

# Sockets

## con

## Python

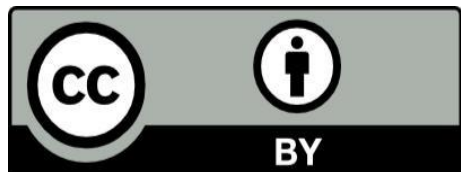
Hernán Castilla  
Jose Altamar

# Sockets con Python

por Hernán Castilla  
y Jose Altamar

Esta guía se distribuye bajo una licencia Creative Commons  
Atribución Colombia.

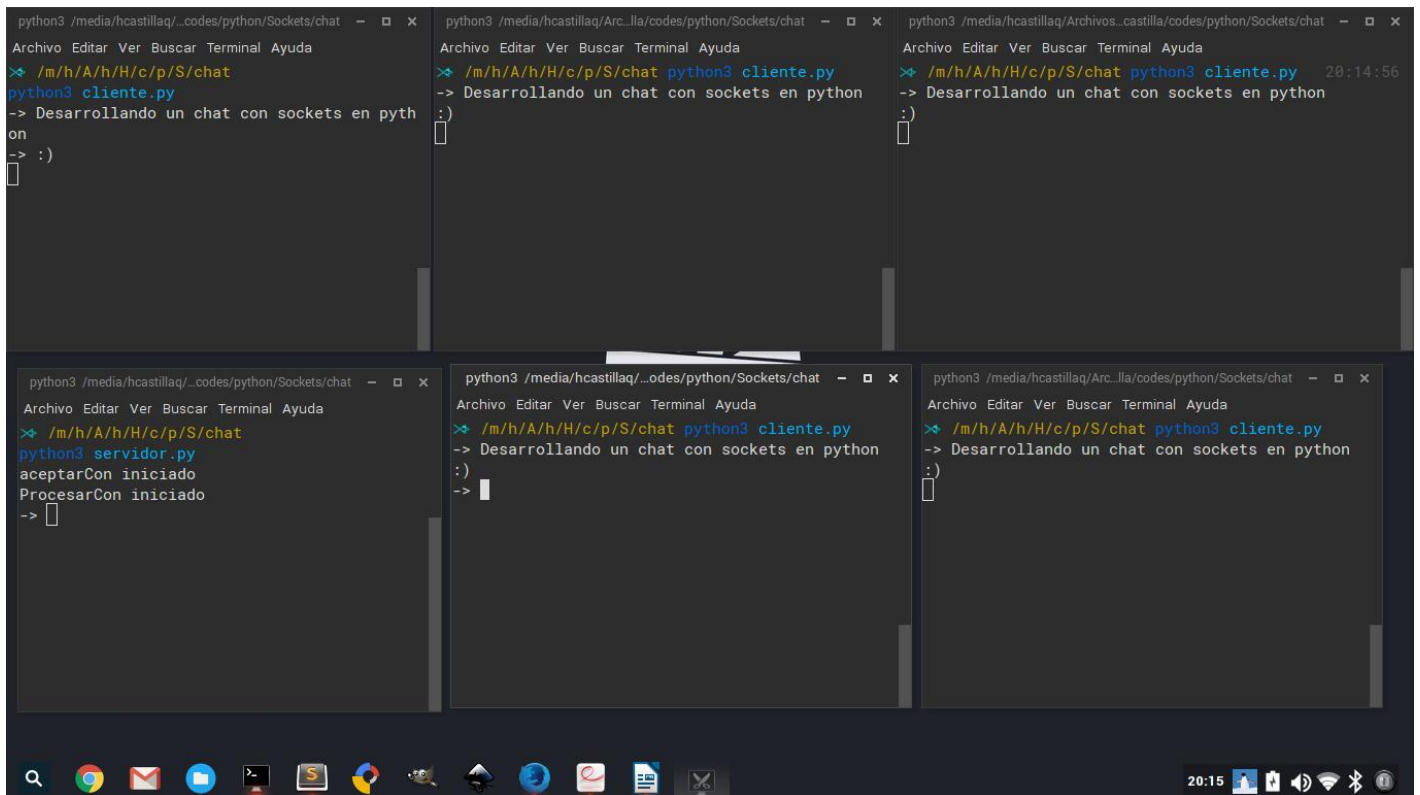
Esta licencia le permite: distribuir, mezclar, ajustar y construir a  
partir de esta obra, incluso con fines comerciales, siempre que le  
sea reconocida la autoría de la creación original.



# Nota:

La idea principal de esta guía son los Sockets no bloqueantes puesto que al día de hoy no he encontrado mucha información sobre este tema y tampoco le queremos echar mas leña al fuego.

La principal expectativa de esta guía es poder desarrollar un pequeño chat utilizando Sockets no bloqueantes.



The image shows a screenshot of a desktop environment with six terminal windows arranged in a 2x3 grid. Each terminal window has a title bar indicating the current directory and the active Python interpreter (python3). The windows show the following commands and outputs:

- Top-left window:** Shows the command `python3 cliente.py` and the output `-> Desarrollando un chat con sockets en python`.
- Top-middle window:** Shows the command `python3 cliente.py` and the output `-> Desarrollando un chat con sockets en python`.
- Top-right window:** Shows the command `python3 cliente.py` and the output `-> Desarrollando un chat con sockets en python`.
- Bottom-left window:** Shows the command `python3 servidor.py` and the output `aceptarCon iniciado` and `ProcesarCon iniciado`.
- Bottom-middle window:** Shows the command `python3 cliente.py` and the output `-> Desarrollando un chat con sockets en python`.
- Bottom-right window:** Shows the command `python3 cliente.py` and the output `-> Desarrollando un chat con sockets en python`.

The desktop environment includes a taskbar at the bottom with various application icons and a system tray on the right showing the time as 20:15.

## Requisitos

- Python3
- Algo de tiempo

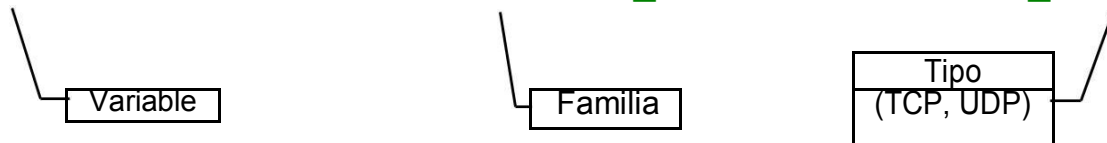
# Manos a la obra

Hagamos una breve introducción a los Sockets en python, lo primero es importarlos:

```
import socket
```

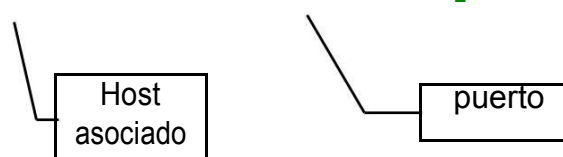
se pueden definir de la siguiente manera

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```



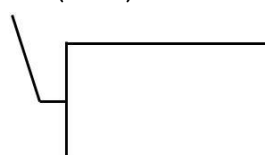
se asocia a un host con el método bind, el cual recibe por parámetro un tupla

```
sock.bind( (str( host ), int( port ) ) )
```



y se encolan un par de conexiones antes de rechazar alguna

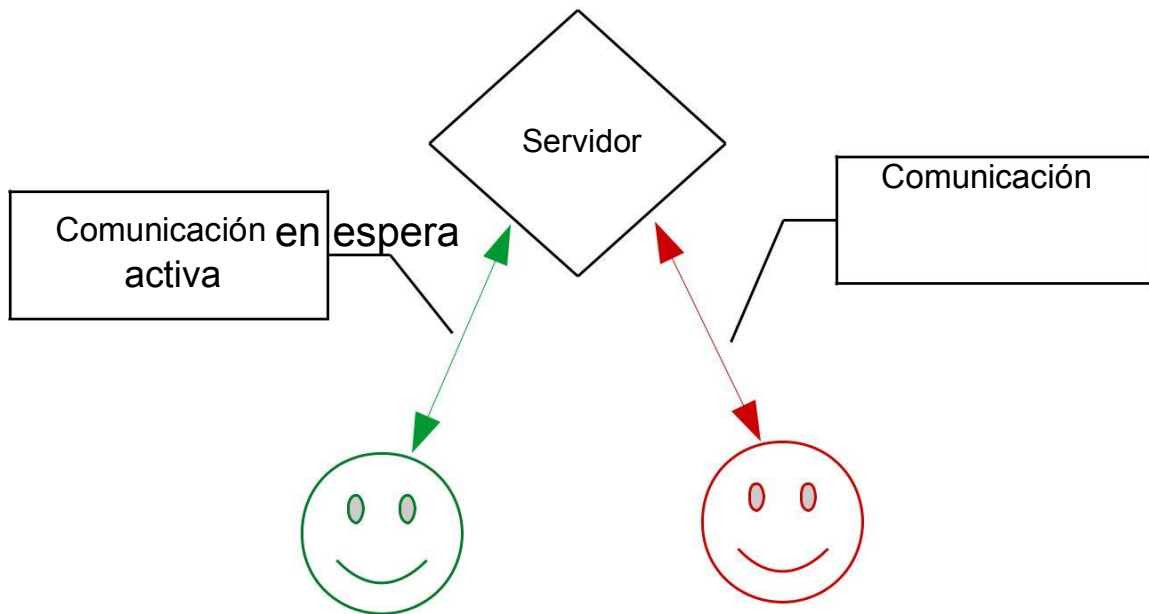
```
sock.listen(10)
```



Numero de  
conexiones  
en la cola

Con esto tenemos un socket de tipo bloqueante, pero ¿cual es el problema?

Tratemos de explicarlo mediante un ejemplo:

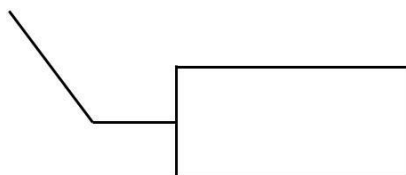


el problema es que el servidor se queda digamos que enchufado a una sola conexión y pone en espera las demás, lo cual para la creación de un chat no es nada conveniente.

¿ Cual es la solución ?

Seria llamar al método `setblocking` el cual recibe un parámetro de tipo booleano

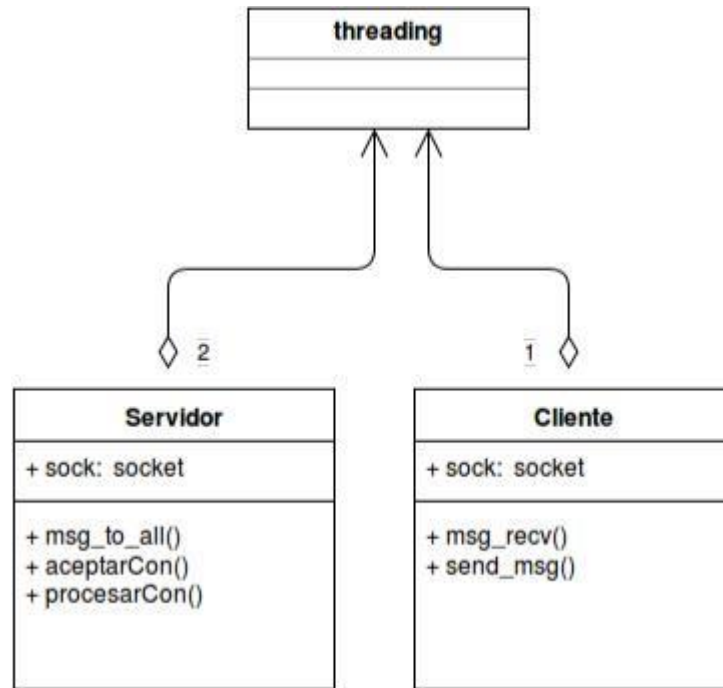
```
self.sock.setblocking(False)
```





Metodo que  
desbloquea  
el socket

# Vamos con el chat



importamos las librerías

```
import socket
import threading
import sys
import pickle
```

la librería pickle nos permite serializar los mensajes para poder mandarlos en forma de bytes.

# Cliente

```
class Cliente():  
  
    def __init__(self, host="localhost", port=7000):  
  
        self.sock = socket.socket(socket.AF_INET,  
                                   socket.SOCK_STREAM)  
  
        self.sock.connect((str(host), int(port)))
```

ya tenemos listo el socket, ahora para leer todos los mensajes que envía el servidor utilizamos un hilo...

Si tienes dudas sobre los hilos al final dejare algunos links.

```
def __init__(self, host="localhost", port=7000):  
  
    .....  
  
    msg_rcv = threading.  
        Thread(target=self.msg_rcv)  
  
    msg_rcv.daemon =  
    True msg_rcv.start()
```

ahora creamos un ciclo que mantendrá vivo el hilo principal y nos permitirá escribir los mensajes.

```
def __init__(self, host="localhost", port=7000):  
  
    .....  
  
    while True:  
        msg = input('-> ')  
        if msg != 'salir':  
            self.send_msg(msg)  
        else:  
            self.sock.close()  
            sys.exit()
```

definimos la función msg\_recv:

```
def msg_recv(self):  
    while True:  
        try:  
            data = self.sock.recv(1028)  
            if data:  
                data = pickle.loads(data)  
                print(data)  
        except:  
            pass
```

esto no es mas que un while sin fin, el cual siempre estará pendiente a los mensajes que envíe el servidor para mostralos por pantalla.

Ahora definimos la función para enviar los mensajes `send_msg`

```
def send_msg(self,  
    msg): try:  
        self.sock.send(pickle.dumps(msg))  
    except:  
        print('error')
```

# Servidor

creamos la clase servidor y definimos su constructor pasándole por parámetro el host y el puerto para la creación de socket

```
class Servidor():

    def __init__(self, host="localhost", port=7000):

        #arreglo para guardar los
        #clientes conectados
        self.clientes = []

        self.sock = socket.socket(socket.AF_INET,
                                   socket.SOCK_STREAM)

        self.sock.bind((str(host), int(port)))
        self.sock.listen(10)
        self.sock.setblocking(False)

        #hilos para aceptar y procesar
        #las conexiones
        aceptar = threading
            .Thread(target=self.aceptarCon)
        procesar = threading
            .Thread(target=self.procesarCon)

        aceptar.daemon =
        True aceptar.start()
```

```
procesar.daemon =  
True procesar.start()
```

creamos el while que mantendrá vivo el hilo principal

```
def __init__(self, host="localhost", port=7000):  
    ...  
  
    try:  
        while True:  
            msg = input('-> ')  
            if msg == 'salir':  
                break  
            self.sock.close()  
            sys.exit()  
    except:  
        self.sock.close()  
        sys.exit()
```

definimos la función que nos permitirá enviar los mensajes a todos los clientes conectados

```
def msg_to_all(self, msg, cliente):  
    for c in self.clientes:  
        try:  
            if c != cliente:  
                c.send(msg)  
        except:  
            self.clientes.remove(c)
```



definimos la funcion que aceptara las conexiones y las almacenara en el arreglo de clientes

```
def aceptarCon(self):  
    print("aceptarCon iniciado")  
    while True:  
        try:  
            conn, addr = self.sock.accept()  
            conn.setblocking(False)  
            self.clientes.append(conn)  
        except:  
            pass
```

por ultimo definimos la funcion para procesar las conexiones, esta contendrá un while infinito que estará recorriendo la lista de clientes para saber cuando recibe un mensaje.

```
def procesarCon(self):  
    print("ProcesarCon  
iniciado")  
    while True:  
        if len(self.clientes) > 0:  
            for c in self.clientes:  
                try:  
                    data = c.recv(1024)  
                    if data:  
                        self.msg_to_all(data,c)  
                except:  
                    pass
```

threading:

[http://www.bogotobogo.com/python/Multithread/python\\_multithreading\\_creating\\_threads.php](http://www.bogotobogo.com/python/Multithread/python_multithreading_creating_threads.php)

sockets:

HYPERLINK

"https://docs.python.org/3/library/socket.html" <https://docs.python.org/3/library/socket.ht>