What is Socket Programming?

Sockets (socket programming) is a program that enables two sockets to send and receive data, **bi-directionally**, at any given moment.

It works by connecting two sockets (or nodes) together and allowing them to communicate in real time

Socket Basics

A network socket is an endpoint of an inter-process communication flow across a computer network. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents. Today, most communication between computers is based on the internet protocol; therefore most network sockets are internet sockets. To create a connection between machines, Python programs import the socket module, create a socket object, and call the object’s methods to establish connections and send and receive data. Sockets are the endpoints of a bidirectional communications channel.

Socket in Python

Python provides two levels of access to network services. At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols. Python also has libraries that provide higher level access to specific application level network protocols, such as FTP, HTTP, SMTP, and so on. Sockets may be implemented over a number of different channel types: UNIX domain sockets, TCP, UDP, and so on. The socket library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

Vocabulary of Sockets

Term	Description
domain	The family of protocols that will be used as the transport mechanism. These values are constants such as AF_INET, PF_INET, PF_UNIX, PF_X25, and so on.
type	The type of communications between the two endpoints, typically SOCK_STREAMfor connection-oriented protocols and SOCK_DGRAMfor connectionless protocols.
protocol	Typically zero, this may be used to identify a variant of a protocol within a domain and type.
hostname	The identifier of a network interface: • A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation • A string "<broadcast>", which specifies an INADDR_BROADCAST address. • A zero-length string, which specifies INADDR_ANY, or • An Integer, interpreted as a binary address in host byte order.

port	Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service.

The socket Module

To create a socket, you must use the socket.socket() function available in socket module, which has the general syntax:
s = socket.socket (socket_family, socket_type, protocol=0)

Description of the parameters:

- socket_family: This is either AF_UNIX or AF_INET, as explained earlier.
- socket_type: This is either SOCK_STREAM (TCP/IP Protocol) or SOCK_DGRAM (UDP).
- protocol: This is usually left out, defaulting to 0.

Once you have socket object, then you can use required functions to create your client or server program.

Server Socket Methods

Method	Description
s.bind()	This method binds address (hostname, port number pair) to socket.
s.listen()	This method sets up and start TCP listener.
s.accept()	This passively accept TCP client connection, waiting until connection arrives (blocking).

Client Socket Methods

Method	Description
s.connect()	This method actively initiates TCP server connection.

General Socket Methods

Method	Description
s.recv()	This method receives TCP message
s.send()	This method transmits TCP message
s.recvfrom()	This method receives UDP message
s.sendto()	This method transmits UDP message
s.close()	This method closes socket
socket.gethostname()	Returns the hostname.

A Simple Server

To write Internet servers, we use the socket function available in socket module to create a socket object. A socket object is then used to call other functions to setup a socket server. Now call bind(hostname, port) function to specify a port for your service on the given host. Next, call the accept method of the returned object. This method waits until a client connects to the port you specified, and then returns a connection object that represents the connection to that client.

Server.py

```
#a TCP/IP server that sends a messages to client
import socket

#take the server name
host='localhost'

#take the port number
port=5000

#create a socket at server side using TCP/IP protocol
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#bind the socket with server server and port numner
s.bind((host,port))

#allow maximum 1 connection to the socket
s.listen(1)

#wait till a client accepts connection
c, addr = s.accept()

#display client address
print("Connection from: ",str(addr))
```

```
#send messages to the client after encoding into binary string

c.send(b"Hello client, how are U")

msg="Bye!"

c.send(msg.encode())



#disconnecting the server

c.close()
```

A Simple Client

Now we will write a very simple client program which will open a connection to a given port 12345 and given host. This is very simple to create a socket client using Python's socket module function. The socket.connect(hosname, port) opens a TCP connection to hostname on the port. Once you have a socket open, you can read from it like any IO object. When done, remember to close it, as you would close a file. The following code is a very simple client that connects to a given host and port, reads any available data from the socket, and then exits:

Client.py

```
#a TCP/IP client that receives a message from server
import socket


#take the server name
host='localhost'
#take the port number
port=5000


#create a socket at server side using TCP/IP protocol
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#connect it to server and port number
s.connect((host,port))

#receive message string from server, at a time 1024 B
msg = s.recv(1024)

#receive as long as message strings are not empty
while msg:
   print('Received: '+ msg.decode())
   msg = s.recv(1024)
```

Output :

```
Received: Hello client, how are U

Received: Bye!
```

Try to execute these programs on two different terminals. First execute server.py in one terminal and then execute client.py in another terminal.

Python Internet Modules

A list of some important modules which could be used in Python Network/Internet programming.

Protocol	**Common function**	**Port No**	**Python module**
HTTP	Web pages	80	httplib, urllib, xmlrpclib
NNTP	Usenet news	119	nntplib
FTP	File transfers	20	ftplib, urllib
SMTP	Sending email	25	smtplib
POP3	Fetching email	110	poplib
IMAP4	Fetching email	143	imaplib
Telnet	Command lines	23	telnetlib
Gopher	Document transfers	70	gopherlib, urllib