**What is Socket Programming in Python and how to master it?**

The Internet has undeniably become the ‘Soul of Existence’ and its activity is characterized by ‘Connections’ or ‘Networks’. These networks are made possible using one of the most crucial fundamentals of **Sockets.**This article covers all areas dealing with Socket Programming in Python. Sockets help you make these connections, while Python, undoubtedly, makes it easy.

Let’s take a quick look at all the topics covered in this article:

**Why use Sockets?**

Sockets are the backbone of networking. They make the transfer of information possible between two different programs or devices. For example, when you open up your browser, you as a client are creating a connection to the server for the transfer of information.

Before diving deeper into this communication, let’s first figure out what exactly are these sockets.

**What are Sockets?**

In general terms, sockets are interior endpoints built for sending and receiving data. A single network will have two sockets, one for each communicating device or program. These sockets are a combination of an IP address and a Port. A single device can have ‘n’ number of sockets based on the port number that is being used. Different ports are available for different types of protocols.

**More info about sockets on:**

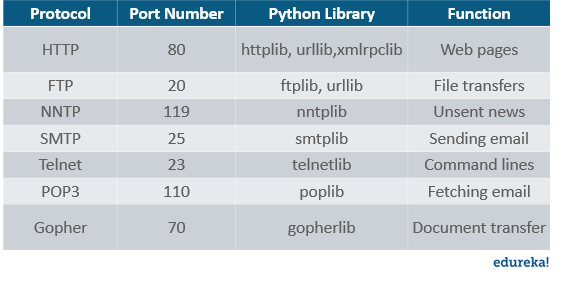
The socket is primarily a **concept used in the transport layer of the Internet protocol suite** or session layer of the OSI model

https://en.wikipedia.org/wiki/Network\_socket

https://en.wikipedia.org/wiki/OSI\_model

https://en.wikipedia.org/wiki/Transport\_layer

Take a look at the following image for more about some of the common port numbers and the related protocols:



Now that you are clear about the concept of sockets, let’s now take a look at the Socket module of Python:

**How to achieve Socket Programming in Python:**

To achieve Socket Programming in Python, you will need to import the **socket** module. This module consists of built-in methods that are required for creating sockets and help them associate with each other.

Some of the important methods are as follows:

|  |  |
| --- | --- |
| Methods | Description |
| *socket.socket()* | used to create sockets (required on both server as well as client ends to create sockets) |
| *socket.accept()* | used to accept a connection. It returns a pair of values (conn, address) where conn is a new socket object for sending or receiving data and address is the address of the socket present at the other end of the connection |
| *socket.bind()* | used to bind to the address that is specified as a parameter |
| *socket.close()* | used to mark the socket as closed |
| *socket.connect()* | used to connect to a remote address specified as the parameter |
| *socket.listen()* | enables the server to accept connections |

Now that you have understood the importance of socket module, let’s move on to see how it can serve to create servers and clients for Socket Programming in Python.

**What is a Server?**

A server is either a program, a computer, or a device that is devoted to managing network resources. Servers can either be on the same device or computer or locally connected to other devices and computers or even remote. There are various types of servers such as database servers, network servers, print servers, etc.

Servers commonly make use of methods like socket.socket(), socket.bind(), socket.listen(), etc to establish a connection and bind to the clients. Now let’s write a program to create a server. Consider the following example:

**EXAMPLE:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | import socket  s=socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  s.bind((socket.gethostname(),1234))  #port number can be anything between 0-65535(we usually specify non-previleged ports which are > 1023)  s.listen(5)    while True:      clt,adr=s.accept()      print(f"Connection to {adr}established")     #f string is literal string prefixed with f which     #contains python expressions inside braces      clt.send(bytes("Socket Programming in Python","utf-8 ")) #to send info to clientsocket |

As you can see, the first necessity to create a socket is to import the socket module. After that the socket.socket() method is used to create a server-side socket.

**NOTE:**

AF\_INET refers to Address from the Internet and it requires a pair of (host, port) where the host can either be a URL of some particular website or its address and the port number is an integer. SOCK\_STREAM is used to create TCP Protocols.

[[](https://www.edureka.co/data-science-python-certification-course)](https://www.edureka.co/data-science-python-certification-course" \t "_blank)

**[Data Science with Python Certification Training Course](https://www.edureka.co/data-science-python-certification-course" \t "_blank)**

[Explore Curriculum](https://www.edureka.co/data-science-python-certification-course" \t "_blank)

The bind() method accepts two parameters as a tuple (host, port). However, it’s better to use 4-digit port numbers as the lower ones are usually occupied. The listen() method allows the server to accept connections. Here, 5 is the queue for multiple connections that come up simultaneously. The minimum value that can be specified here is 0 (If you give a lesser value, it’s changed to 0).  In case no parameter is specified, it takes a default suitable one.

The while loop allows accepting connections forever. ‘clt’ and ‘adr’ are the client object and address. The print statement just prints out the address and the port number of the client socket. Finally, clt.send is used to send the data in bytes.

Now that our server is all set, let us move on towards the client.

**What is a Client?**

A client is either a computer or software that receives information or services from the server. In a client-server module, clients requests for services from servers. The best example is a web browser such as Google Chrome, Firefox, etc. These web browsers request web servers for the required web pages and services as directed by the user. Other examples include online games, online chats, etc.

Now let’s take a look at how to code the client-side program.

**EXAMPLE:**

|  |  |
| --- | --- |
| 1  2  3  4  5 | import socket  s=socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  s.connect((socket.gethostname(), 2346))  msg=s.recv(1024)  print(msg.decode("utf-8")) |

The first step is to import the socket module and then create a socket just like you did while creating a server. Then, to create a connection between the client-server you will need to use the connect() method by specifying (host, port).

**NOTE:**gethostname is used when client and server are on on the same computer. (LAN – localip / WAN – publicip)

Here, the client wants to receive some information from the server and for this, you need to use the recv() method and the information is stored in another variable msg. Just keep in mind that the information being passed will be in bytes and in the client in the above program can receive up to 1024 bytes (buffer size) in a single transfer. It can be specified to any amount depending on the amount of information being transferred.

Finally, the message being transferred should be decoded and printed.

Now that you are aware of how to create client-server programs, let’s move on to see how they need to be executed.