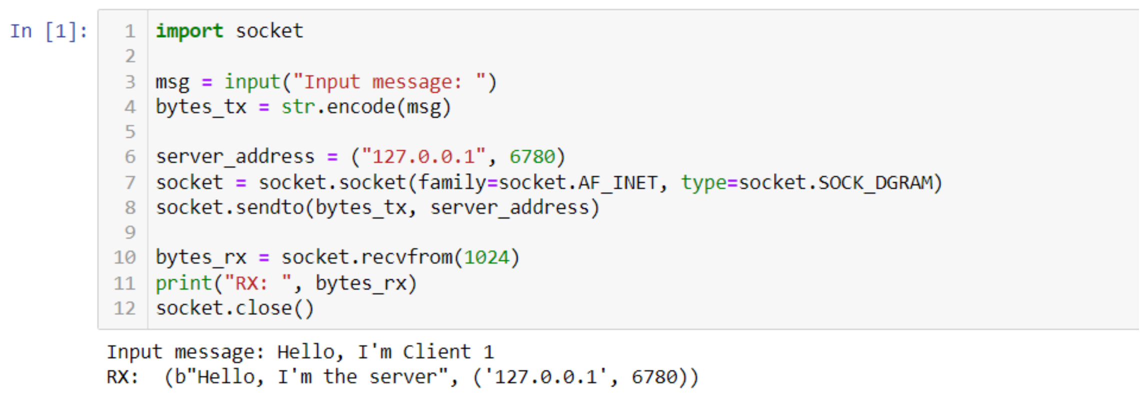
Sistemas Distribuidos
Práctica 1

Nicolás Corsini

UDP-1:

UDP-1.1

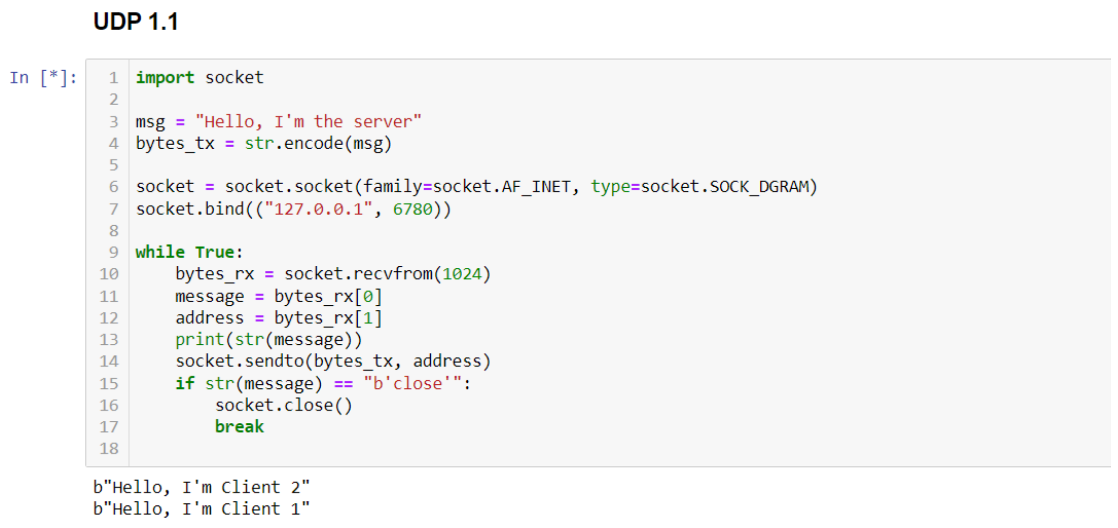
Código del cliente:



```
In [1]:
import socket

msg = input("Input message: ")
bytes_tx = str.encode(msg)

server_address = ("127.0.0.1", 6780)
socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
socket.sendto(bytes_tx, server_address)

bytes_rx = socket.recvfrom(1024)
print("RX: ", bytes_rx)
socket.close()
```

```
Input message: Hello, I'm Client 1
RX:  (b"Hello, I'm the server", ('127.0.0.1', 6780))
```

Código del servidor:



UDP 1.1

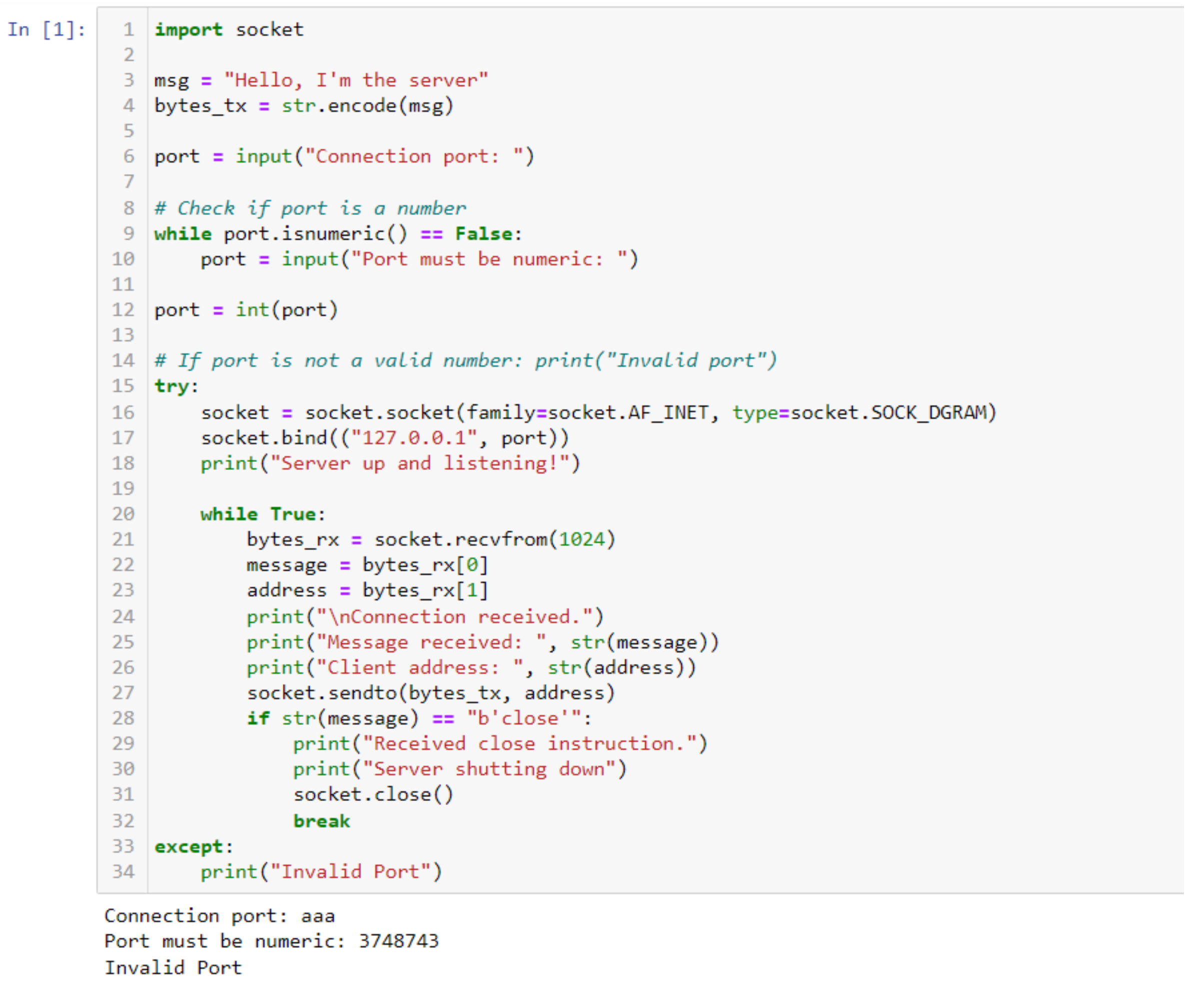
```
In [*]:
import socket

msg = "Hello, I'm the server"
bytes_tx = str.encode(msg)

socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
socket.bind(("127.0.0.1", 6780))

while True:
    bytes_rx = socket.recvfrom(1024)
    message = bytes_rx[0]
    address = bytes_rx[1]
    print(str(message))
    socket.sendto(bytes_tx, address)
    if str(message) == "b'close'":
        socket.close()
        break

```

```
b"Hello, I'm Client 2"
b"Hello, I'm Client 1"
```

Como se puede observar, el servidor arranca y se queda esperando un número ilimitado de respuestas. Al ejecutar 2 clientes diferentes, el servidor recibe el envío de los dos y les responde.

UDP-1.2

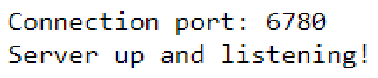
Código del servidor:



```
In [1]:
import socket

msg = "Hello, I'm the server"
bytes_tx = str.encode(msg)

port = input("Connection port: ")

# Check if port is a number
while port.isnumeric() == False:
    port = input("Port must be numeric: ")

port = int(port)

# If port is not a valid number: print("Invalid port")
try:
    socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
    socket.bind(("127.0.0.1", port))
    print("Server up and listening!")

    while True:
        bytes_rx = socket.recvfrom(1024)
        message = bytes_rx[0]
        address = bytes_rx[1]
        print("\nConnection received.")
        print("Message received: ", str(message))
        print("Client address: ", str(address))
        socket.sendto(bytes_tx, address)
        if str(message) == "b'close'":
            print("Received close instruction.")
            print("Server shutting down")
            socket.close()
            break
except:
    print("Invalid Port")
```

```
Connection port: aaa
Port must be numeric: 3748743
Invalid Port
```

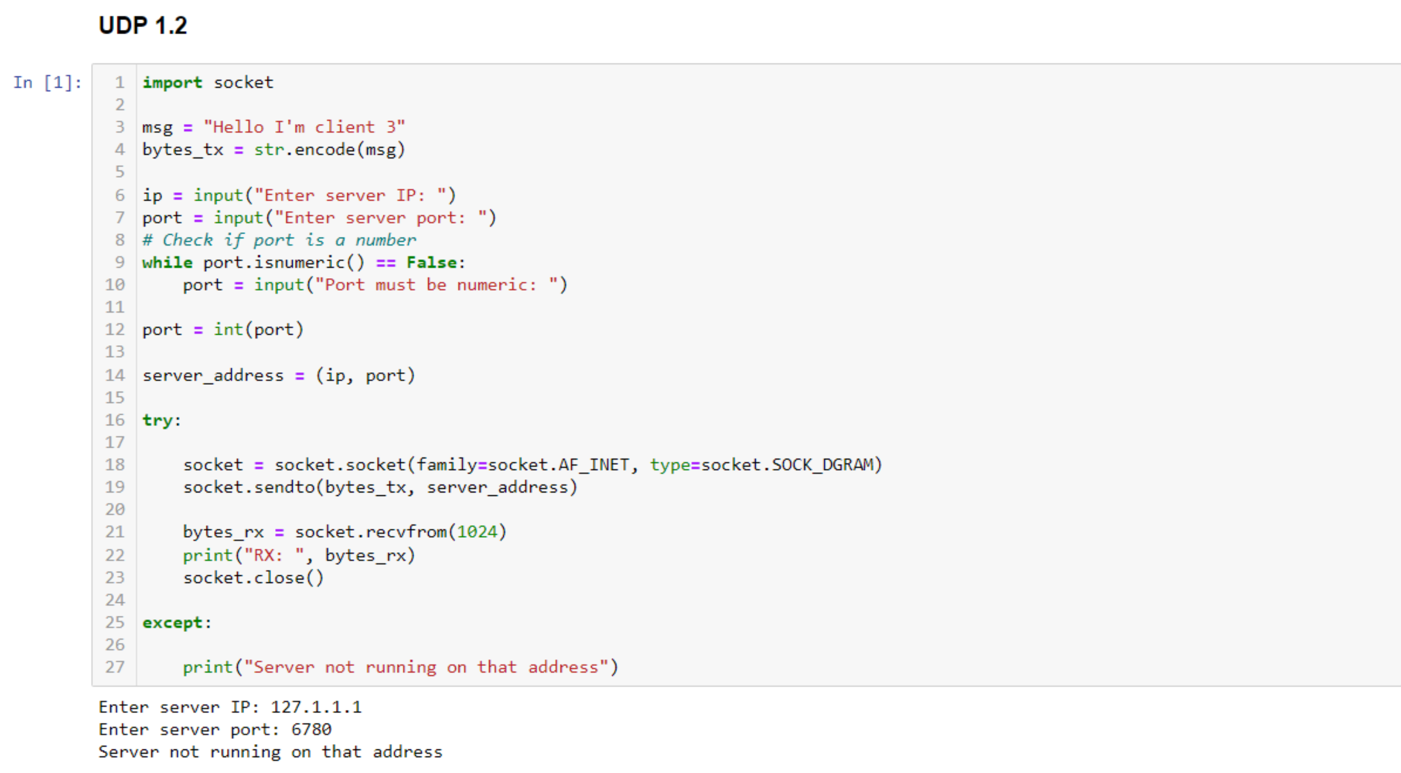
El servidor comprueba que el puerto especificado es numérico y que es válido. Si el puerto es válido muestra:



```
Connection port: 6780
Server up and listening!
```

Además, si el cliente le mandase el mensaje “close”, el servidor se pararía automáticamente.

Código del cliente:



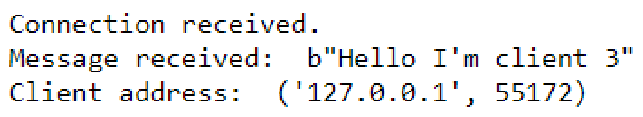
UDP 1.2

```
In [1]:
import socket

msg = "Hello I'm client 3"
bytes_tx = str.encode(msg)

ip = input("Enter server IP: ")
port = input("Enter server port: ")
# Check if port is a number
while port.isnumeric() == False:
    port = input("Port must be numeric: ")

port = int(port)

server_address = (ip, port)

try:

    socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
    socket.sendto(bytes_tx, server_address)

    bytes_rx = socket.recvfrom(1024)
    print("RX: ", bytes_rx)
    socket.close()

except:

    print("Server not running on that address")
```

```
Enter server IP: 127.1.1.1
Enter server port: 6780
Server not running on that address
```

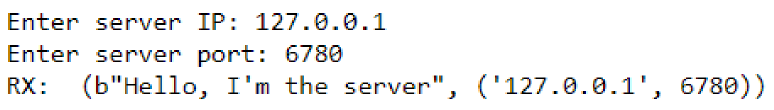
Si no encuentra al servidor, muestra el mensaje de “Server not running on that address”.

Si la address es correcta, el cliente manda el mensaje y el servidor lo recibe:



```
Connection received.
Message received:  b"Hello I'm client 3"
Client address:  ('127.0.0.1', 55172)
```

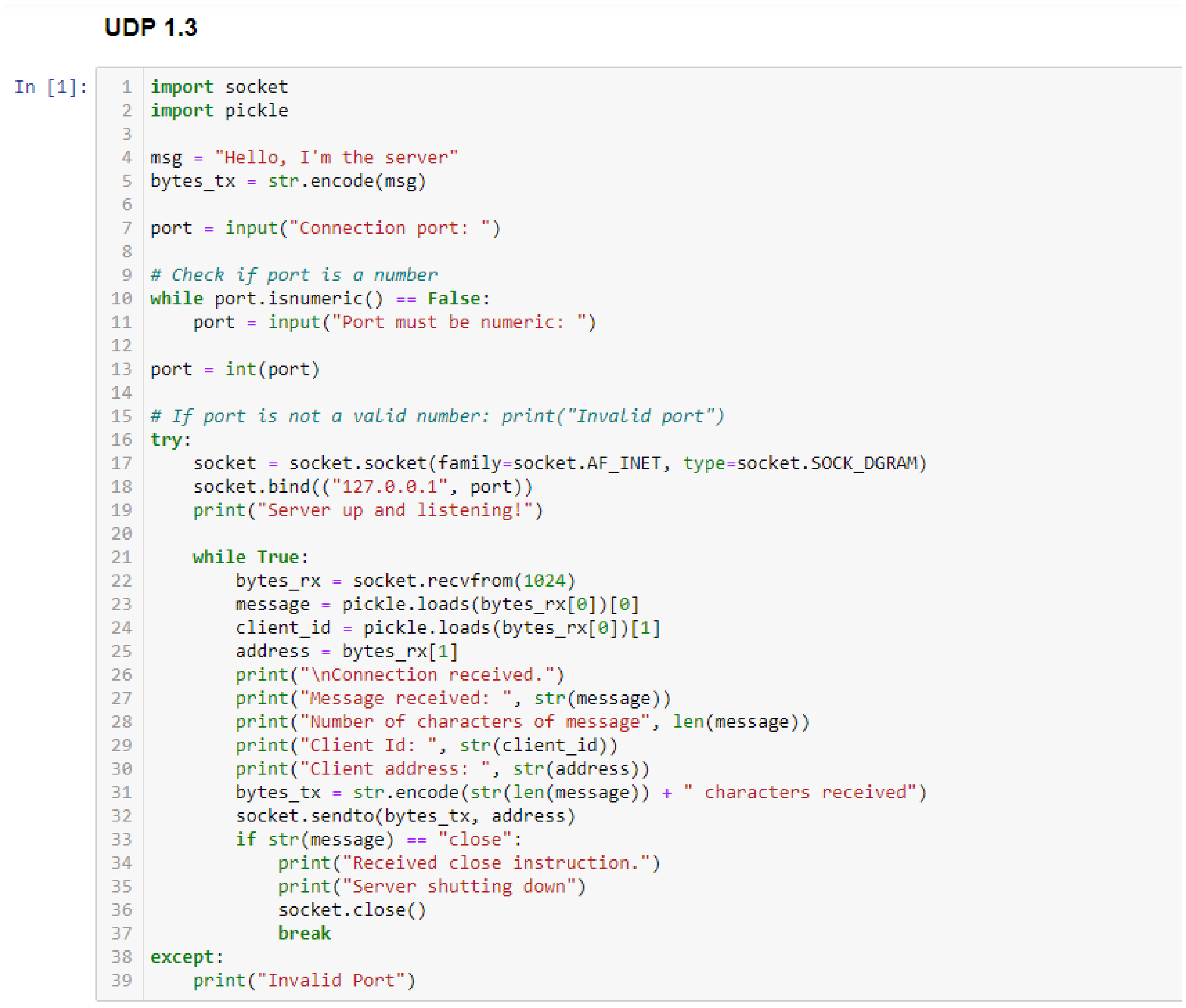
En el cliente se muestra la respuesta del servidor:



```
Enter server IP: 127.0.0.1
Enter server port: 6780
RX:  (b"Hello, I'm the server", ('127.0.0.1', 6780))
```

UDP-1.3

Código del servidor:



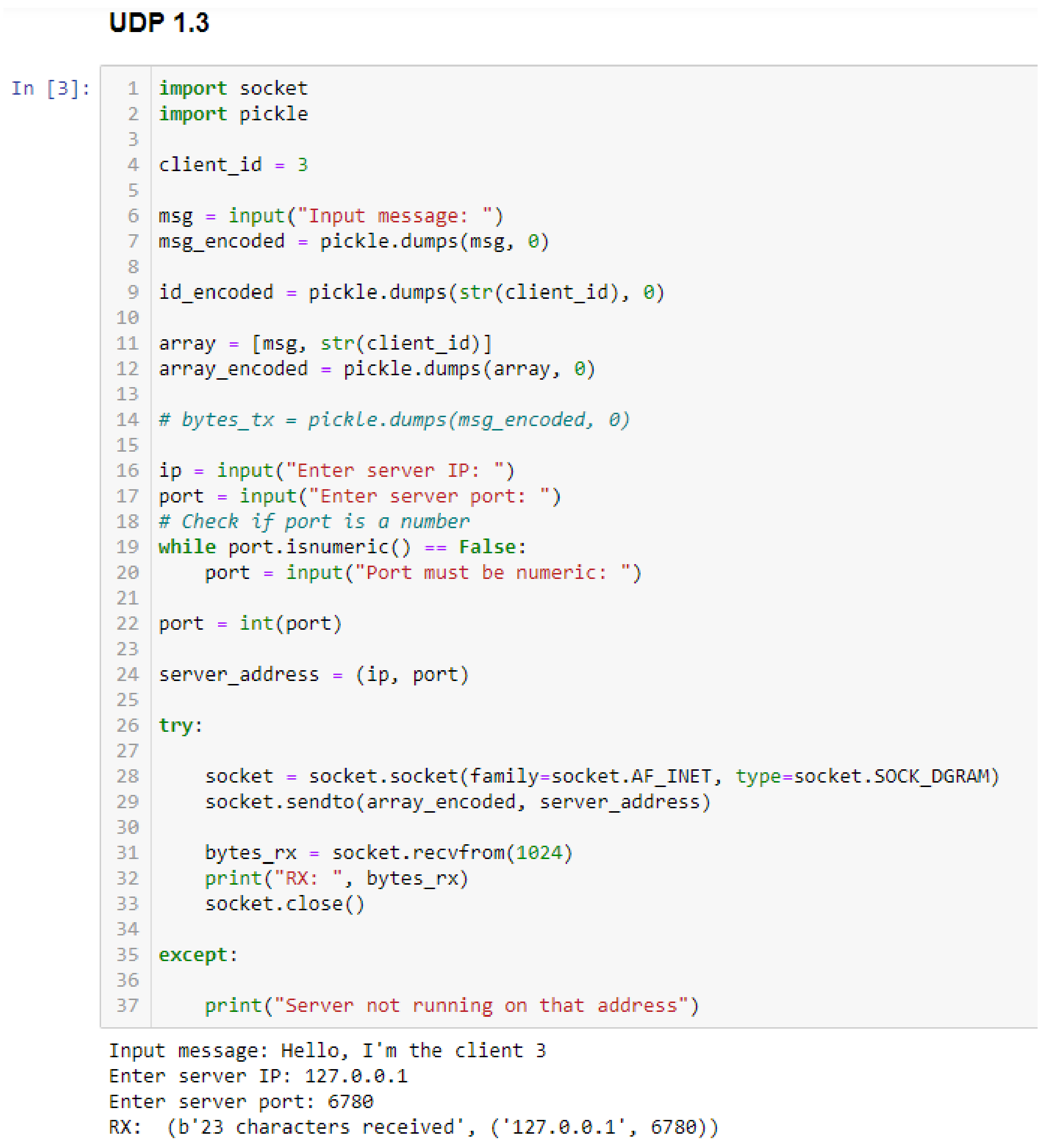
UDP 1.3

In [1]:

```python
import socket
import pickle

msg = "Hello, I'm the server"
bytes_tx = str.encode(msg)

port = input("Connection port: ")

# Check if port is a number
while port.isnumeric() == False:
    port = input("Port must be numeric: ")

port = int(port)

# If port is not a valid number: print("Invalid port")
try:
    socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
    socket.bind(("127.0.0.1", port))
    print("Server up and listening!")

    while True:
        bytes_rx = socket.recvfrom(1024)
        message = pickle.loads(bytes_rx[0])[0]
        client_id = pickle.loads(bytes_rx[0])[1]
        address = bytes_rx[1]
        print("\nConnection received.")
        print("Message received: ", str(message))
        print("Number of characters of message", len(message))
        print("Client Id: ", str(client_id))
        print("Client address: ", str(address))
        bytes_tx = str.encode(str(len(message)) + " characters received")
        socket.sendto(bytes_tx, address)
        if str(message) == "close":
            print("Received close instruction.")
            print("Server shutting down")
            socket.close()
            break
except:
    print("Invalid Port")
```

El servidor recibe un array empaquetado con pickle y lo desempaqueta guardando el id del cliente y el texto del mensaje. Le manda al cliente el número de caracteres de su mensaje.

Código del cliente:



UDP 1.3

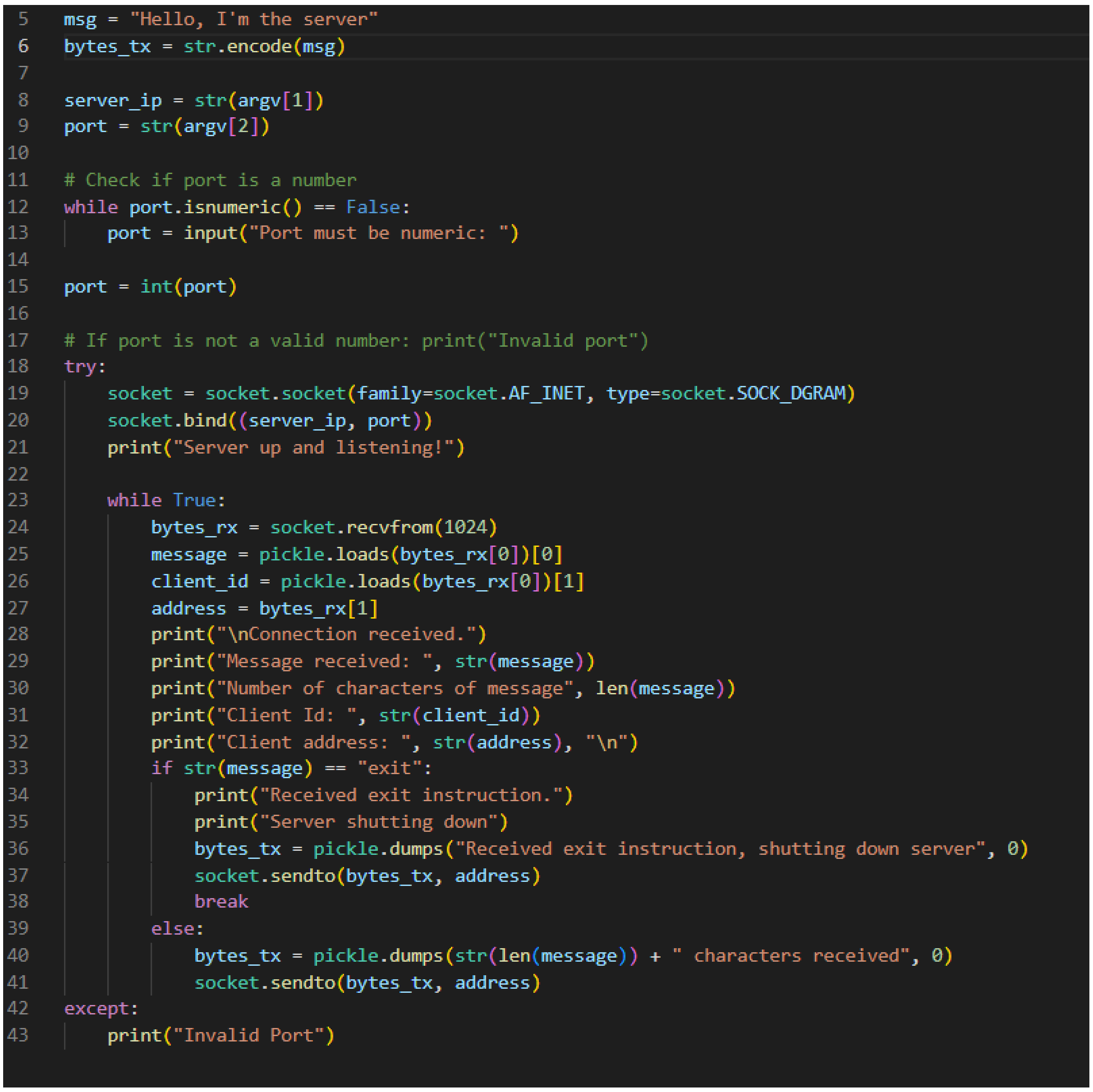
In [3]:
```python
import socket
import pickle

client_id = 3

msg = input("Input message: ")
msg_encoded = pickle.dumps(msg, 0)

id_encoded = pickle.dumps(str(client_id), 0)

array = [msg, str(client_id)]
array_encoded = pickle.dumps(array, 0)

# bytes_tx = pickle.dumps(msg_encoded, 0)

ip = input("Enter server IP: ")
port = input("Enter server port: ")
# Check if port is a number
while port.isnumeric() == False:
    port = input("Port must be numeric: ")

port = int(port)

server_address = (ip, port)

try:

    socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
    socket.sendto(array_encoded, server_address)

    bytes_rx = socket.recvfrom(1024)
    print("RX: ", bytes_rx)
    socket.close()

except:

    print("Server not running on that address")
```

```
Input message: Hello, I'm the client 3
Enter server IP: 127.0.0.1
Enter server port: 6780
RX:  (b'23 characters received', ('127.0.0.1', 6780))
```

El cliente empaqueta un array que contiene el mensaje y su Id y lo manda al servidor.

UDP-1.4

Código del servidor:



```python
msg = "Hello, I'm the server"
bytes_tx = str.encode(msg)

server_ip = str(argv[1])
port = str(argv[2])

# Check if port is a number
while port.isnumeric() == False:
    port = input("Port must be numeric: ")

port = int(port)

# If port is not a valid number: print("Invalid port")
try:
    socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
    socket.bind((server_ip, port))
    print("Server up and listening!")

    while True:
        bytes_rx = socket.recvfrom(1024)
        message = pickle.loads(bytes_rx[0])[0]
        client_id = pickle.loads(bytes_rx[0])[1]
        address = bytes_rx[1]
        print("\nConnection received.")
        print("Message received: ", str(message))
        print("Number of characters of message", len(message))
        print("Client Id: ", str(client_id))
        print("Client address: ", str(address), "\n")
        if str(message) == "exit":
            print("Received exit instruction.")
            print("Server shutting down")
            bytes_tx = pickle.dumps("Received exit instruction, shutting down server", 0)
            socket.sendto(bytes_tx, address)
            break
        else:
            bytes_tx = pickle.dumps(str(len(message)) + " characters received", 0)
            socket.sendto(bytes_tx, address)
except:
    print("Invalid Port")
```

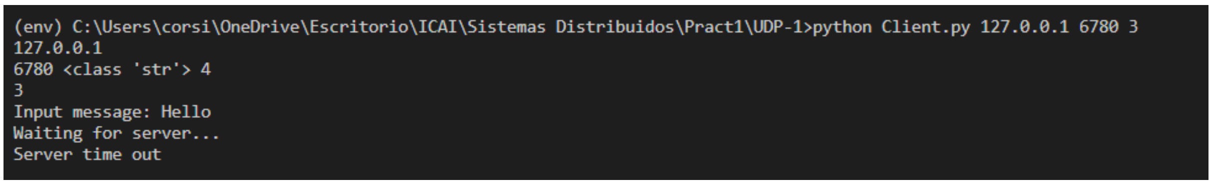
El código del servidor es el mismo pero pasado a VS para poder llamarlo más cómodamente por consola. En este caso, la IP y el puerto se obtienen de la línea de comandos.

Código del cliente:

```
import socket
import pickle
from sys import argv
import time

server_ip = argv[1]
port = str(argv[2])
client_id = argv[3]

print(server_ip)
print(port, type(port), len(port))
print(client_id)


id_encoded = pickle.dumps(str(client_id), 0)


# bytes_tx = pickle.dumps(msg_encoded, 0)

# Check if port is a number
while port.isnumeric() == False:
    port = input("Port must be numeric: ")

port = int(port)

server_address = (server_ip, port)


seconds = 10
# time.clock()

connection = True
elapsed = 0
socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

while connection==True:
    connection = False
    msg = input("Input message: ")
    print("Waiting for server...")
    start = time.time()
    msg_encoded = pickle.dumps(msg, 0)
    array = [msg, str(client_id)]
    array_encoded = pickle.dumps(array, 0)
    while not connection:
        try:
            socket.sendto(array_encoded, server_address)
            bytes_rx = socket.recvfrom(1024)
            msg_received = pickle.loads(bytes_rx[0])
```

```python
            connection = True
        except:
            connection = False
            time.sleep(1)
            elapsed = time.time() - start
            if elapsed > seconds:
                print("Server time out")
                break

    if connection == True:
        try:
            print("Response: ", msg_received, "\n")
            # socket.close()
        except:
            # socket.close()
            print("Error")
    if msg == "exit":
        socket.close()
        break
```

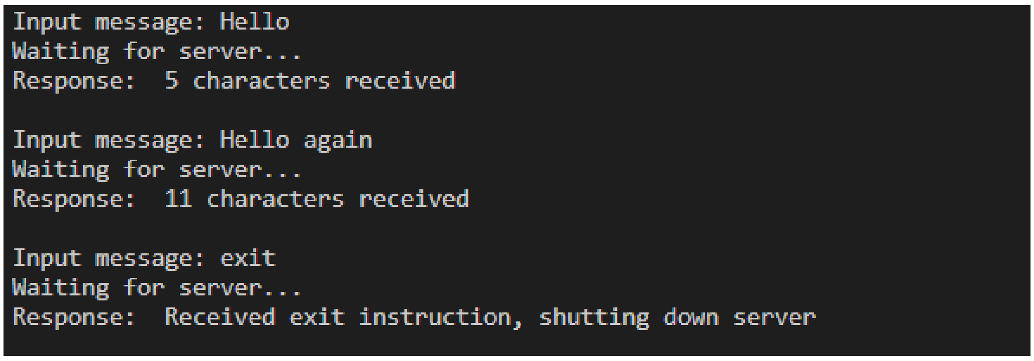
Si el cliente no recibe conexión con el servidor durante 10 segundos muestra un mensaje de "Time out":



```
(env) C:\Users\corsi\OneDrive\Escritorio\ICAI\Sistemas Distribuidos\Pract1\UDP-1>python Client.py 127.0.0.1 6780 3
127.0.0.1
6780 <class 'str'> 4
3
Input message: Hello
Waiting for server...
Server time out
```

Esto permite iniciar el cliente antes que el servidor, pues se tiene una ventana de 10 segundos para hacerlo.

Si el servidor está corriendo correctamente:



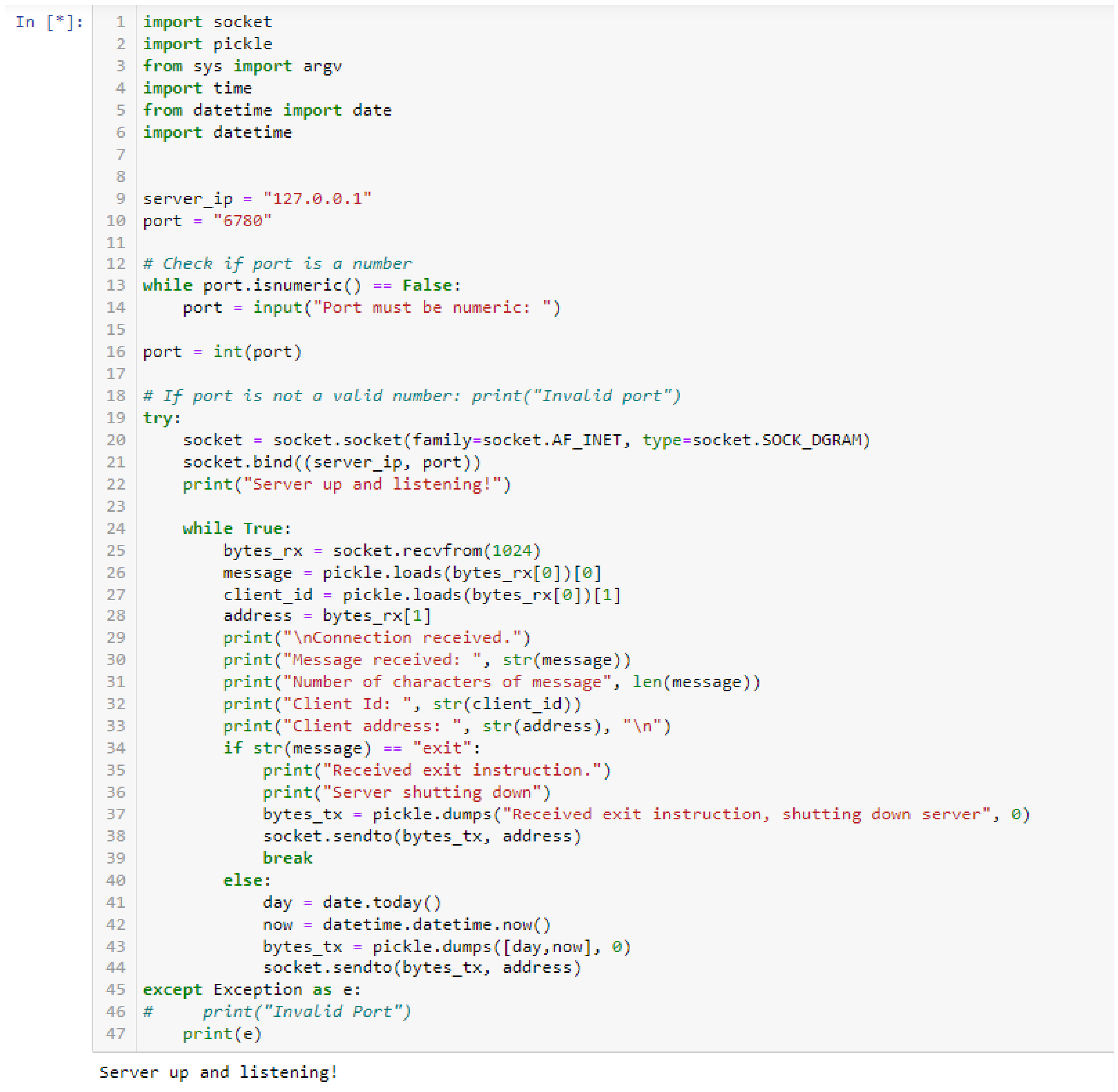
```
Input message: Hello
Waiting for server...
Response:  5 characters received

Input message: Hello again
Waiting for server...
Response:  11 characters received

Input message: exit
Waiting for server...
Response:  Received exit instruction, shutting down server
```

El cliente puede mandar tantos mensajes como quiera. Si manda 'exit', hará que se cierre el servidor.

UDP-2:

UDP-2.1

Código del servidor:



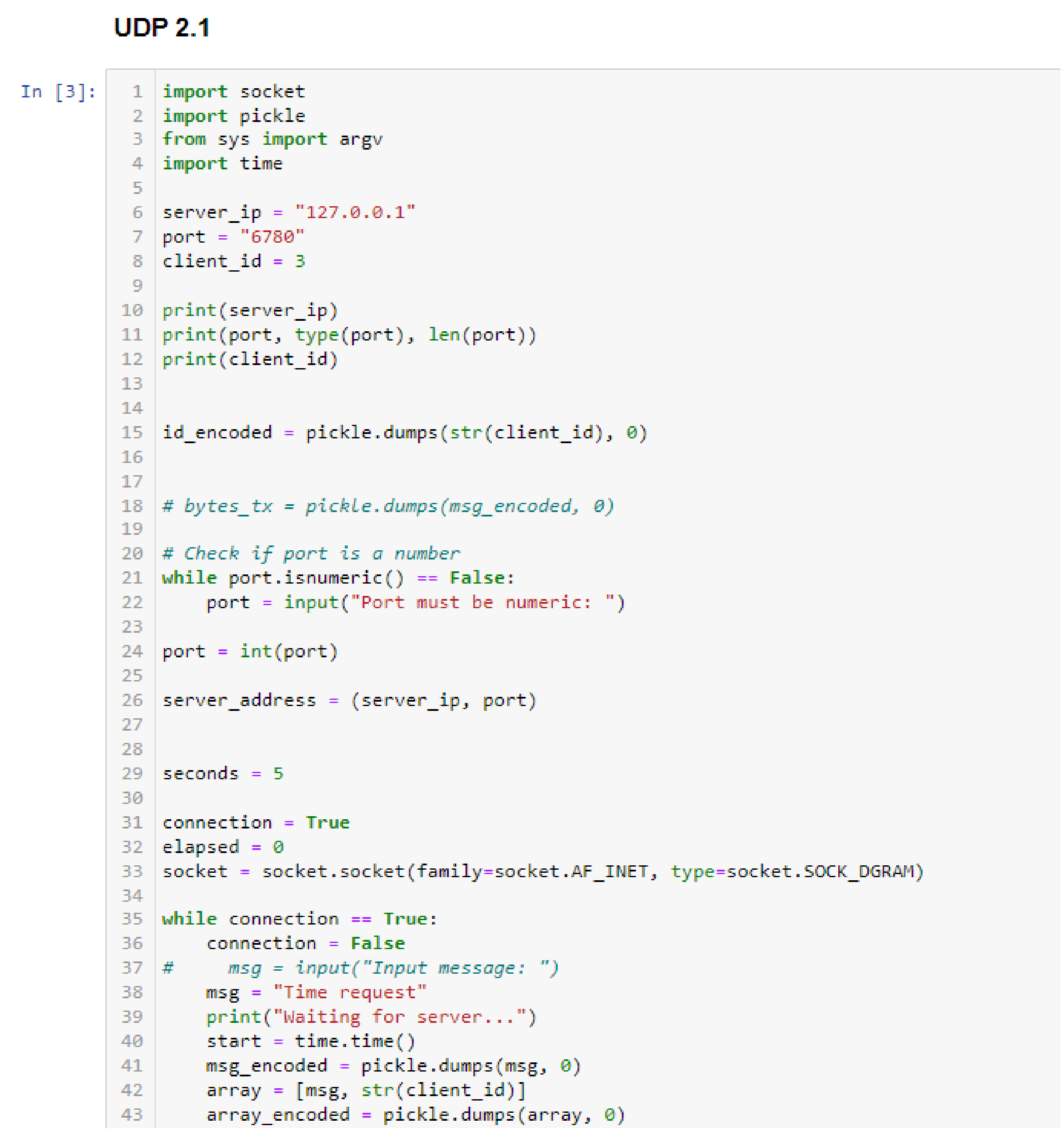
In [*]:

```python
import socket
import pickle
from sys import argv
import time
from datetime import date
import datetime


server_ip = "127.0.0.1"
port = "6780"

# Check if port is a number
while port.isnumeric() == False:
    port = input("Port must be numeric: ")

port = int(port)

# If port is not a valid number: print("Invalid port")
try:
    socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
    socket.bind((server_ip, port))
    print("Server up and listening!")

    while True:
        bytes_rx = socket.recvfrom(1024)
        message = pickle.loads(bytes_rx[0])[0]
        client_id = pickle.loads(bytes_rx[0])[1]
        address = bytes_rx[1]
        print("\nConnection received.")
        print("Message received: ", str(message))
        print("Number of characters of message", len(message))
        print("Client Id: ", str(client_id))
        print("Client address: ", str(address), "\n")
        if str(message) == "exit":
            print("Received exit instruction.")
            print("Server shutting down")
            bytes_tx = pickle.dumps("Received exit instruction, shutting down server", 0)
            socket.sendto(bytes_tx, address)
            break
        else:
            day = date.today()
            now = datetime.datetime.now()
            bytes_tx = pickle.dumps([day,now], 0)
            socket.sendto(bytes_tx, address)
except Exception as e:
#     print("Invalid Port")
    print(e)
```

```
Server up and listening!
```

Si recibe cualquier instrucción que no sea ‘exit’, el servidor su hora exacta en el momento de la recepción.

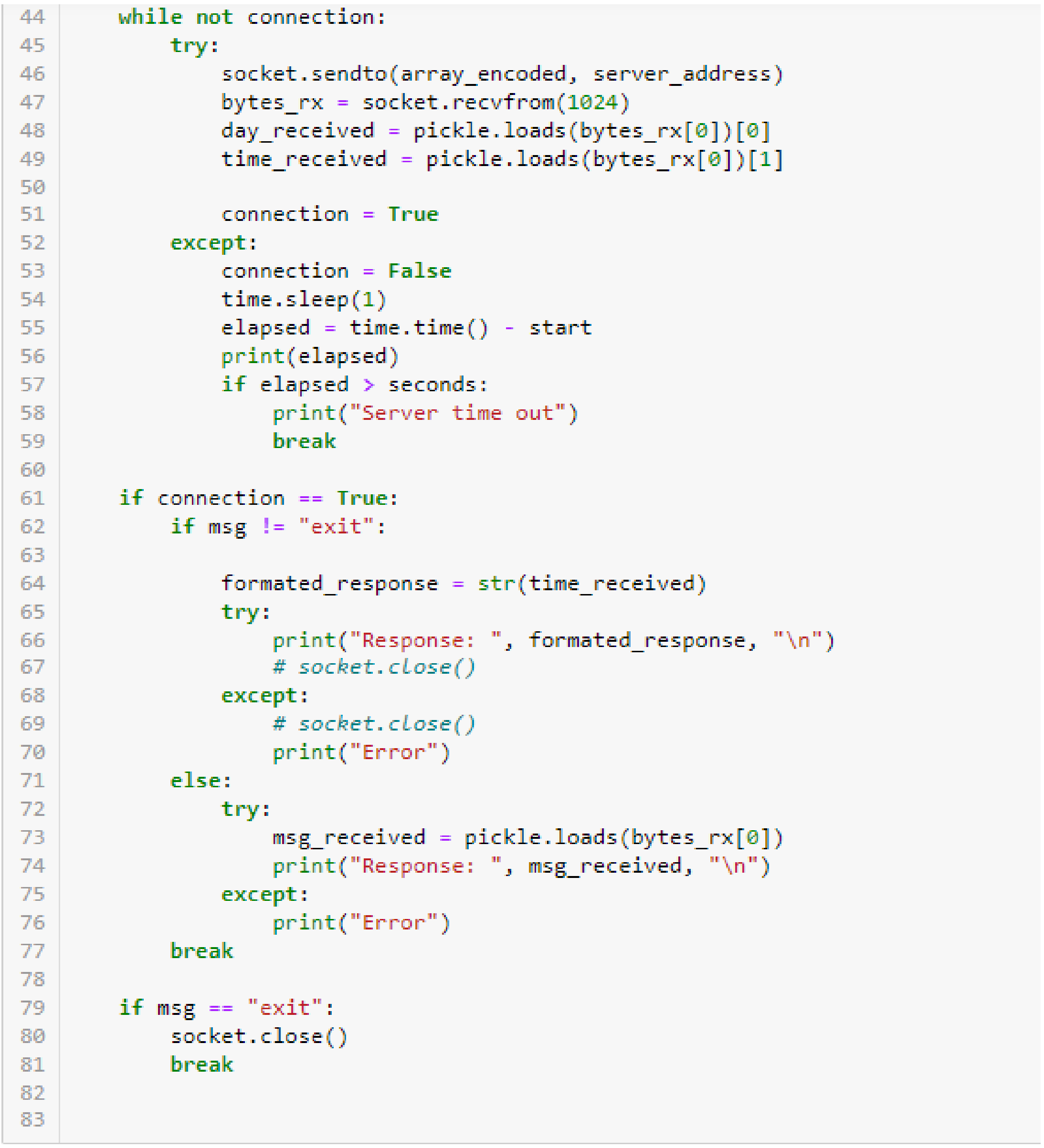
Código del cliente:



UDP 2.1

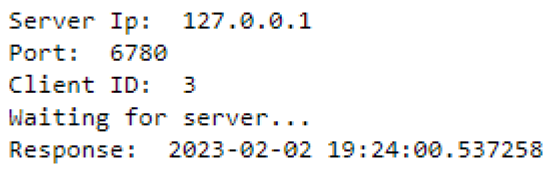
In [3]:

```python
import socket
import pickle
from sys import argv
import time

server_ip = "127.0.0.1"
port = "6780"
client_id = 3

print(server_ip)
print(port, type(port), len(port))
print(client_id)


id_encoded = pickle.dumps(str(client_id), 0)


# bytes_tx = pickle.dumps(msg_encoded, 0)

# Check if port is a number
while port.isnumeric() == False:
    port = input("Port must be numeric: ")

port = int(port)

server_address = (server_ip, port)


seconds = 5

connection = True
elapsed = 0
socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

while connection == True:
    connection = False
#     msg = input("Input message: ")
    msg = "Time request"
    print("Waiting for server...")
    start = time.time()
    msg_encoded = pickle.dumps(msg, 0)
    array = [msg, str(client_id)]
    array_encoded = pickle.dumps(array, 0)
```



```python
    while not connection:
        try:
            socket.sendto(array_encoded, server_address)
            bytes_rx = socket.recvfrom(1024)
            day_received = pickle.loads(bytes_rx[0])[0]
            time_received = pickle.loads(bytes_rx[0])[1]

            connection = True
        except:
            connection = False
            time.sleep(1)
            elapsed = time.time() - start
            print(elapsed)
            if elapsed > seconds:
                print("Server time out")
                break

    if connection == True:
        if msg != "exit":

            formated_response = str(time_received)
            try:
                print("Response: ", formated_response, "\n")
                # socket.close()
            except:
                # socket.close()
                print("Error")
        else:
            try:
                msg_received = pickle.loads(bytes_rx[0])
                print("Response: ", msg_received, "\n")
            except:
                print("Error")
        break

    if msg == "exit":
        socket.close()
        break


```

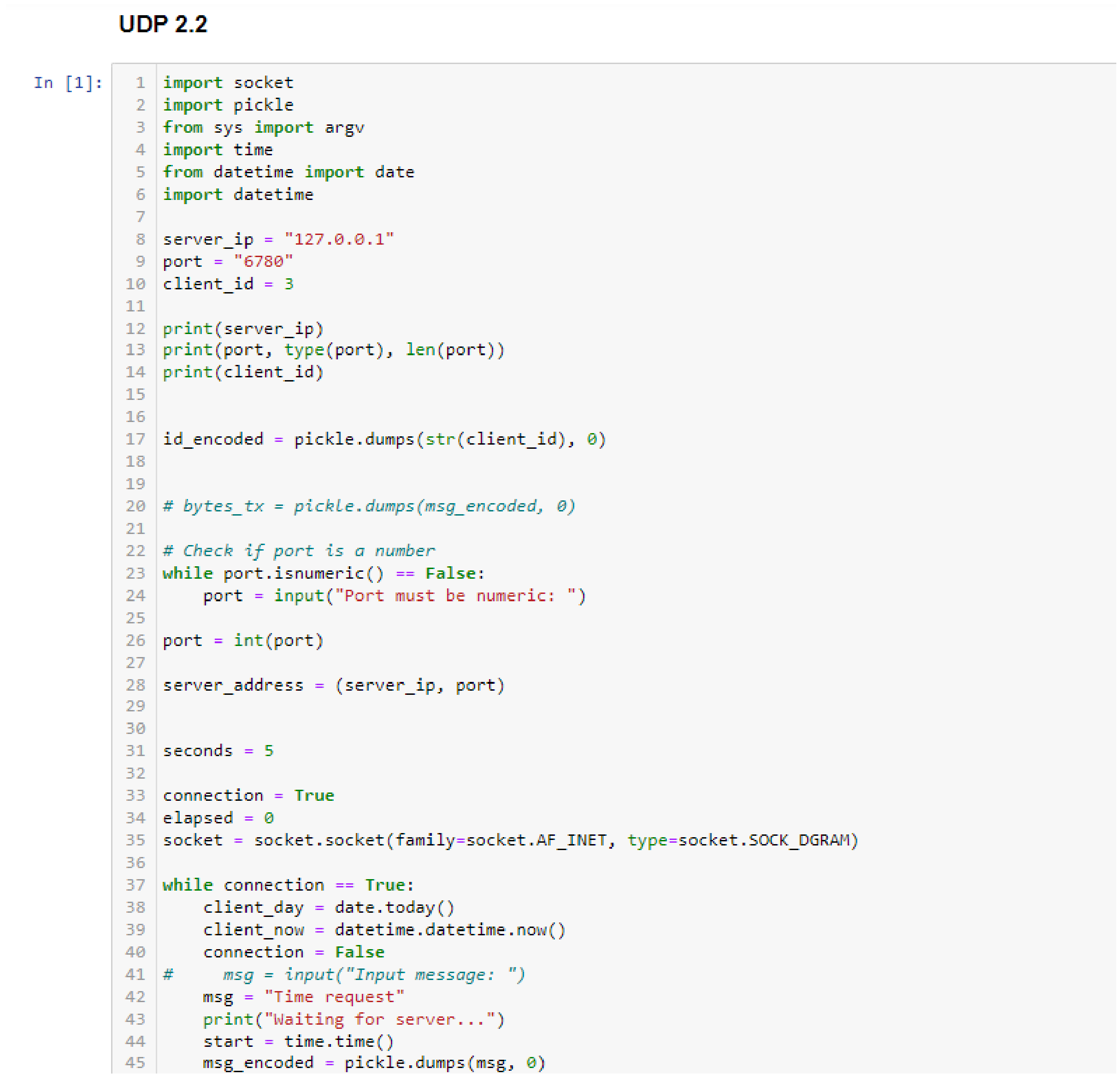
El cliente manda una petición de hora al servidor y espera 5 segundos para recibirla.



```
Server Ip:  127.0.0.1
Port:  6780
Client ID:  3
Waiting for server...
Response:  2023-02-02 19:24:00.537258
```

UDP-2.2

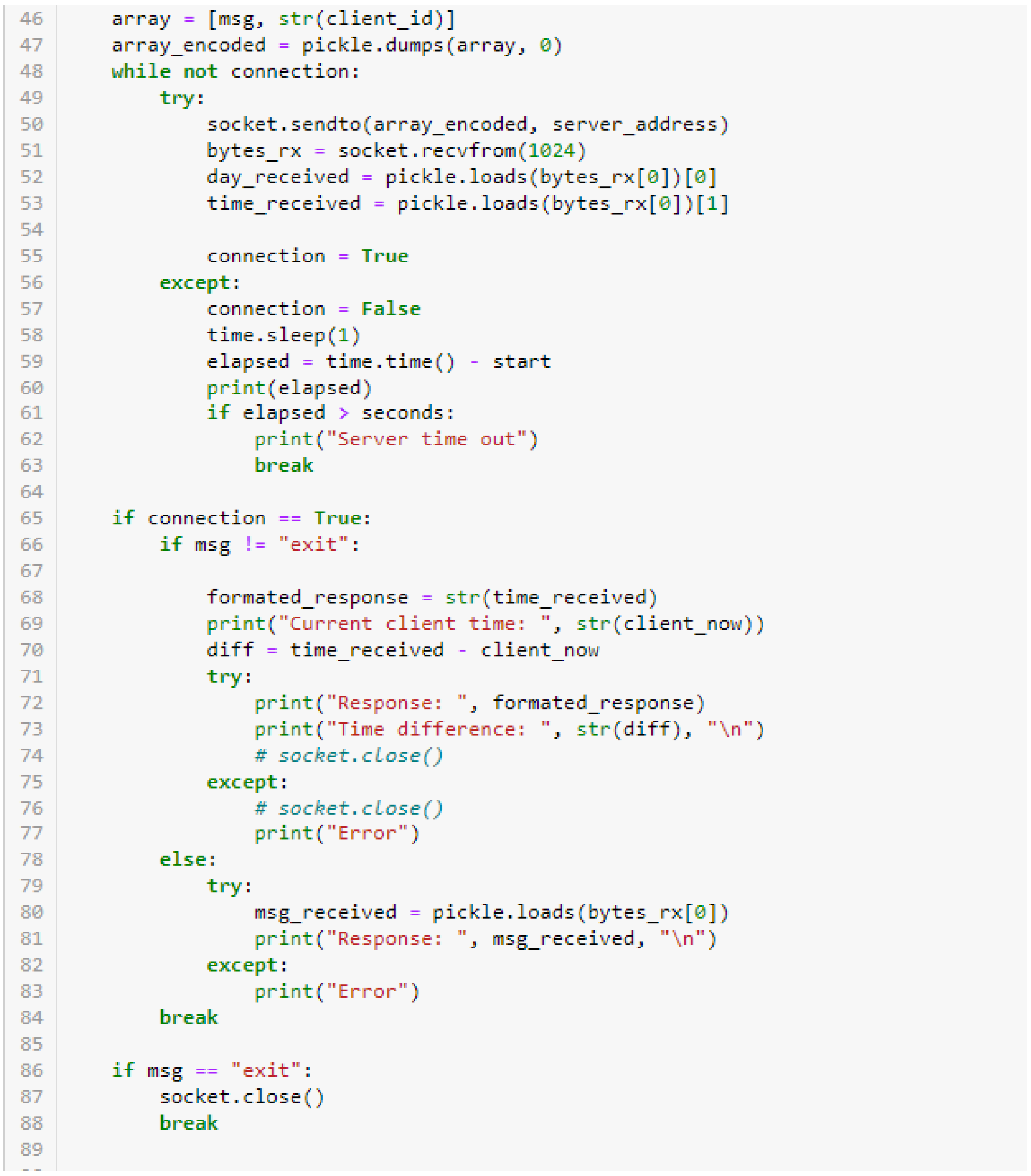
Código del cliente:



UDP 2.2

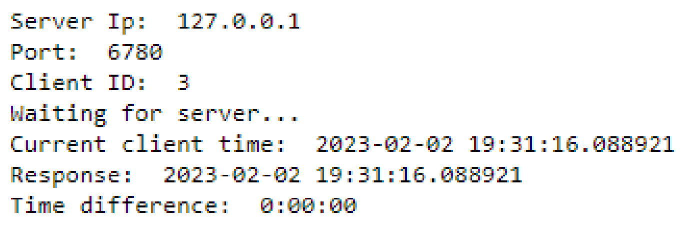
In [1]:

```python
import socket
import pickle
from sys import argv
import time
from datetime import date
import datetime

server_ip = "127.0.0.1"
port = "6780"
client_id = 3

print(server_ip)
print(port, type(port), len(port))
print(client_id)


id_encoded = pickle.dumps(str(client_id), 0)


# bytes_tx = pickle.dumps(msg_encoded, 0)

# Check if port is a number
while port.isnumeric() == False:
    port = input("Port must be numeric: ")

port = int(port)

server_address = (server_ip, port)


seconds = 5

connection = True
elapsed = 0
socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

while connection == True:
    client_day = date.today()
    client_now = datetime.datetime.now()
    connection = False
#     msg = input("Input message: ")
    msg = "Time request"
    print("Waiting for server...")
    start = time.time()
    msg_encoded = pickle.dumps(msg, 0)
```



```python
    array = [msg, str(client_id)]
    array_encoded = pickle.dumps(array, 0)
    while not connection:
        try:
            socket.sendto(array_encoded, server_address)
            bytes_rx = socket.recvfrom(1024)
            day_received = pickle.loads(bytes_rx[0])[0]
            time_received = pickle.loads(bytes_rx[0])[1]

            connection = True
        except:
            connection = False
            time.sleep(1)
            elapsed = time.time() - start
            print(elapsed)
            if elapsed > seconds:
                print("Server time out")
                break

    if connection == True:
        if msg != "exit":

            formated_response = str(time_received)
            print("Current client time: ", str(client_now))
            diff = time_received - client_now
            try:
                print("Response: ", formated_response)
                print("Time difference: ", str(diff), "\n")
                # socket.close()
            except:
                # socket.close()
                print("Error")
        else:
            try:
                msg_received = pickle.loads(bytes_rx[0])
                print("Response: ", msg_received, "\n")
            except:
                print("Error")
        break

    if msg == "exit":
        socket.close()
        break

```

El cliente guarda su hora local antes de hacer la llamada y recibe la hora del servidor. Con esos dos tiempos, calcula la diferencia entre ellos para ver cuánto ha tardado la conexión.

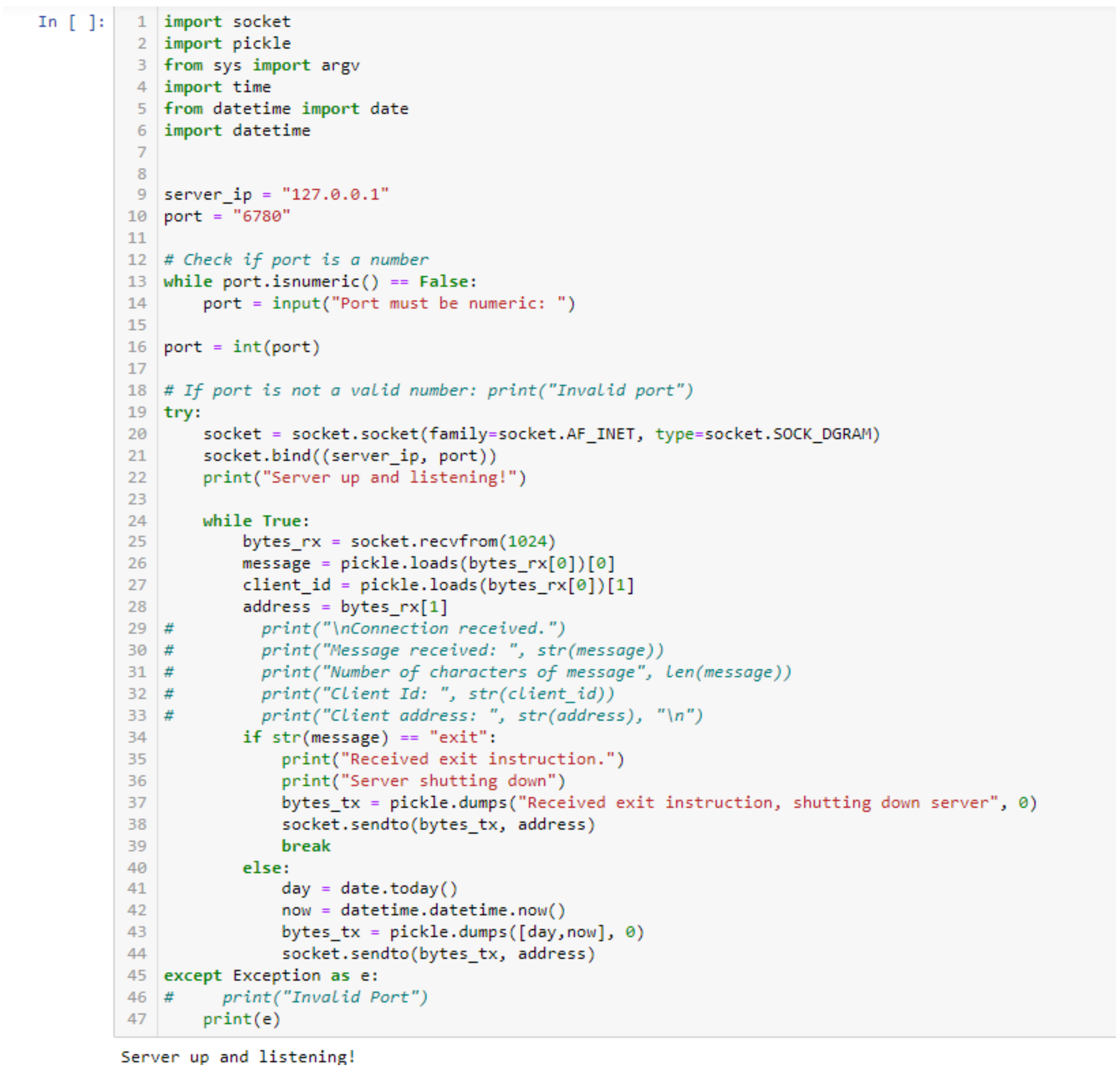


```
Server Ip:  127.0.0.1
Port:  6780
Client ID:  3
Waiting for server...
Current client time:  2023-02-02 19:31:16.088921
Response:  2023-02-02 19:31:16.088921
Time difference:  0:00:00
```

Como se están ejecutando ambos en la misma máquina, el tiempo es demasiado pequeño como para medirlo.

UDP-2.3

Código del servidor:

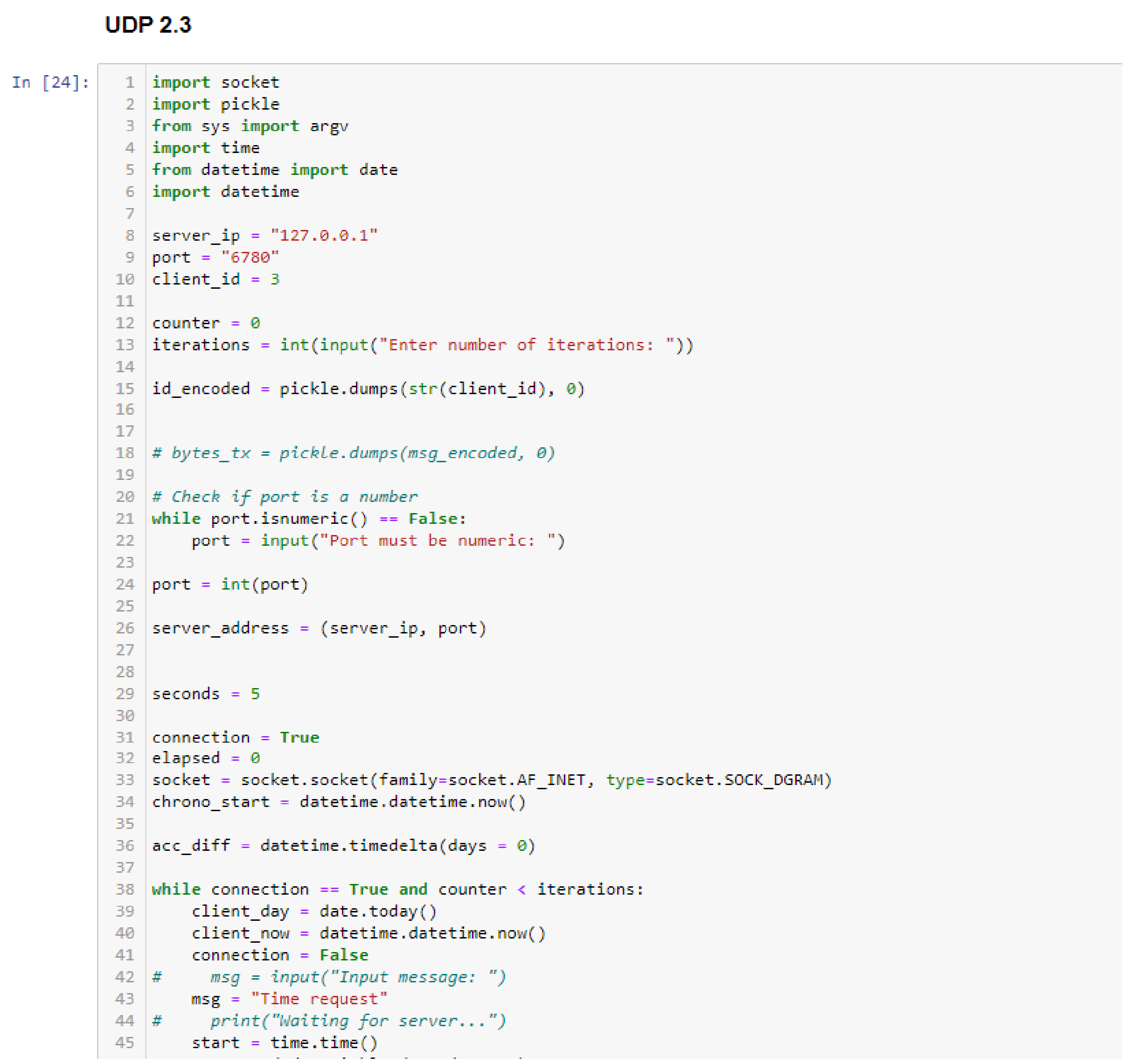


```
In [ ]:
import socket
import pickle
from sys import argv
import time
from datetime import date
import datetime


server_ip = "127.0.0.1"
port = "6780"

# Check if port is a number
while port.isnumeric() == False:
    port = input("Port must be numeric: ")

port = int(port)

# If port is not a valid number: print("Invalid port")
try:
    socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
    socket.bind((server_ip, port))
    print("Server up and listening!")

    while True:
        bytes_rx = socket.recvfrom(1024)
        message = pickle.loads(bytes_rx[0])[0]
        client_id = pickle.loads(bytes_rx[0])[1]
        address = bytes_rx[1]
#         print("\nConnection received.")
#         print("Message received: ", str(message))
#         print("Number of characters of message", len(message))
#         print("Client Id: ", str(client_id))
#         print("Client address: ", str(address), "\n")
        if str(message) == "exit":
            print("Received exit instruction.")
            print("Server shutting down")
            bytes_tx = pickle.dumps("Received exit instruction, shutting down server", 0)
            socket.sendto(bytes_tx, address)
            break
        else:
            day = date.today()
            now = datetime.datetime.now()
            bytes_tx = pickle.dumps([day,now], 0)
            socket.sendto(bytes_tx, address)
except Exception as e:
#     print("Invalid Port")
    print(e)
```

```
Server up and listening!
```

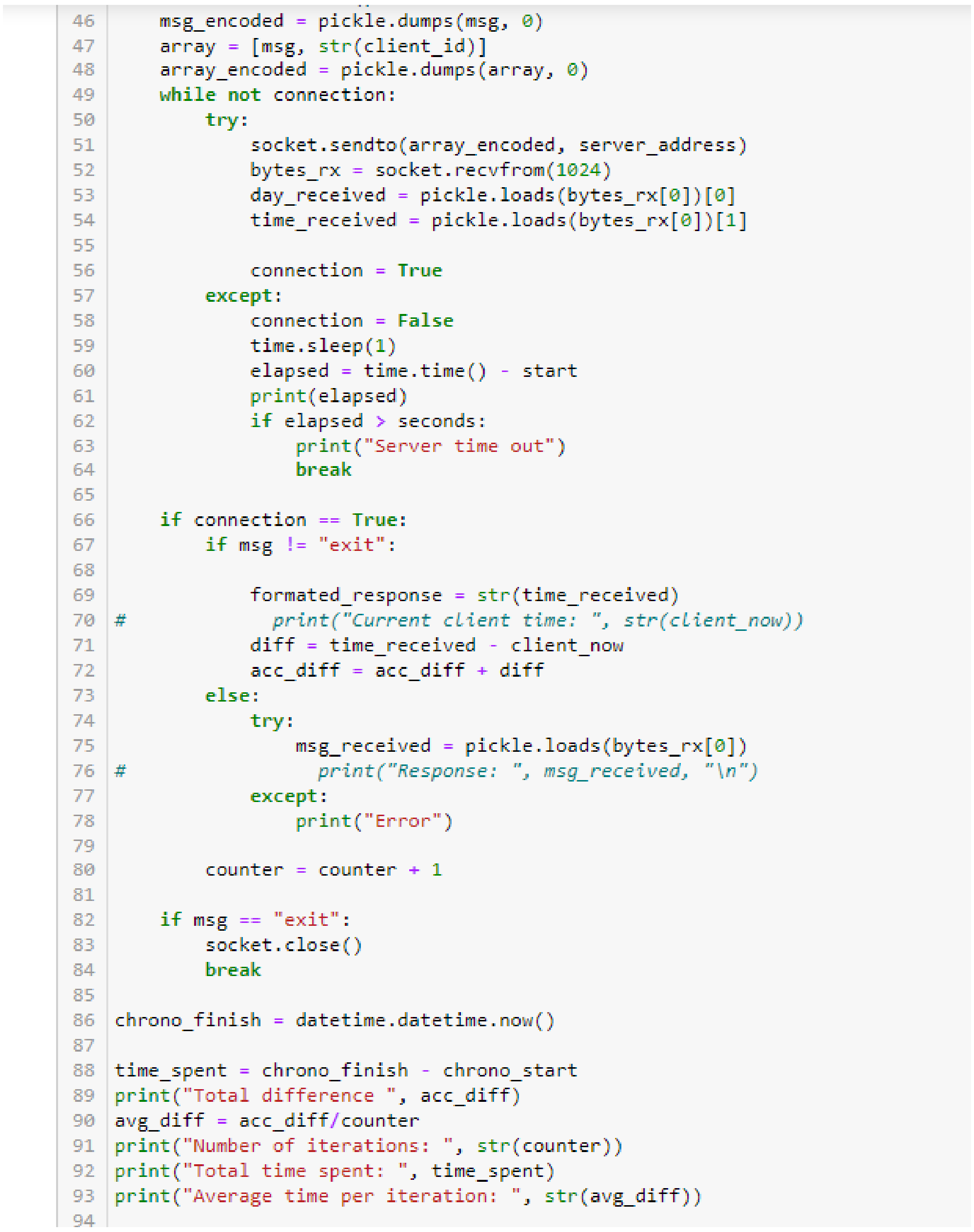
Se han comentado todos los prints para que no se pierdan recursos en mostrar las 100.000 recepciones.

Código del cliente:



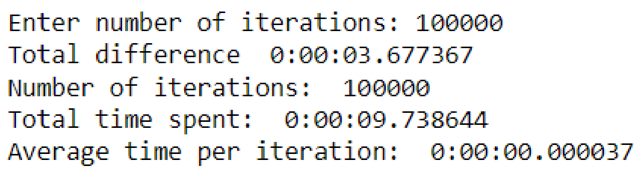
UDP 2.3

```
In [24]:
import socket
import pickle
from sys import argv
import time
from datetime import date
import datetime

server_ip = "127.0.0.1"
port = "6780"
client_id = 3

counter = 0
iterations = int(input("Enter number of iterations: "))

id_encoded = pickle.dumps(str(client_id), 0)


# bytes_tx = pickle.dumps(msg_encoded, 0)

# Check if port is a number
while port.isnumeric() == False:
    port = input("Port must be numeric: ")

port = int(port)

server_address = (server_ip, port)


seconds = 5

connection = True
elapsed = 0
socket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
chrono_start = datetime.datetime.now()

acc_diff = datetime.timedelta(days = 0)

while connection == True and counter < iterations:
    client_day = date.today()
    client_now = datetime.datetime.now()
    connection = False
#     msg = input("Input message: ")
    msg = "Time request"
#     print("Waiting for server...")
    start = time.time()
```



```
    msg_encoded = pickle.dumps(msg, 0)
    array = [msg, str(client_id)]
    array_encoded = pickle.dumps(array, 0)
    while not connection:
        try:
            socket.sendto(array_encoded, server_address)
            bytes_rx = socket.recvfrom(1024)
            day_received = pickle.loads(bytes_rx[0])[0]
            time_received = pickle.loads(bytes_rx[0])[1]

            connection = True
        except:
            connection = False
            time.sleep(1)
            elapsed = time.time() - start
            print(elapsed)
            if elapsed > seconds:
                print("Server time out")
                break

    if connection == True:
        if msg != "exit":

            formated_response = str(time_received)
#             print("Current client time: ", str(client_now))
            diff = time_received - client_now
            acc_diff = acc_diff + diff
        else:
            try:
                msg_received = pickle.loads(bytes_rx[0])
#                 print("Response: ", msg_received, "\n")
            except:
                print("Error")

        counter = counter + 1

    if msg == "exit":
        socket.close()
        break

chrono_finish = datetime.datetime.now()

time_spent = chrono_finish - chrono_start
print("Total difference ", acc_diff)
avg_diff = acc_diff/counter
print("Number of iterations: ", str(counter))
print("Total time spent: ", time_spent)
print("Average time per iteration: ", str(avg_diff))
```

El cliente recibe como entrada un número de iteraciones. Entonces, hará tantas llamadas al servidor como iteraciones. En cada llamada, calcula el tiempo que tarda el mensaje en enviarse al servidor y en devolverse al cliente. Al final, calcula el tiempo medio de todas las iteraciones.



```
Enter number of iterations: 100000
Total difference  0:00:03.677367
Number of iterations:  100000
Total time spent:  0:00:09.738644
Average time per iteration:  0:00:00.000037
```