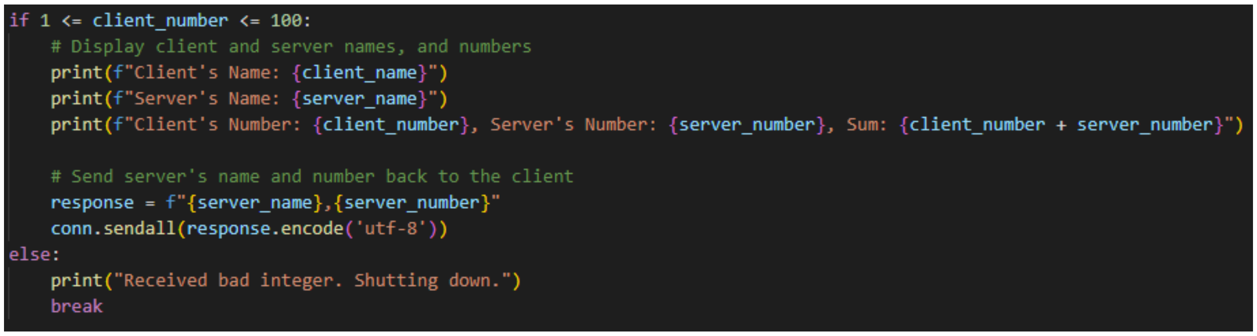
Next, data is received from the open connection.

Networking Homework 4 Documentation



```python
# Receive data from the client
data = conn.recv(1024).decode('utf-8')
client_name, client_number = data.split(',')
client_number = int(client_number)
```

The above code receives 1024 bytes from the buffer, decodes the data from bytes into a string, and splits the data into its name and number components.

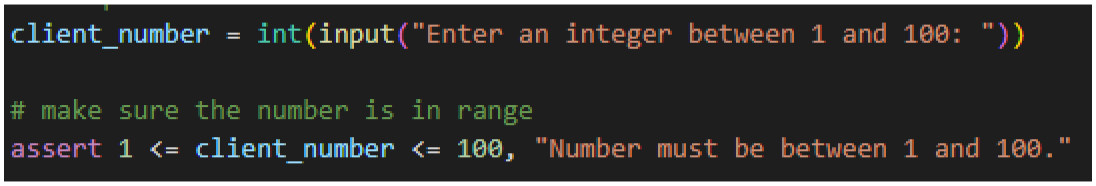


```python
if 1 <= client_number <= 100:
    # Display client and server names, and numbers
    print(f"Client's Name: {client_name}")
    print(f"Server's Name: {server_name}")
    print(f"Client's Number: {client_number}, Server's Number: {server_number}, Sum: {client_number + server_number}")

    # Send server's name and number back to the client
    response = f"{server_name},{server_number}"
    conn.sendall(response.encode('utf-8'))
else:
    print("Received bad integer. Shutting down.")
    break
```

Lastly, the server will display the client/server information and respond to the client with its name and number. This response gets encoded to bytes and is sent over the socket.

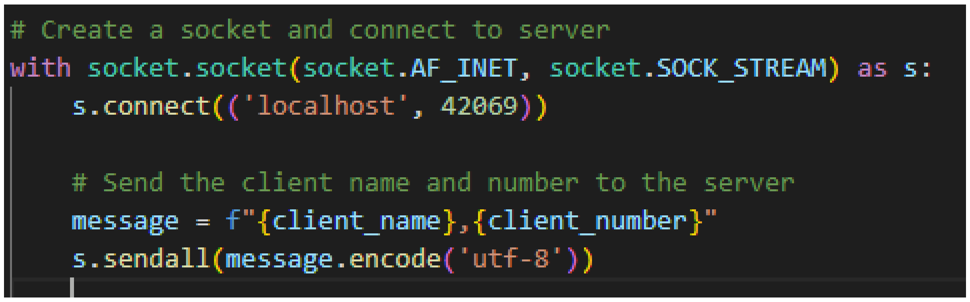
The client side works similarly.



```python
client_number = int(input("Enter an integer between 1 and 100: "))

# make sure the number is in range
assert 1 <= client_number <= 100, "Number must be between 1 and 100."
```

First, the user inputs a number, and the client checks to make sure it is within range.



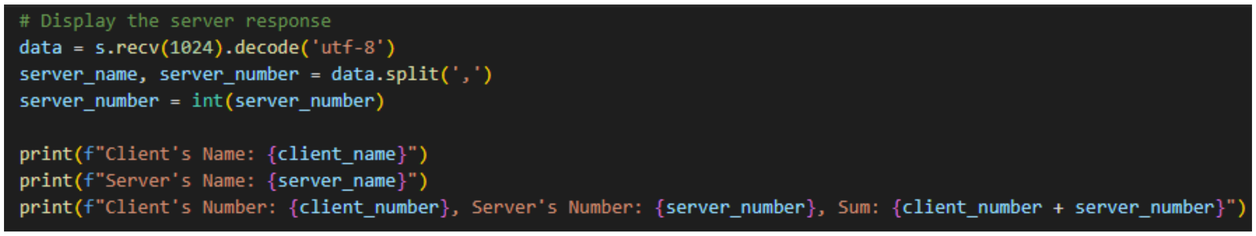
```python
# Create a socket and connect to server
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect(('localhost', 42069))

    # Send the client name and number to the server
    message = f"{client_name},{client_number}"
    s.sendall(message.encode('utf-8'))
```

Next, the client opens a socket and connects to localhost with the same port as the server. It constructs a message string, encodes it, and sends it through the socket.

Once the server has had time to process the client's message, the following occurs:

Networking Homework 4 Documentation



```python
# Display the server response
data = s.recv(1024).decode('utf-8')
server_name, server_number = data.split(',')
server_number = int(server_number)

print(f"Client's Name: {client_name}")
print(f"Server's Name: {server_name}")
print(f"Client's Number: {client_number}, Server's Number: {server_number}, Sum: {client_number + server_number}")
```

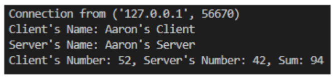
On the client side, we wait for response data to become available from the server, after which the data is decoded and split into its server name and number components. Just like with the server, we display the client name, server name, and both client/server numbers along with their sum.

Test Cases

Case 1

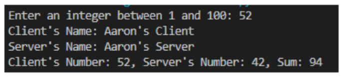
Input: 52

Server Output:



```
Connection from ('127.0.0.1', 56670)
Client's Name: Aaron's Client
Server's Name: Aaron's Server
Client's Number: 52, Server's Number: 42, Sum: 94
```

Client Output:

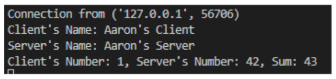


```
Enter an integer between 1 and 100: 52
Client's Name: Aaron's Client
Server's Name: Aaron's Server
Client's Number: 52, Server's Number: 42, Sum: 94
```

Case 2

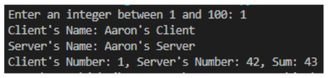
Input: 1

Server Output:



```
Connection from ('127.0.0.1', 56706)
Client's Name: Aaron's Client
Server's Name: Aaron's Server
Client's Number: 1, Server's Number: 42, Sum: 43
```

Client Output:



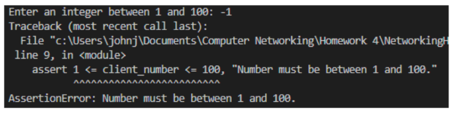
```
Enter an integer between 1 and 100: 1
Client's Name: Aaron's Client
Server's Name: Aaron's Server
Client's Number: 1, Server's Number: 42, Sum: 43
```

Case 3

Input: -1

Server Output:

Client Output:



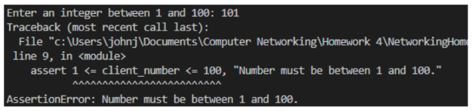
```
Enter an integer between 1 and 100: -1
Traceback (most recent call last):
  File "c:\Users\johnj\Documents\Computer Networking\Homework 4\NetworkingH
 line 9, in <module>
    assert 1 <= client_number <= 100, "Number must be between 1 and 100."
           ^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: Number must be between 1 and 100.
```

Case 4

Input: 101

Server Output:

Client Output:

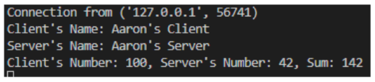


```
Enter an integer between 1 and 100: 101
Traceback (most recent call last):
  File "c:\Users\johnj\Documents\Computer Networking\Homework 4\NetworkingHome
 line 9, in <module>
    assert 1 <= client_number <= 100, "Number must be between 1 and 100."
           ^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError: Number must be between 1 and 100.
```

Case 5

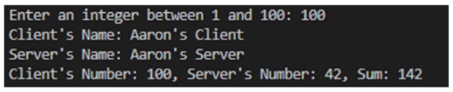
Input: 100

Server Output:



```
Connection from ('127.0.0.1', 56741)
Client's Name: Aaron's Client
Server's Name: Aaron's Server
Client's Number: 100, Server's Number: 42, Sum: 142
```

Client Output:

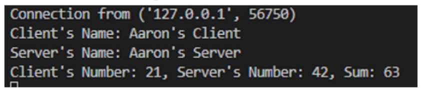


```
Enter an integer between 1 and 100: 100
Client's Name: Aaron's Client
Server's Name: Aaron's Server
Client's Number: 100, Server's Number: 42, Sum: 142
```

Case 6

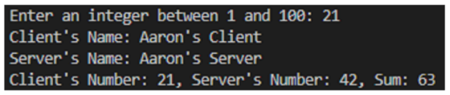
Input: 21

Server Output:



```
Connection from ('127.0.0.1', 56750)
Client's Name: Aaron's Client
Server's Name: Aaron's Server
Client's Number: 21, Server's Number: 42, Sum: 63
```

Client Output:



```
Enter an integer between 1 and 100: 21
Client's Name: Aaron's Client
Server's Name: Aaron's Server
Client's Number: 21, Server's Number: 42, Sum: 63
```