# SOCKET PROGRAMMING

## SOCKETS

- It is possible to establish a logical connecting point between a server and a client so that communication can be done through that point. This point is called ‘Sockets’.
- Each socket is given an identification number, which is called ‘port number’.
- Port number takes 2 bytes and can be 0 to 65535.

## Reserved Port Numbers

| Port Number | Application or Service |
|---|---|
| 13 | Date and time services |
| 21 | FTP which transfers files |
| 23 | Telnet, which provides remote login |
| 25 | SMTP, which delivers mail |
| 67 | BOOTP, which provides configuration at boot time |
| 80 | HTTP, which transfer web pages |
| 109 | POP2, which is a mailing service |
| 110 | POP3, which is a mailing service |
| 119 | NNTP, which is for transferring news articles |
| 443 | HTTPS, which transfer web pages securely |

## CREATING SOCKET

- To create socket:

s = socket.socket(address_family,type)

where address_family = socket.AF_INET (#Internet protocol (IPv4)) – this is default

or socket.AF_INET6 (#Internet protocol(IPv6))

& type = socket.SOCK_STREAM (# for TCP)- this is default

or socket.SOCK_DGRAM (# for UDP)

## KNOWING IP ADDRESS

- This function takes the website name and returns its IP Address.

```
                  addr = socket.gethostbyname(‘www.google.co.in’)
```

- **Sample Program :**

```
import socket
host = ‘www.google.co.in’
Try:
                  addr = socket.gethostbyname(host)
                  print(‘IP Address= ‘ + addr)
Except socket.gaierror:
                  print(‘The website does not exist’)
```

## URL (Uniform Resource Locator)

- URL represents the address that is specified to access some information or resource on Internet.

  http://www.dreamtechpress.com:80/index.html

- The URL contains four parts:

1. The protocol to use. (http://)
2. The server name or IP address of the server. (www.dreamtechpress.com)
3. The third part represents port number, which is optional. (:80)
4. The last part is the file that is referred. (/index.html)

## URL

- We can pass the URL to urlparse() function, it returns a tuple containing the parts of the URL.

  tpl = urllib.parse.urlparse(‘urlstring’)

- The tuple ‘tpl’ contain the following :

1. **scheme** = this gives the protocol name used in the URL.
2. **netloc** = gives the website name on the net with port number if present.
3. **path** = gives the path of the web page.
4. **port** = gives the port number.

## READING THE SOURCE CODE OF A WEB PAGE

- If we know the URL of a Web page , it is possible to get the source code of the web page with the help of urlopen() function.
- Example :

```
import urllib.request

file = urllib.request.urlopen(https://www.python.org/)

print(file.read())
```

## DOWNLOADING A WEB PAGE FROM INTERNET

- First we should open the web page using the urlopen() function. This function returns the content of the web page into a file like object. We can read from this object using read() method as:

```
file = urllib.request.urlopen(https://www.python.org/)
content = file.read()
f = open(‘myfile.html’ , ’wb’)
f.write(content)
```

## TCP / IP SERVER

- A server is a program that provides services to other computers on the network or Internet.
- Similarly a client is a program that receives services from the servers.
- **Steps at Server Side** :

1. Create a TCP / IP socket at server side using socket() function:

   s = socket.socket(socket.AF_INET , socket.STREAM)

   s = socket.socket()  # this uses TCP / IP protocol by default

2. Bind the socket with host name and port number using bind() method.

   s.bind(host , port)  # here, (host,port) is a tuple.

   where host is a IP Address.

## TCP / IP SERVER

3. We can specify maximum number of connections using listen() method.

   s,listen(5)  # maximum connections allowed are 5

4. The server should wait till client accepts coonection. This is done using accept() method as :

   c,addr = s.accept()  # this method returs c and addr

5. Finally  using send(0 method , we can send message strings to client.

   c.send(b”message string”)   # ‘b’ prefixed before the message string indicates that the string is a binary string.

   The other method to convert a string into binary string is :

   string.encode()

6. After sending the messages to the client , the server can be disconnected by closing the connection object as : c.close()

## TCP / IP CLIENT

- **Steps at Client Side :**

1. At the client side , we should first create the socket.

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

2. Connect the socket to the server and port number using connect() method.

```
s.connect((host,port))
```

3. To receive the message from the server , we can use recv() method as:

```
msg = s.recv(1024)  # 1024 is the buffer size.
```

4. Finally , we should disconnect the client by calling close() method.

```
s.close()
```

## UDP SERVER

- If we want to create a server that uses UDP protocol to send messages , we have to specify

  s = socket.socket(socket.AF_INET , socket.SOCK_DGRAM)

- It means the server will send data in the form of packets called ‘datagrams’.
- Using sendto() function , the server can send data to the client.

  s.sendto(“message string”,(host , port))

  where message string is the binary string & tuple(host,port) represents the host name and port number of the client.

## UDP CLIENT

- At the client side , the socket should be created.

  s=socket.socket(socket.AF_INET , SOCKET.SOCK_DGRAM)

- The socket should be bound to the server using bind() methos as:

  s.bind((host,port))

- The client can receive message with the help of recvfrom() method as :\

  msg , addr = s.recvfrom(1024)   # 1024 is the buffer size.

# UDP CLIENT

We can use recvfrom() method inside a while loop and receive all the messages as :

```
while msg:
        print('Received: ' + msg.decode())
        msg , addr = s.recvfrom(1024)
```

- We have to set some time for the socket so that the client will automatically disconnect after that time elapses.

```
s.settimeout(5)
```

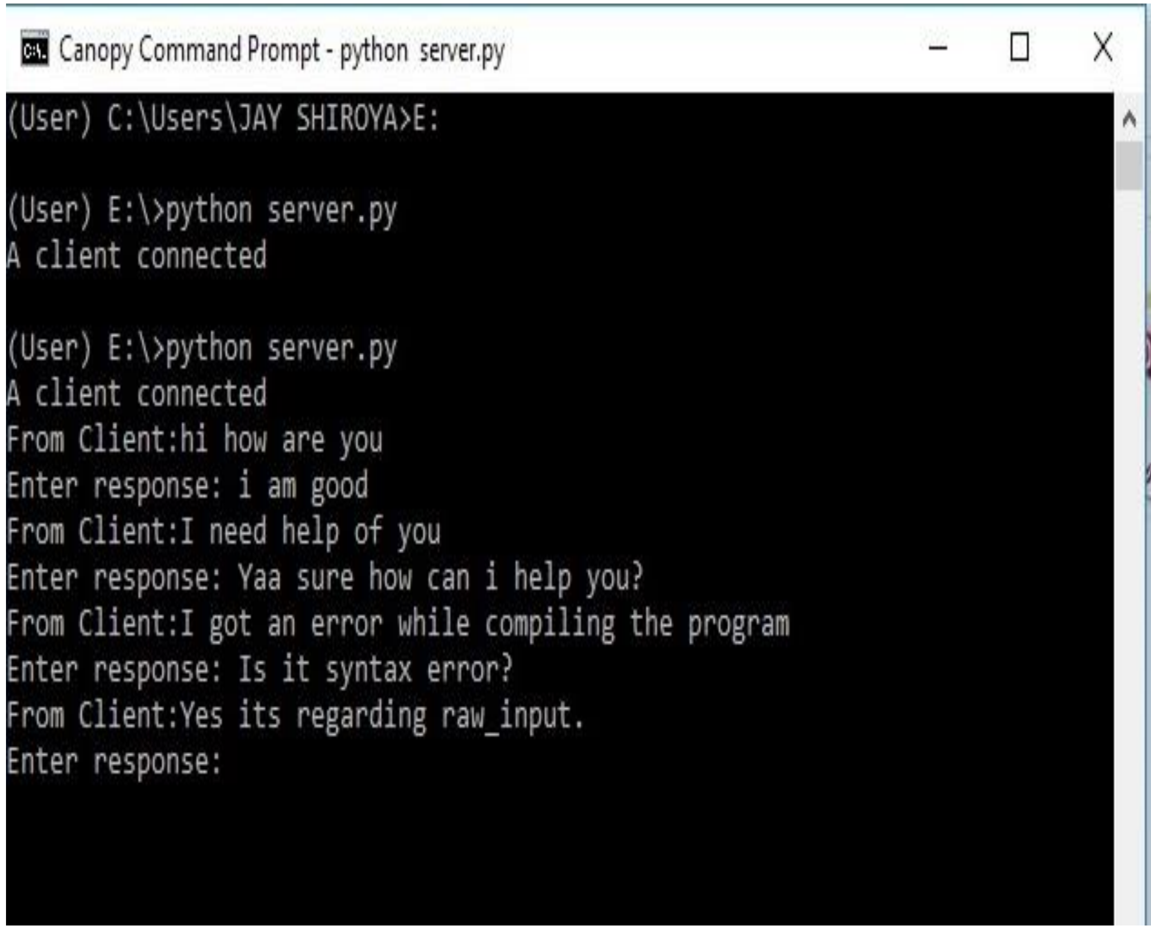
## CHAT APPLICATION(SERVER SIDE)

```
import socket
host = '127.0.0.1'
port = 9000
s = socket.socket()
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR , 1)
s.bind((host,port))
s.listen(1)
c,addr = s.accept()
print ("A client connected")
while True:
    data = c.recv(1024)
    if not data:
        break
    data=data.decode()
    print ("From Client:"+data)
    data1 = input("Enter response: ")
    c.send(data1.encode())
c.close()
```

## CHAT APPLICATION(CLIENT SIDE)

```
import socket
host = '127.0.0.1'
port = 9000
s = socket.socket()
s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR , 1)
s.connect((host,port))
str1 = input ("Enter data: ")
while str1 != 'exit':
    s.send(str1.encode())
    data = s.recv(1024)
    data = data.decode()
    print ("From server:"+data)
    str1 = input("Enter data: ")
s.close()
```

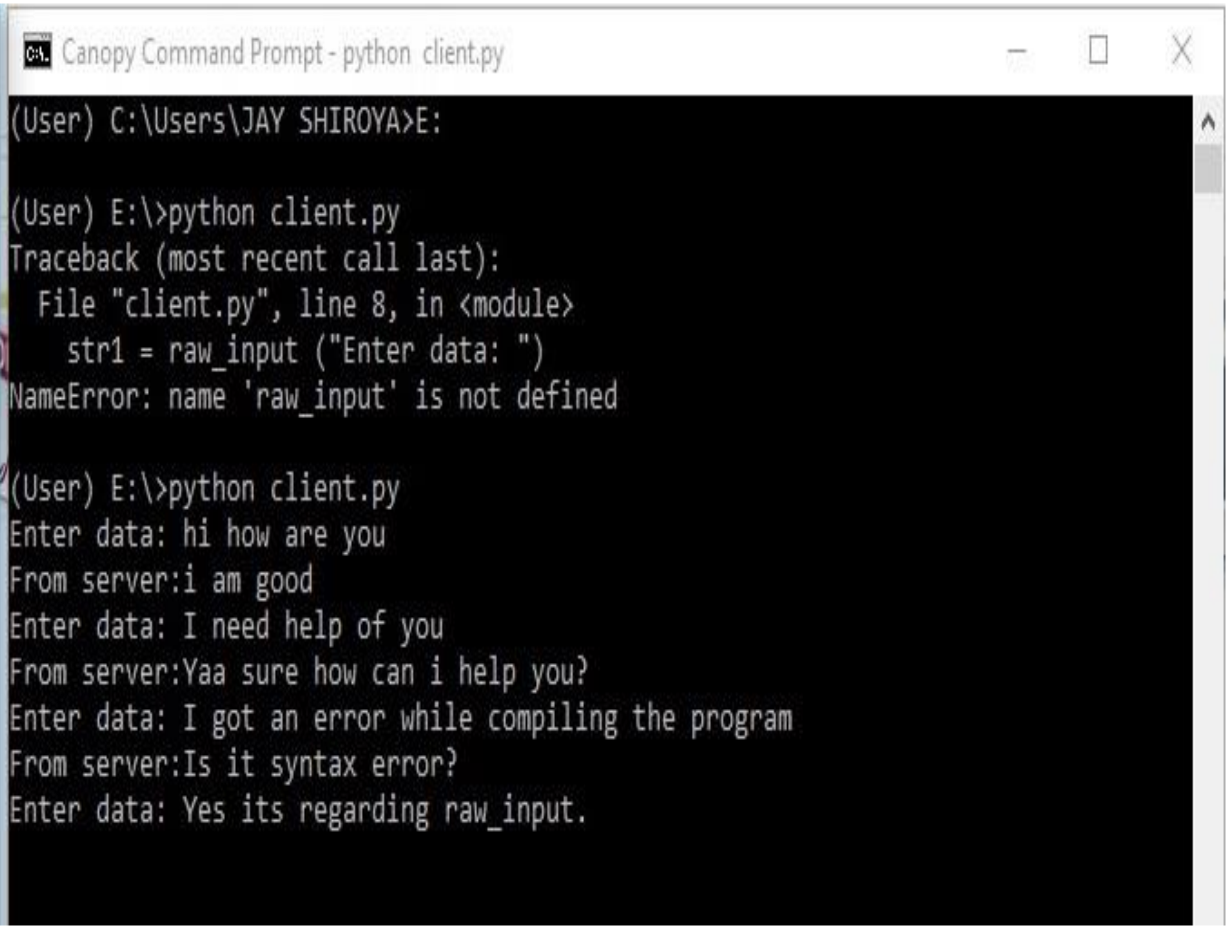
# OUTPUT:



```
(User) C:\Users\JAY SHIROYA>E:

(User) E:\>python server.py
A client connected

(User) E:\>python server.py
A client connected
From Client:hi how are you
Enter response: i am good
From Client:I need help of you
Enter response: Yaa sure how can i help you?
From Client:I got an error while compiling the program
Enter response: Is it syntax error?
From Client:Yes its regarding raw_input.
Enter response:
```



```
(User) C:\Users\JAY SHIROYA>E:

(User) E:\>python client.py
Traceback (most recent call last):
  File "client.py", line 8, in <module>
    str1 = raw_input ("Enter data: ")
NameError: name 'raw_input' is not defined

(User) E:\>python client.py
Enter data: hi how are you
From server:i am good
Enter data: I need help of you
From server:Yaa sure how can i help you?
Enter data: I got an error while compiling the program
From server:Is it syntax error?
Enter data: Yes its regarding raw_input.
```

## WRAPPER FUNCTIONS

- A function call that encompasses a call to a secondary function.
- Any parameters passed to the wrapper are passed to the secondary call
- The return value from the secondary call is returned to the original caller.
- A better approach to handle error in socket calls.

# gethostbyname()

- **Signature** :

  **gethostbyname(hostname)**

- **OverView** :

  1. Given a host name the **gethostbyname()** function returns the IP address of the host.
  2. The returned IP address is an IPv4 address.
  3. If the developer needs to resolve the hostname into IPv6 addresses or both IPv4 and IPv6 addresses are needed, **socket.getaddrinfo()** function can be used.
  4. When the parameter hostname is omitted hostname defaults to localhost.

## gethostname()

- Return a string containing the hostname of the machine where the Python interpreter is currently executing.
- If you want to know the current machine’s IP address, you may want to use **gethostbyname(gethostname()).** This operation assumes that there is a valid address-to-host mapping for the host, and the assumption does not always hold.

- Signature :

  **socket.gethostname()**

## gethostbtaddr()

- Return a triple **(hostname, aliaslist, ipaddrlist)** where *hostname* is the primary host name responding to the given *ip_address*, *aliaslist* is a (possibly empty) list of alternative host names for the same address, and *ipaddrlist* is a list of IPv4/v6 addresses for the same interface on the same host(most likely containing only a single address).

- Signature :

  **socket.gethostbyaddr(ip_address)**

## getservbyname()

- Translate an Internet service name and protocol name to a port number for that service. The optional protocol name, if given, should be ‘TCP' or ‘UDP', otherwise any protocol will match.

- **Signature :**

  **socket.getservbyname(servicename,protocolname)**

## getservbyport()

- Translate an Internet port number and protocol name to a service name for that service. The optional protocol name, if given, should be ‘TCP' or ‘UDP', otherwise any protocol will match.
- Signature :

  **socket.getservbyport(port,protocolname)**

## getpeername()

- Return the remote address to which the socket is connected. This is useful to find out the port number of a remote IPv4/v6 socket, for instance. On some systems this function is not supported.

- Signature :

  **socket.getpeername()**

## getsockname()

- Return the socket’s own address. This is useful to find out the port number of an IPv4/v6 socket, for instance.

- Signature :

```
socket.getsockname()
```

THANK YOU