CODE DESCRIPTION

PROBLEM IDENTIFICATION

The main goal for this Computer Networking coursework was to create a UDP server – client network with at least 2 simultaneously working instances of the latter.

The server was to receive one arbitrary integer from each client and put them through the specified mathematical function to determine the winner. Finally the message should be sent to the winners' client. The important thing to notice is that the first client should be hanging waiting for the second one to send his integer.

IMPLEMENTATION OF THE SERVER

The server side of the solution was created using a socket server framework. The solution created can work with **two clients only** each time. Code provided can be easily modified to work with more than 2 clients but for the sake of simplicity it was kept to minimum requirements.

BELOW ARE THE MOST IMPORTANT BITS OF THE CODE.

```
# adding TWO clients to the game
if len(connected) < 1:
    connected.append(self.client_address)
    stored_client = connected[0]
elif connected[0] == stored_client:
    connected.append(self.client_address)
```

This bit of code ensures that 2 clients are connected and their IP addresses are kept in the **cache**. First client to join will be trigger the first **if** statement and be stored as the stored_client. The **elif** will always be equal to TRUE and make sure that the second clients IP is also stored.

```
# receiving messages
cid = self.request[0]

cint = self.request[1].recv(1024)
print("\nInteger received: ")
print(cint.decode("utf-8", "ignore"))
```

Receiving messages is done by requesting the function to call to itself for the message from the client. The **request [0]** is receiving the ID while **request [1]** is used to receive the integer from the client. Message needs to be decoded because it was coded into utf-8 prior to sending. Reason for that is that the socket interface does not support standard string encryption.

```
# maths
cint = int(cint.decode("utf-8", "ignore"))
count = (cint+count) % 10
```

This is the maths function which was described in the specification. The received message is turned into the integer, processed and stored in the **count** variable.

```
# make sure we have TWO clients connected
if len(connected) == 2:
```

This check is required to make sure the server has 2 clients connected **before** processing the integers.

```
# sending messages to client
if nOfWins < 3:
    if count == 0:
        self.request[1].sendto(win.encode("utf-8"), connected[0])
        print("\nWINNER IS:", stored_client)
        nOfWins += 1

        # clearing the cache
        del connected[:]
        stored_client = ('null', 0)
    else:
        self.request[1].sendto(lose.encode("utf-8"), connected[0])
        print("\n", stored_client, "lost!")

        # clearing the cache
        del connected[:]
        stored_client = ('null', 0)
else:
    self.request[1].sendto(stop.encode("utf-8"), connected[0])
    print("\nTHE GAME IS OVER!")

    # clearing the cache
    del connected[:]
    stored_client = ('null', 0)
```

This is executed after the above **if** returns true.

First, number of winning messages are checked. If they are below specified margin the code checks the **value** in the **count** variable. In case **count** is equal to zero the program sends out the winning message (encoded to utf-8 to fit the interface) to the stored client. Number of wins is incremented by 1.

After this the cache is cleared by deleting the values in the **connected []** list and resetting the **stored clients'** value. Else the alternative message is send out and cache is cleared again.

In case the number of winning messages is **more than 3** the "Game over!" message is send to the client and cache is cleared again.

```
# serve forever
if __name__ == "__main__":
    server = socketserver.UDPServer(server_address, MyUDPHandler,
bind_and_activate=True)
    server.serve_forever()
```

This bit of code is set **outside** the class to make sure it works **all the time** as opposed to working only when the client in connected. The **if** statement is chosen so it will be equal to TRUE and keep the server running.

IMPLEMENTATION OF THE CLIENT

Client was implemented using the standard socket datagram communication.

```
# prompt an ID from the user
clientID = input('Please enter your ID (1-100): ')

# validation
while int(clientID) < 1 or int(clientID) > 100:
    print('\nClientID value is unacceptable please try again')
    clientID = input('Please enter your ID (1-100): ')
```

First, the **client ID** is stored and checked against the validation rules.

```
# prompt and int from the user
clientINT = input('Please enter an integer of your choice: ')
```

Then the client inputted **integer** is also stored.

```
# sending 2 packets
sock.sendto(clientID.encode('utf-8'), server_address)
sock.sendto(clientINT.encode('utf-8'), server_address)
```

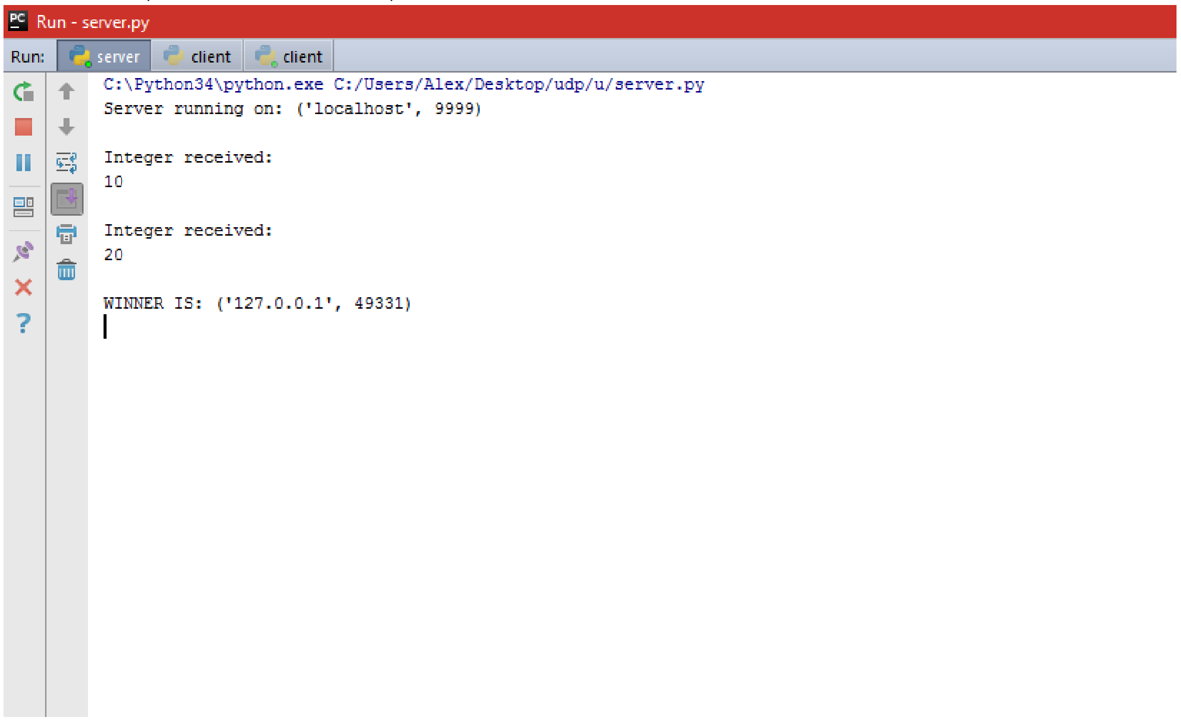
After this 2 packets are encoded (utf-8) and sent to the server.

```
# receiving packets
while state == 0:
    print("\nwaiting to receive...")
    received = sock.recv(4096)
    received = received.decode("utf-8", "ignore")
    if received == "You win!" or received == "You lose!":

        # closing the client
        state += 1
        print("\nTHE RESULT IS: ", received)
        sock.close()
    elif received == "Game over!":
        state += 1
        print("\nGAME IS OVER!")
        sock.close()
```

Finally, the above bit of code only runs while the state of the client is equal to 0. Messages are being constantly received or awaited to be received. In case the messages are triggering any of the 3 messages specified, the program displays them to the client. After this the state of the program is set to 1 (offline) and **socket** is closed.

SCREENSHOTS OF THE WORKING SOLUTION

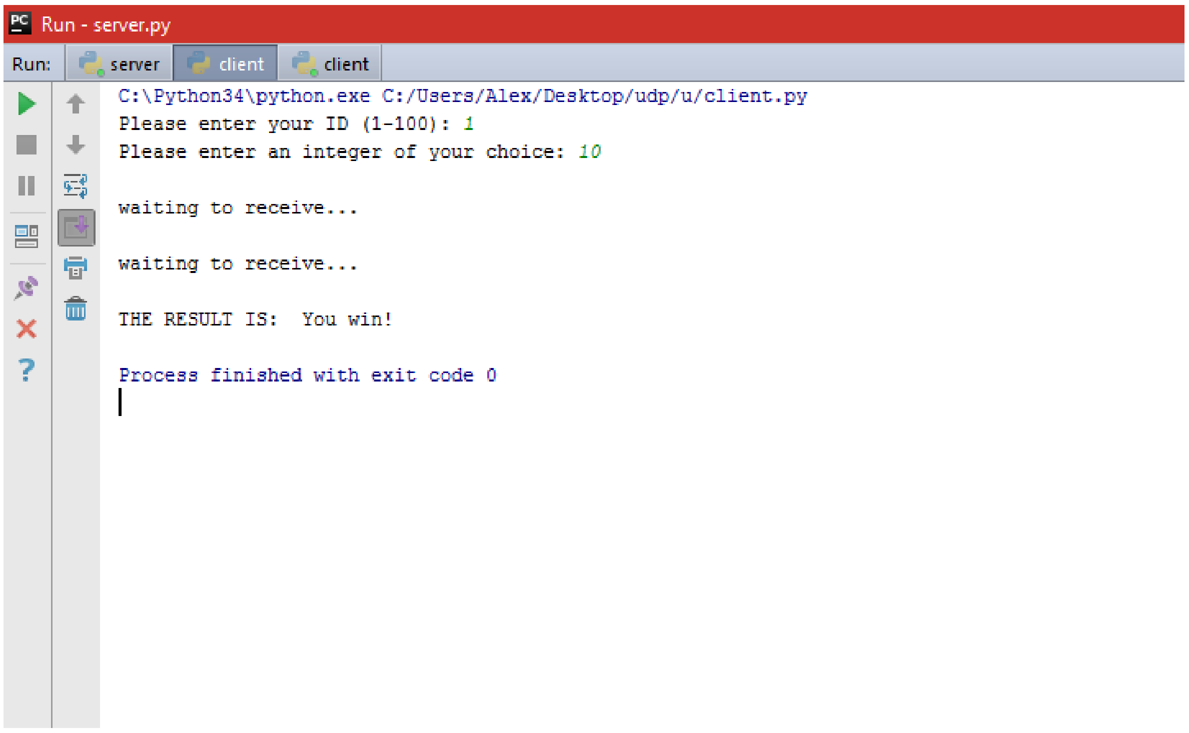
SERVER (WIN MESSAGE)



```
C:\Python34\python.exe C:/Users/Alex/Desktop/udp/u/server.py
Server running on: ('localhost', 9999)

Integer received:
10

Integer received:
20

WINNER IS: ('127.0.0.1', 49331)
```

Server shows 2 integers received and the winners' IP address.

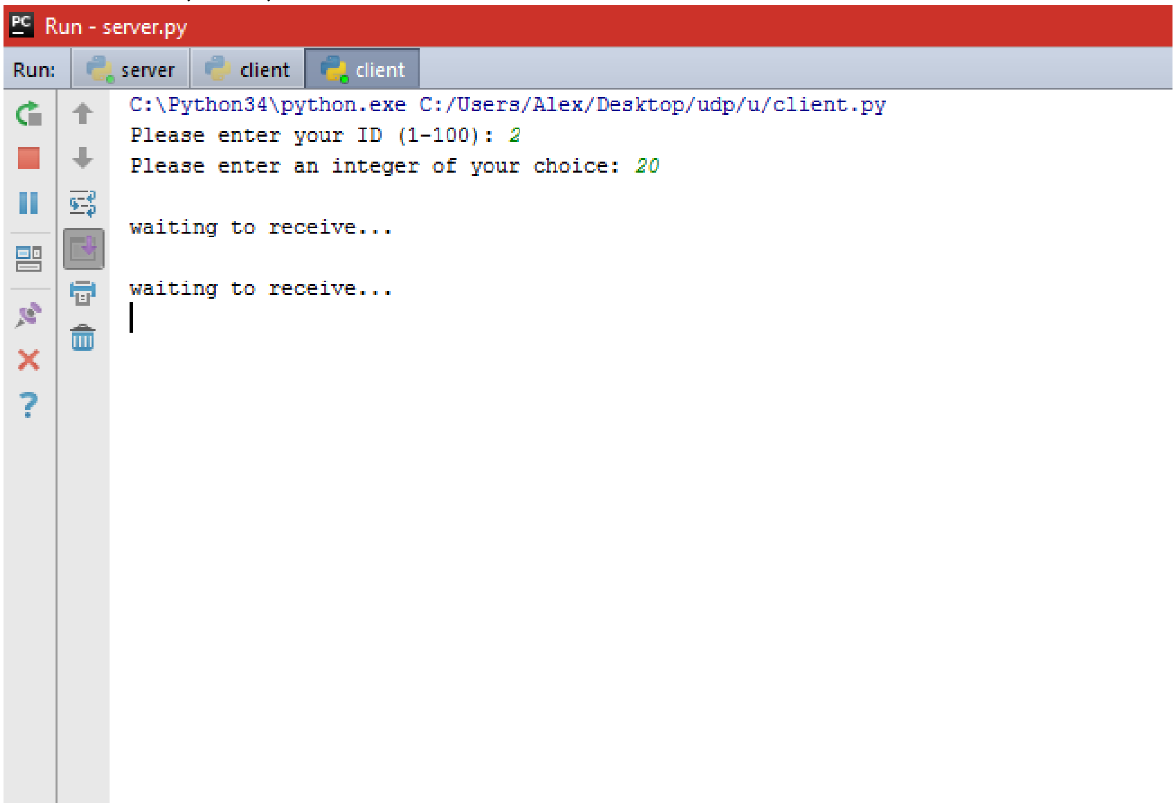
CLIENT #1 (WIN MESSAGE)



```
C:\Python34\python.exe C:/Users/Alex/Desktop/udp/u/client.py
Please enter your ID (1-100): 1
Please enter an integer of your choice: 10

waiting to receive...

waiting to receive...

THE RESULT IS:  You win!

Process finished with exit code 0
```

Client #1 inputs 2 integers. After the wait the result is displayed.

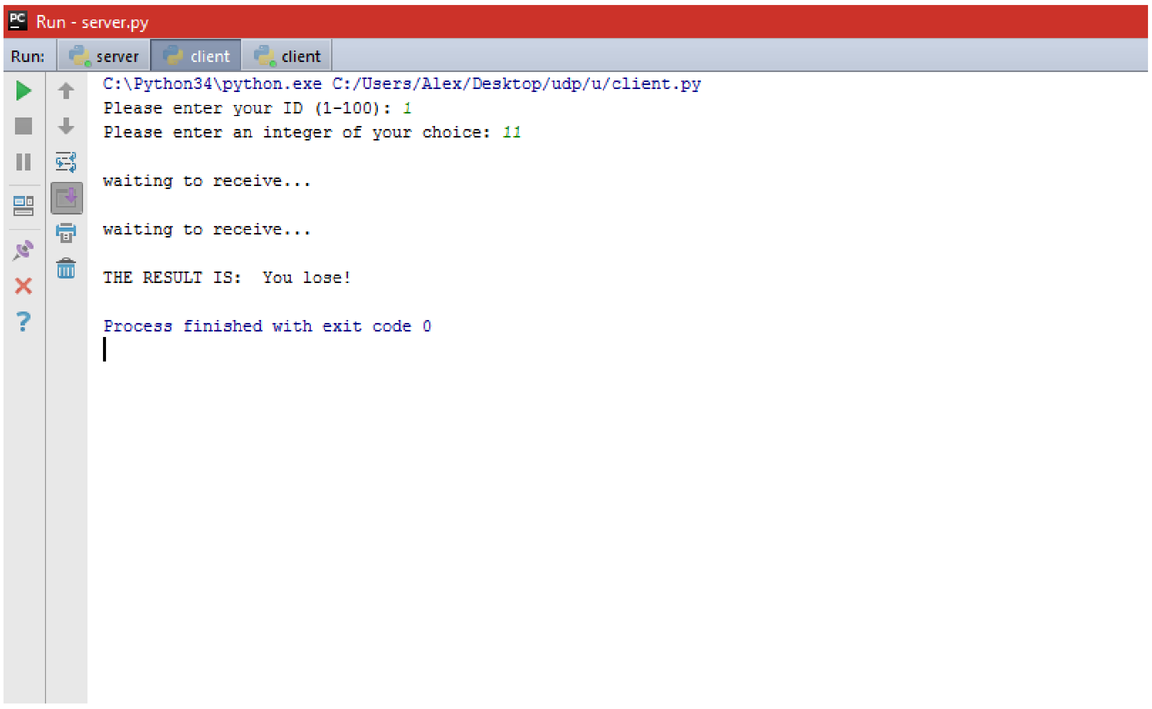
CLIENT #2 (IDLE)



```
C:\Python34\python.exe C:/Users/Alex/Desktop/udp/u/client.py
Please enter your ID (1-100): 2
Please enter an integer of your choice: 20

waiting to receive...

waiting to receive...
```

Client #2 is kept idle as he is not supposed to receive any messages.

CLIENT #1 (LOSS MESSAGE)

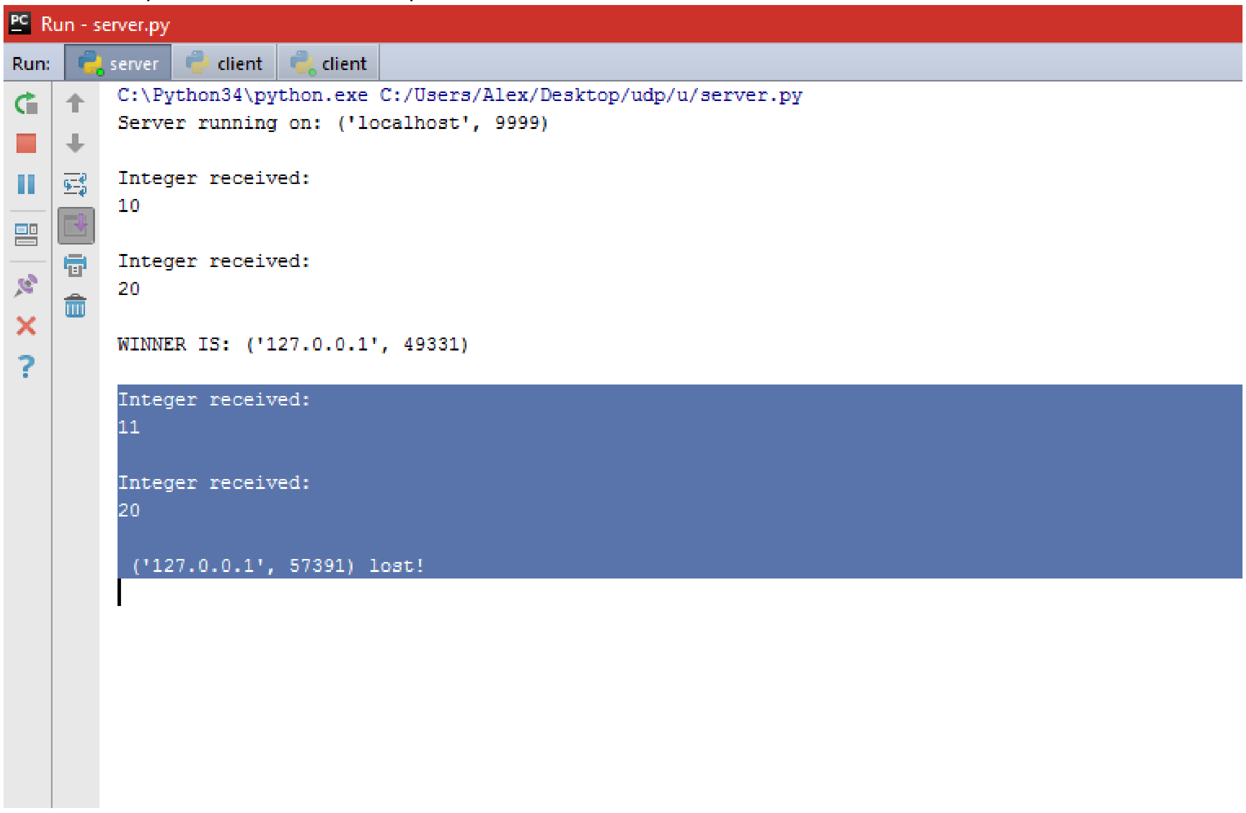


```
C:\Python34\python.exe C:/Users/Alex/Desktop/udp/u/client.py
Please enter your ID (1-100): 1
Please enter an integer of your choice: 11

waiting to receive...

waiting to receive...

THE RESULT IS:  You lose!

Process finished with exit code 0
```

The loss is calculated because the **count** value is equal to 1 in this case.

(Other client sent 20 as the integer)

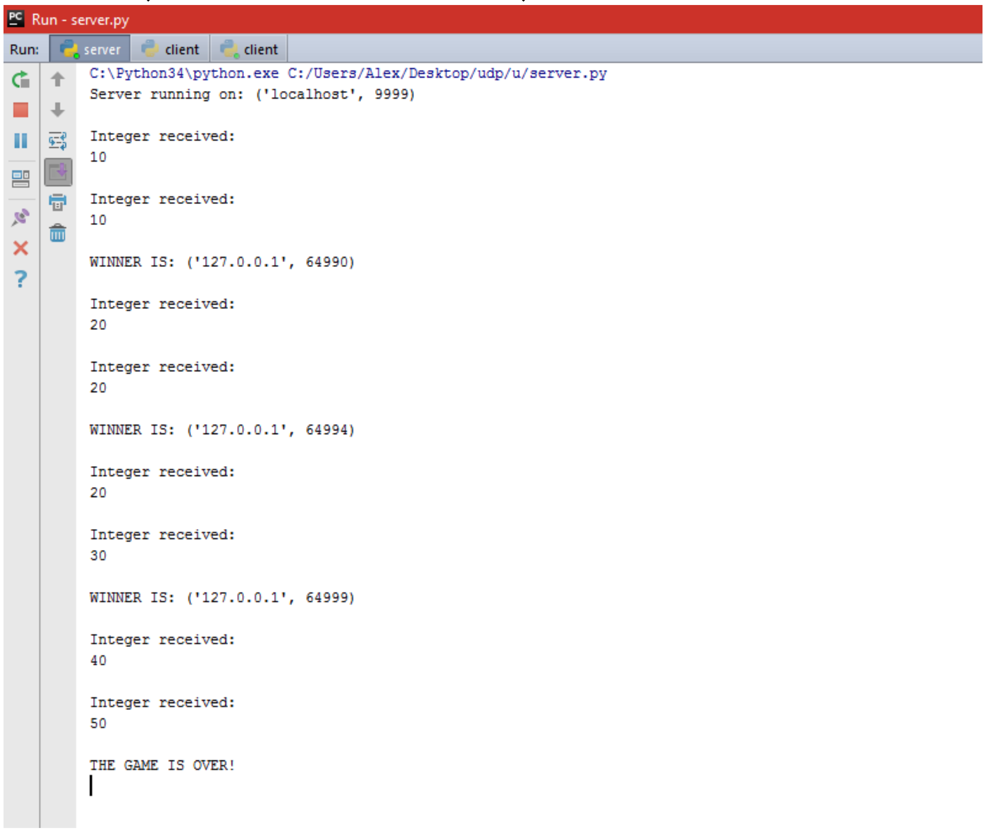
SERVER (LOSS MESSAGE)



```
C:\Python34\python.exe C:/Users/Alex/Desktop/udp/u/server.py
Server running on: ('localhost', 9999)

Integer received:
10

Integer received:
20

WINNER IS: ('127.0.0.1', 49331)

Integer received:
11

Integer received:
20

 ('127.0.0.1', 57391) lost!
```

Server displays the IP of the client that lost.

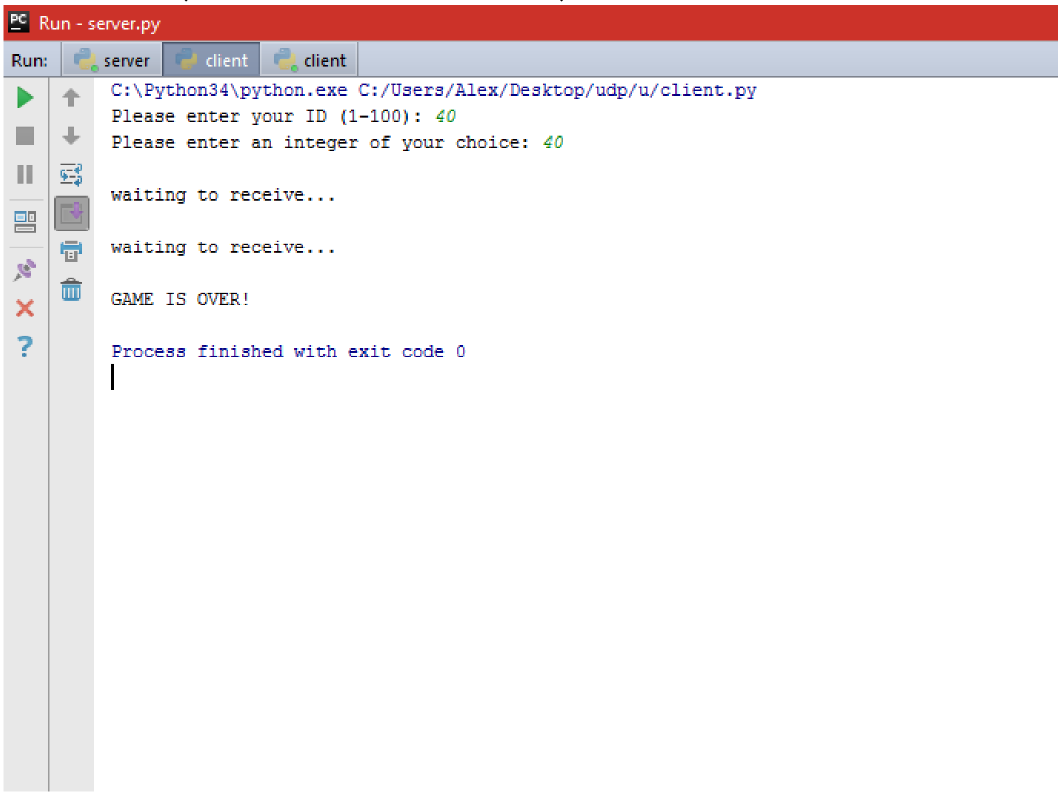
SERVER (GAME OVER MESSAGE)



```
C:\Python34\python.exe C:/Users/Alex/Desktop/udp/u/server.py
Server running on: ('localhost', 9999)

Integer received:
10

Integer received:
10

WINNER IS: ('127.0.0.1', 64990)

Integer received:
20

Integer received:
20

WINNER IS: ('127.0.0.1', 64994)

Integer received:
20

Integer received:
30

WINNER IS: ('127.0.0.1', 64999)

Integer received:
40

Integer received:
50

THE GAME IS OVER!
```

Server displays the "GAME OVER" message **after** 3 winning messages have been sent.

CLIENT #1 (GAME OVER MESSAGE)



```
C:\Python34\python.exe C:/Users/Alex/Desktop/udp/u/client.py
Please enter your ID (1-100): 40
Please enter an integer of your choice: 40

waiting to receive...

waiting to receive...

GAME IS OVER!

Process finished with exit code 0
```

Client #1 receives the "GAME OVER" message. After trying to receive the answer.

(Other client sent number 50, so should have displayed the winning message otherwise)