# What is socket programming?

Socket programming is a communication between nodes (machine) over network.

A server is a software program that wait for client request and serve the incomming process, while a client is a requester of the service.

A client request resources from the server and the server respond to the server.

To establish, a connection between the `client` and the `server`. we create a `socket` in each program and then connect both socket together.

`Basic Concepts:`

- `Socket:` An endpoint for sending or receiving data across a network.
- `Server:` Listening for incoming connections from clients.
- `Client:` Connects to the server to send and receive data.

**`Server:`**

```
# server.py

import socket

# create server socket """OBJECT""" using socket(family, type, proto) method.
# socket.SOCK_STREAM :- Specify the type, this is default type
# socket.AF_INTER: Default socket family

# Step 1: Create a socket object
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Get machine hostname
host_name = socket.gethostname()
# Get machine host id
host_ip = socket.gethostbyname(host_name)

# Step 2: Bind the socket to a specific address and port
server_socket.bind((host_ip, 12345)) # 12345 it is a port number, you can choose
any port number

# Step 3: Start listening for connections (max backlog = 5)
server_socket.listen()

# Step 4: Accept a connection from a client
client_socket, client_address = server_socket.accept()
print(f"Connection from {client_address} has been established!")

# Step 5: Send a message to the client
client_socket.send("Hello, Client!".encode('utf-8'))
```

```
# Step 6: Close the connection
client_socket.close()
server_socket.close()
```

**Client:**

```
# client.py

import socket

# step 1: Create a socket object
client_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)

# Get hostname and host ip
host_name = socket.gethostname()
host_ip = socket.gethostbyname(host_name)

# step 2: Connect to the server
client_socket.connect((host_ip,12345))

# step 3: Receive data from the server
data  = client_socket.recv(1024)
print(data.encode('utf-8'))

# Step 4: Close the connection
client_socket.close()
```

`Explanation:`

- `socket.socket():` This creates a new socket using the Internet (`AF_INET`) address family and TCP (`SOCK_STREAM`).
- `connect(('localhost', 12345)):` This method connects the client to the server running on localhost at port 12345.
- `recv(1024):` Receives data from the server. The 1024 specifies the maximum amount of data (in bytes) to be received at once.
- `decode('utf-8'):` Decodes the received bytes into a UTF-8 string.
- `close():` Closes the connection after receiving the message.

**Methods:**

- `socket.socket(family, type, proto):`

  This method creates a new socket object. You must specify the family `(IPV4 or IPV6)` and the socket type `(TCP or UDP)`.

  `example:`

```python
import socket

# Create a TCP/IPV4 socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Create a UDP/IPV4 socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

`Parameters:`

- `AF_INET:` IPV4 (Internet protocol v4).
- `AF_INET6:` IPV4 (Internet protocol v6).
- `SOCK_STREAM:` TCP (Stream based socket).
- `SOCK_DGRAM:` UDP (Datagram based socket).

- `socket.bind((host, port)):`

  This method bind the socket to an `address` and a `port` number. It's used by server-side sockets to specify the network interface and port.

  `Example:`

  ```python
  sock.bind(('localhost',12345))
  ```

  `Parameters:`

  - `address:` A tuple of the IP address and port number, e.g., `('localhost', 12345)` or `('127.0.0.1', 8080)`.

- `socket.listen(max=max_connection_listen):`

  This method is used on server-side sockets to start listening for incoming connection requests. It takes an optional argument that defines the maximum number of queued connections.

  `Example:`

  ```python
  sock.listen(5)
  ```

- `socket.accept():`

  This method is called on a listening server socket to accept a new incoming connection. It returns a new socket object representing the `client` connection and the `client’s` address.

`Example:`

```
client_socket, client_address = sock.accept()
```

`Returns:`

- `client_socket:` A new socket object for communication with the client.
- `client_address:` The address of the client.

- `socket.send(data) / socket.sendAll()`

  These methods send data through the socket.

  - `send():` sends data but might not send all data in one go.
  - `sendAll():` ensures that all data is sent by sending repeatedly until all bytes have been transmitted.

  `Example:`

  ```
  # Send a message to the server
  client_socket.send(b"Hello Server")
  ```

  `Parameters:`

  - `data:` Data to be sent, as bytes. If sending a string, convert it to bytes using .encode().

- `socket.recv():`

  This method receives data from the socket.

  `Example:`

  ```
  data = client_socket.recv(1024)
  ```

  `Parameters:`

  - `pufsize:` The maximum amount of data (in bytes) to be received at once.

- `socket.close():`

  This method closes the socket and terminates the connection.

`Example:`

```
sock.close()
```

- `socket.settimeout():` This methods sets a timeout for blocking socket operations.(like `recv()`, `accept()`)

  `Example:`

  ```
  sock.settimeout(5.0)
  ```

- `socket.gettimeout():` This method retrieves the timeout value for the socket.

  `Example:`

  ```
  timeout_value = sock.gettimeout()

  # return the timeout value in seconds
  ```

- `socket.getpeername():` This method returns the address of the remote endpoint to which the socket is connected.

  `Example:`

  ```
  client_address = sock.getpeername()

  # The remote address (IP and port number).
  ```

- `socket.getsockname():` This method returns the socket's own address (the local IP and port).

  `Example:`

  ```
  local_address = sock.getsockname()

  # The socket's own address (IP and port).
  ```

- `socket.shutdown():` This method shuts down the socket's reception, transmission, or both. It is often used before closing a socket.

  `Example:`

  ```
  sock.shutdown(socket.SHUT_RDWR)  # Shut down both reading and writing
  ```

  `Parameters:` Specifies what to shut down. Possible values

  - `socket.SHUT_RD:` Disables further receptions.
  - `socket.SHUT_WR:` Disables further transmissions.
  - `socket.SHUT_RDWR:` Disables both receptions and transmissions.

- `socket.fileno():` This method returns the socket's file descriptor (a low-level identifier for the socket).

  `Example:`

  ```
  fd = sock.fileno()

  # The file descriptor of the socket as an integer.
  ```

- `socket.getsockopt():` This method retrieves a socket option.

  `Example:`

  ```
  optval = sock.getsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR)

  # The file descriptor of the socket as an integer.
  ```

  `Parameters:`

  - `level:` The protocol level at which the option resides, such as `socket.SOL_SOCKET`
  - `optname:` The name of the options e.g., `socket.SO_REUSEADDR`.

- `socket.gethostname():` This function returns the hostname of the local machine.

  `Example:`

```
hostname = socket.gethostname()

# The hostname of the local machine as a string.
```

- `socket.gethostbyname():` This function returns the hostname of the local machine.

  `Example:`

  ```
  ip_address  = socket.gethostbyname('www.google.com' OR `hostname`)

  # The hostname of the local machine as a string.
  ```

- `socket.getaddrinfo():` This function translates a hostname and service name to a list of address information.

  `Example:`

  ```
  addr_info = socket.getaddrinfo('localhost', 'http')

  # The hostname of the local machine as a string.
  ```

  `Parameter:`

  - `host:` The hostname to resolve.
  - `service:` The service name, such as `http`.

- `socket.inet_aton():` This function converts an IPv4 address from the standard dotted decimal notation into a 32-bit packed binary format.

  `Example:`

  ```
  binary_ip = socket.inet_aton('192.168.1.1')
  print(binary_ip)  # Outputs something like: b'\xc0\xa8\x01\x01'
  ```

  `Parameter:`

  - `ip_string`: A string representation of an IPv4 address, e.g., `'192.168.1.1'`.

- `socket.inet_ntoa():` This function converts a packed 32-bit binary IP address to its standard dotted decimal string format.

  `Example:`

  ```
  ip_string = socket.inet_ntoa(b'\xc0\xa8\x01\x01')
  print(ip_string)  # Outputs: '192.168.1.1'
  ```

  `Parameter:`

  - `packed_ip:` A 4-byte packed binary format of an IPv4 address.

- `socket.inet_pton():` This function converts an IP address from its text representation to its packed binary format. It supports both IPv4 and IPv6.

  `Example:`

  ```
  binary_ipv6 = socket.inet_pton(socket.AF_INET6, '2001:db8::1')
  binary_ipv4 = socket.inet_pton(socket.AF_INET, '192.168.1.1')
  ```

  `Parameter:`

  - `family:` The address family (`AF_INET` for `IPv4`, `AF_INET6` for `IPv6`).
  - `ip_string:` A string representation of the IP address (either `IPv4` or `IPv6`).

- `socket.inet_ntop():` This function converts a packed binary format IP address back to its text representation.

  `Example:`

  ```
  ip_string = socket.inet_ntop(socket.AF_INET6, binary_ipv6)
  ```

  `Parameter:`

  - `family:` The address family (`AF_INET` for `IPv4`, `AF_INET6` for `IPv6`).
  - `ip_string:` A string representation of the IP address (either `IPv4` or `IPv6`).

- `socket.create_connection():` This function simplifies the process of creating a client connection. It automatically handles socket creation and connection to a server.

  `Example:`

```python
conn = socket.create_connection(('www.google.com', 80))
```

`Parameter:`

- `address:` A tuple containing the IP or hostname and the port number of the server.
- `timeout:` (optional): A timeout value in seconds.

- `socket.fromfd():` This function creates a socket object from a file descriptor. It is used in low-level socket programming when interacting with raw file descriptors.

  `Example:`

  ```python
  import os
  fd = os.open('somefile', os.O_RDWR)  # Assume this file descriptor is valid for a socket
  sock = socket.fromfd(fd, socket.AF_INET, socket.SOCK_STREAM)
  ```

  `Parameter:`

  - `fd:` The file descriptor.
  - `family:` The address family (`AF_INET`, `AF_INET6`, etc.).
  - `type:` The socket type (`SOCK_STREAM`, `SOCK_DGRAM`, etc.).

- `socket.dup():` This method duplicates a socket object and returns a new socket object using the same file descriptor.

  `Example:`

  ```python
  dup_sock = sock.dup()
  ```

  `Parameter:`

  - `fd:` The file descriptor.
  - `family:` The address family (`AF_INET`, `AF_INET6`, etc.).
  - `type:` The socket type (`SOCK_STREAM`, `SOCK_DGRAM`, etc.).