The Request Response Cycle

Simple Letter Writing

Imagine you’re writing to a penpal. The process would look something like this:

1. Write a letter
2. Specify your penpal’s address
3. Drop the letter in your mailbox
4. The letter goes through the postal system and arrives at your penpal’s mailbox

Your penpal then goes through a very similar set of steps:

1. Read your letter and write a response
2. Specify your address
3. Drop their letter in their mailbox
4. The letter goes through the postal system and arrives at your mailbox

Analogy

You are the Client

Your penpal is the Server

Your letter is the Request

Your penpal’s letter is the Response

The postal system, the thing responsible for ensuring your letters are delivered, is The Internet

HTTP is the language you write in so that your penpal can understand you. You may write in English because you know you both understand English.

Client and Server

The basis of all web interactions is someone asking for information, and receiving information.

In order to ask for and receive any information, we need two players - the asker and the producer.

In basic web interactions, the ‘asker’ is a client and the ‘producer’ is a server.

Clients send Requests to Servers asking for some kind of information.

Upon receiving a Request, Servers send Responses back to the Client.

In the web development world, a client is a web browser, not an individual person.

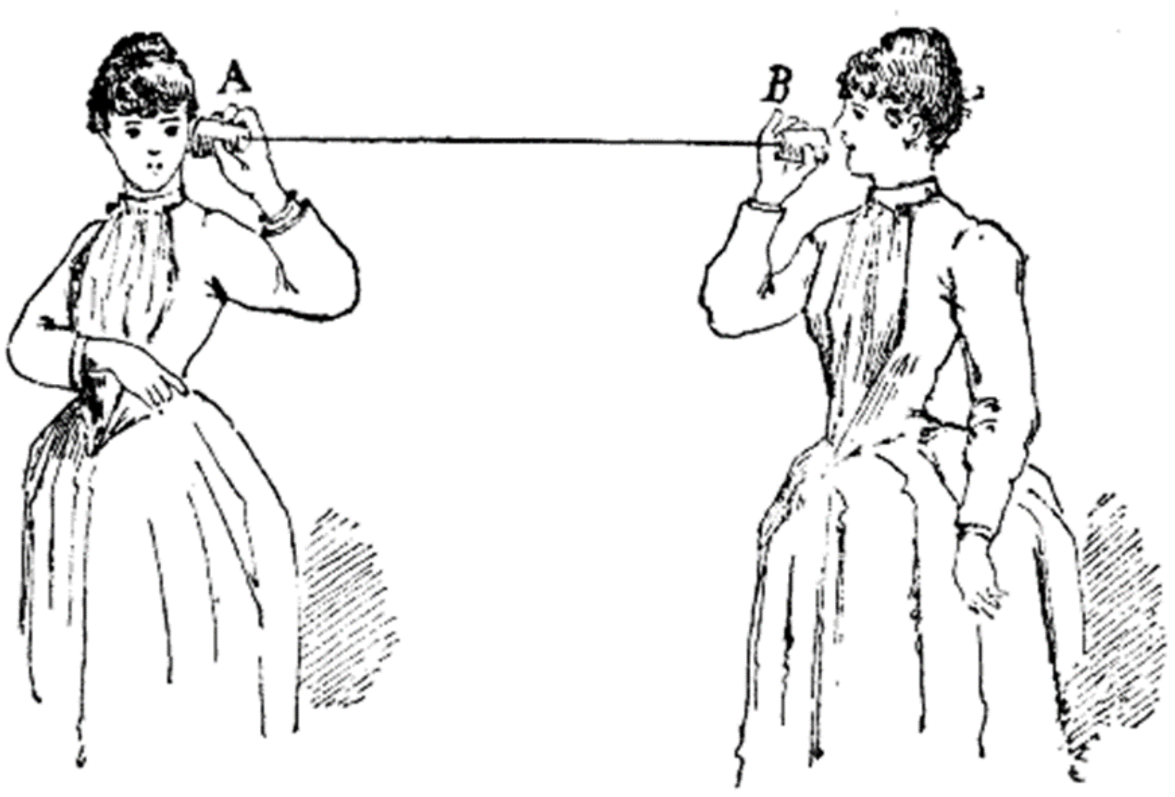
The person using the browser is referred to as a user.

Sockets

Sockets and the socket API are used to send messages across a network.

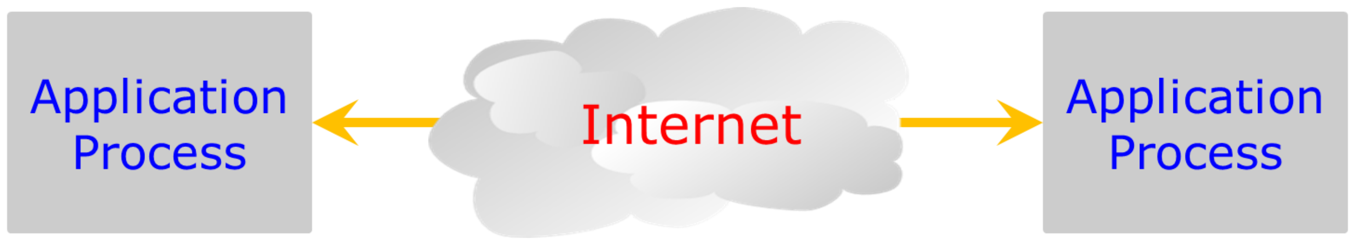
They provide a form of inter-process communication (IPC).

The network can be a logical, local network to the computer, or one that’s physically connected to an external network, with its own connections to other networks.



TCP Connections / Sockets

In computer networking, an Internet socket or network socket is an endpoint of a bidirectional inter-process communication flow across an Internet Protocol-based computer network, such as the Internet.”

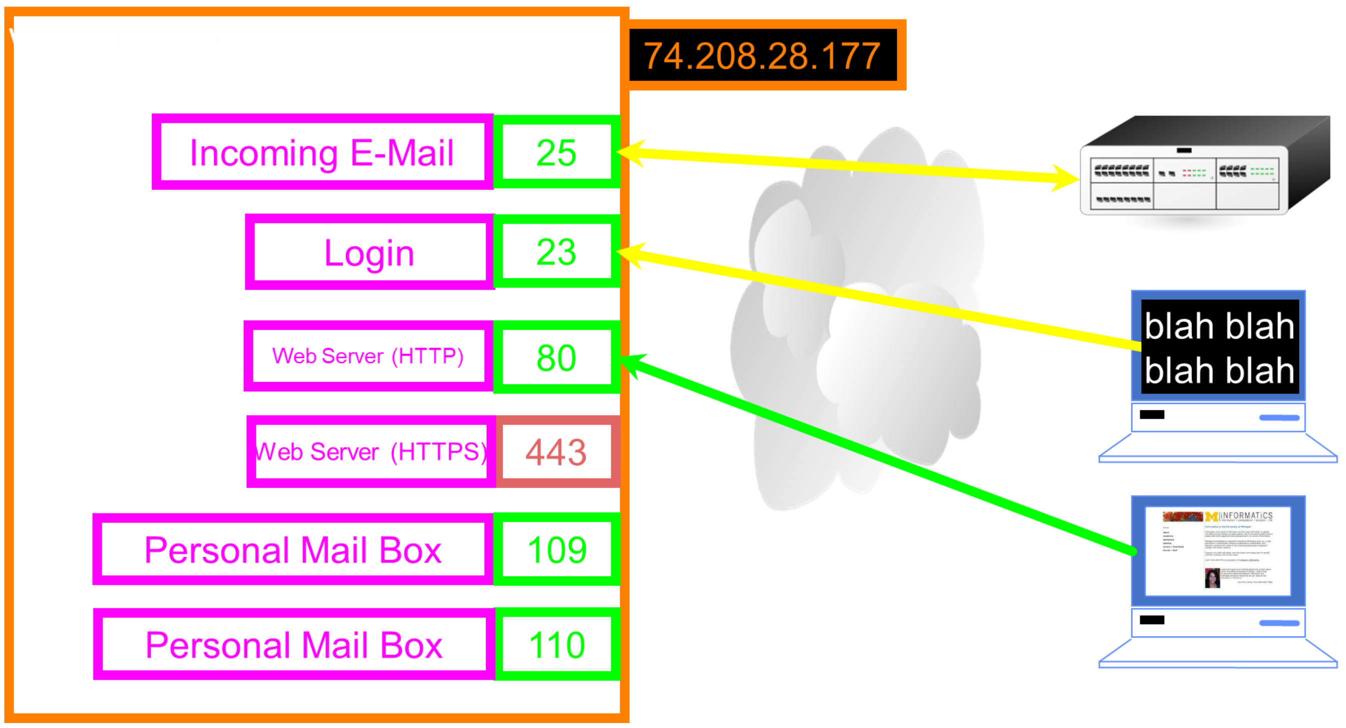


Port Numbers

A port is an application-specific or process-specific software communications endpoint

It allows multiple networked applications to coexist on the same server

There is a list of well-known port numbers



Uniform Resource Locator

http://data.pr4e.org/page1.htm

protocol host document

TCP v/s UDP

You’re going to create a socket object using socket.socket(), specifying the socket type as socket.SOCK_STREAM.

When you do that, the default protocol that’s used is the Transmission Control Protocol (TCP).

Why should you use TCP? The Transmission Control Protocol (TCP):

Is reliable: Packets dropped in the network are detected and retransmitted by the sender.

Has in-order data delivery: Data is read by your application in the order it was written by the sender.

In contrast, User Datagram Protocol (UDP) sockets created with socket.

SOCK_DGRAM aren’t reliable, and data read by the receiver can be out-of-order from the sender’s writes.

Although UDP isn’t ideal for sending an email, viewing a webpage, or downloading a file,

it is largely preferred for real-time communications like broadcast or multitask network transmission.

Python Socket Module

.socket()

.bind()

.listen()

.accept()

.connect()

.send()

.recv()

.close()

.gethostname()

.socket

Is like opening a file handle

.bind

The bind() method of Python's socket class assigns an IP address and a port number to a socket instance.

The bind() method is used when a socket needs to be made a server socket.

As server programs listen on published ports, it is required that a port and the IP address to be assigned explicitly to a server socket.

For client programs, it is not required to bind the socket explicitly to a port.

The kernel of the operating system takes care of assigning the source IP and a temporary port number.

The client socket can use the connect() method, after the socket creation is complete to contact the server socket.

.listen

Calling listen() makes a socket ready for accepting connections.

The listen() method should be called before calling the accept() method on the server socket.

.accept

The accept() method of Python's socket class, accepts an incoming connection request from a TCP client.

The accept() method is called on a TCP based server socket.

.connect

Used to connect with a server by passing IP address and port number

.send

The send()method of Python's socket class is used to send data from one socket to another socket.

The send()method can only be used with a connected socket. That is, send() can be used only with a TCP based socket and it can not be used with UDP socket.

The send() method can be used to send data from a TCP based client socket to a TCP based client-connected socket at the server side and vice versa.

The data sent should be in bytes format. String data can be converted to bytes by using the encode() method of string class.

.sendto

The method sendto() of the Python's socket class, is used to send datagrams to a UDP socket.

The communication could be from either side. It could be from client to server or from the server to client.

For sendto() to be used, the socket should not be in already connected state.

.recv()

The recv() method Python's socket class, reads a number of bytes sent from an TCP socket.

Like send(), the recv() method as well is to be called on a TCP socket.

.recvfrom

The recvfrom() method Python's socket class, reads a number of bytes sent from an UDP socket.

Like sendto(), the recvfrom() method as well is to be called on a UDP socket.

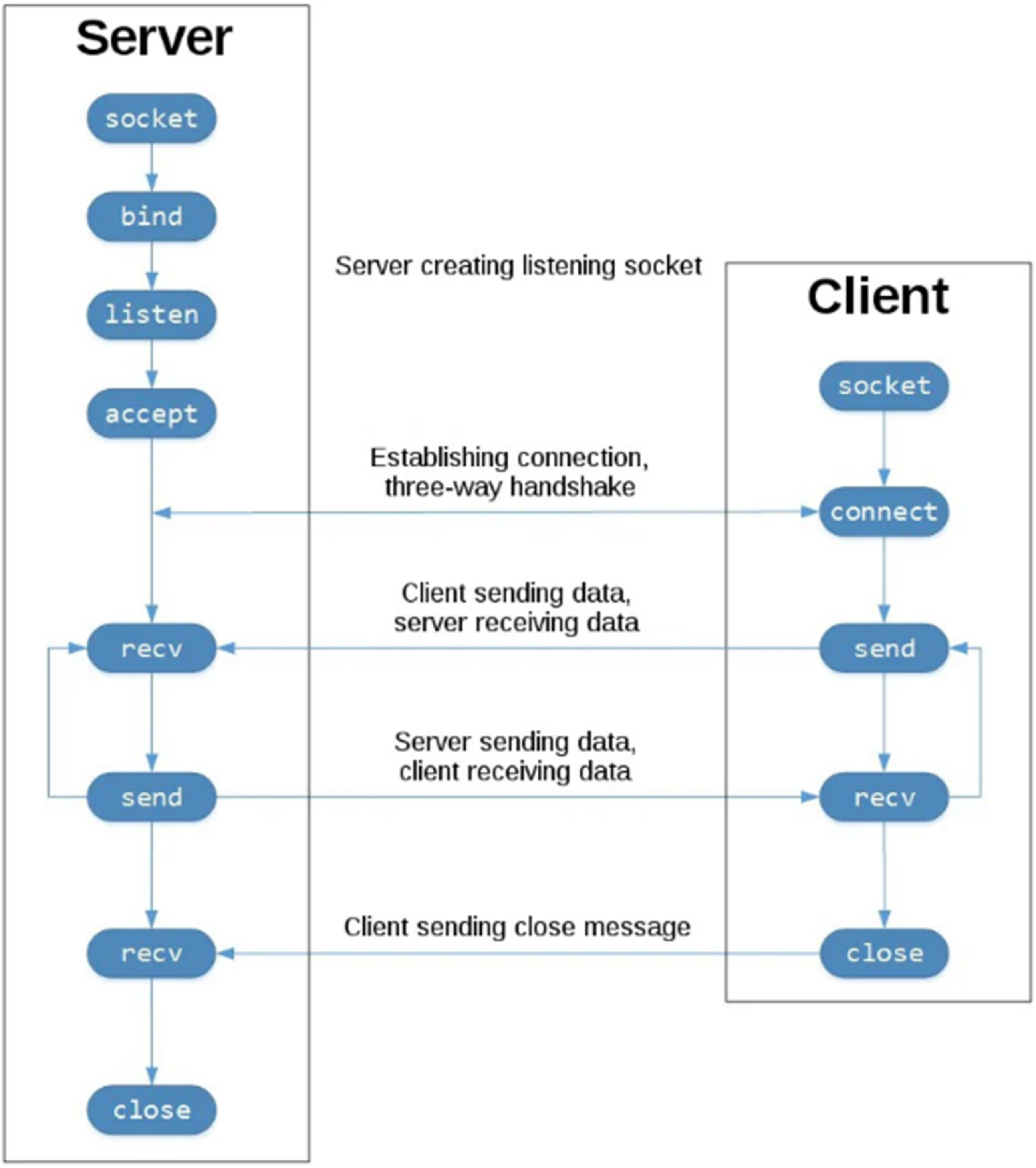
Unlike sendto(), the method recvfrom() does not take an IP address and port as a parameter.

The recvfrom() method can be used with an UDP server to receive data from a UDP client or it can be used with an UDP client to receive data from a UDP server.

.gethostname

The Python function socket.gethostname() returns the host name of the current system under which the Python interpreter is executed.

TCP Client Server Flow


Server
socket
bind
listen
accept
recv
send
recv
close
Server creating listening socket
Establishing connection, three-way handshake
Client sending data, server receiving data
Server sending data, client receiving data
Client sending close message
Client
socket
connect
send
recv
close

Internet

The Internet is the network between devices that allows clients and servers to exchange this information.

HTTP (HyperText Transfer Protocol)

HTTP is a set of rules for how this exchange of information happens. Clients and Servers adhere to these rules to ensure that they understand each other’s Requests and Responses.

HTTP Request Response Cycle

1. You open your browser, the Client, and type in a web address
2. The browser takes this address and builds an HTTP Request. It addresses it to the Server
3. The Request is handed off to your Internet Service Provider (ISP) and they send it through the Internet, mostly a series of wires and fiber optic cables, to the Server
4. The Server reads the Request. It knows how to read it because it is formatted as an HTTP Request.
5. The Server generates an HTTP Response to that Request.
6. The server hands the Response off to their ISP and it goes through the internet to arrive at your computer.
7. Your browser reads the Response. It knows how to read it because it is formatted as an HTTP Response.
8. Your browser displays the data on your machine.

GET hostname ,ipaddress and port

In [4]:

```
import socket

# Get the hostname of the current machine
hostname = socket.gethostname()

# Get the IP address associated with the hostname
ip_address = socket.gethostbyname(hostname)

# Create a dummy socket to get the port
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind(('localhost', 0))  # Bind to a random available port
    _, port = s.getsockname()

# Print the IP address and port
print("Hostname:", hostname)
print("IP Address:", ip_address)
print("Port:", port)
```

```
Hostname: SYCEIT309A-186
IP Address: 192.168.103.186
Port: 59209
```

Program for TCP Server client connection

In [ ]:

```
%%writefile Server_TCP.py
import socket
host=socket.gethostname()
port=2000
Addr=(host,port)
server_socket=socket.socket()
server_socket.bind(Addr)
server_socket.listen()
conn,addr=server_socket.accept()
print("Connection from ",str(addr))
while True:
    data=conn.recv(1024).decode()
    if not data:
        break
    print(data)
    data=input("->")
    conn.send(data.encode())
conn.close()
```

In [ ]:

```
%%writefile Client_TCP.py
import socket
host=socket.gethostname()
port=2000
client_socket=socket.socket()
client_socket.connect((host,port))
message=input("->")
while message!="":
    client_socket.send(message.encode())
    data=client_socket.recv(1024).decode()
    print("Received from Server -> ",data)
    message=input('->')
client_socket.close()
```

Program for TCP Server client connection with other PC

In [ ]:

```
%%writefile Server_TCP1.py
import socket
host=socket.gethostname()
port=2000
Addr=(host,port)
server_socket=socket.socket()
server_socket.bind(Addr)
server_socket.listen()
conn,addr=server_socket.accept()
print("Connection from ",str(addr))
while True:
    data=conn.recv(1024).decode()
    if not data:
        break
    print(data)
    data=input("->")
    conn.send(data.encode())
conn.close()
```

In [ ]:

```
%%writefile Client_TCP1.py
import socket
host="192.168.107.27"
port=2000
client_socket=socket.socket()
client_socket.connect((host,port))
message=input("->")
while message!="":
    client_socket.send(message.encode())
    data=client_socket.recv(1024).decode()
    print("Received from Server -> ",data)
    message=input('->')
client_socket.close()
```

Program for UDP Server client connection

In [ ]:

```
%%writefile udp_server.py
import socket
host=""
port=5000
udp_server=socket.socket(type=socket.SOCK_DGRAM)
udp_server.bind((host,port))
while True:
    print("Waiting for Message ")
    data,addr=udp_server.recvfrom(1024)
    print("Received",data.decode(),"from",addr)
    msg=input("Enter msg:")
    udp_server.sendto(msg.encode(),addr)
udp_server.close()
```

In [ ]:

```
%%writefile udp_client.py
import socket
host="localhost"
port=5000
udp_client=socket.socket(type=socket.SOCK_DGRAM)
while True:
    data=input("Enter data to Send:")
    if not data:
        break
    udp_client.sendto(data.encode(),(host,port))
    print("Ready to Receive Data")
    data,addr=udp_client.recvfrom(1024)
    if not data:
        break
    print("Received",data.decode())
udp_client.close()
```

In [ ]:

```
%%writefile udp_client1.py
import socket
host="192.168.107.27"
port=5000
udp_client=socket.socket(type=socket.SOCK_DGRAM)
while True:
    data=input("Enter data to Send:")
    if not data:
        break
    udp_client.sendto(data.encode(),(host,port))
    print("Ready to Receive Data")
    data,addr=udp_client.recvfrom(1024)
    if not data:
        break
    print("Received",data.decode())
udp_client.close()
```

Program for web networking

In [ ]:

```
import socket
mysock=socket.socket()
mysock.connect(("www.ljku.edu.in",80))
cmd="GET https://www.ljku.edu.in/lju-at-a-glance HTTP/1.0\n\n".encode()
mysock.send(cmd)
while True:
    data=mysock.recv(1024)
    if len(data)<1:
        break
    print(data.decode(),end="")
mysock.close()
```

Program for HTTP Server connection

In [ ]:

```
from socket import *

def createServer():
    serversocket = socket()
    serversocket.bind(('localhost',9050))
    serversocket.listen()
    while(1):
        (clientsocket, address) = serversocket.accept()

        rd = clientsocket.recv(5000).decode()
        print(rd)

        data = """HTTP/1.1 200 OK\nContent-Type:text/html; charset=utf-8\n\n<html><body><
        clientsocket.send(data.encode())

    serversocket.close()

print('Access http://localhost:9050')
createServer()
```

```
Access http://localhost:9050 (http://localhost:9050)
GET / HTTP/1.1
Host: localhost:9050
Connection: keep-alive
sec-ch-ua: "Not.A/Brand";v="8", "Chromium";v="114", "Microsoft Edge";v="11
4"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36 Edg/114.0.1823.43
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,i
mage/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:8890/ (http://localhost:8890/)
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: _xsrf=2|660f51e6|281f1a0ce8e5d046e48ca60c852b79b4|1684309766; user
name-localhost-8888="2|1:0|10:1686720217|23:username-localhost-8888|44:YTJ
lZTQwYTJlYmEzNDA4Mzg2YjczNDRlZGM0YTdkMDg=|6ef94206aafaa131f858dc38ebd9327c
8e58988f0f1b80e22d89c7938e40c037"; username-localhost-8889="2|1:0|10:16867
29673|23:username-localhost-8889|44:YWZmM2UyNDhjODQ3NDRlODliYTdiYmZhODEzNT
Y4NjU=|48a2508fc5b3d1f30c2b9ede530d5d2a9d779e64922be0bd1a22dafbc2031fce";
username-localhost-8890="2|1:0|10:1686729727|23:username-localhost-8890|4
4:N2JiNGE3ZmI3MjJmNGY3ZjljY2FjNjVjYWVmOTNhNzA=|a76d4732a3fa803460c62cb723a
45e09f69f6b8b27399202b3a6d5e89a78ad1a"


GET /favicon.ico HTTP/1.1
Host: localhost:9050
Connection: keep-alive
sec-ch-ua: "Not.A/Brand";v="8", "Chromium";v="114", "Microsoft Edge";v="11
4"
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/114.0.0.0 Safari/537.36 Edg/114.0.1823.43
sec-ch-ua-platform: "Windows"
Accept: image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: http://localhost:9050/ (http://localhost:9050/)
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: _xsrf=2|660f51e6|281f1a0ce8e5d046e48ca60c852b79b4|1684309766; user
name-localhost-8888="2|1:0|10:1686720217|23:username-localhost-8888|44:YTJ
lZTQwYTJlYmEzNDA4Mzg2YjczNDRlZGM0YTdkMDg=|6ef94206aafaa131f858dc38ebd9327c
8e58988f0f1b80e22d89c7938e40c037"; username-localhost-8889="2|1:0|10:16867
29673|23:username-localhost-8889|44:YWZmM2UyNDhjODQ3NDRlODliYTdiYmZhODEzNT
Y4NjU=|48a2508fc5b3d1f30c2b9ede530d5d2a9d779e64922be0bd1a22dafbc2031fce";
username-localhost-8890="2|1:0|10:1686729727|23:username-localhost-8890|4
4:N2JiNGE3ZmI3MjJmNGY3ZjljY2FjNjVjYWVmOTNhNzA=|a76d4732a3fa803460c62cb723a
45e09f69f6b8b27399202b3a6d5e89a78ad1a"
```

What is an API?

An application programming interface is a way for two or more computer programs to communicate with each other. It is a type of software interface, offering a service to other pieces of software. A document or standard that describes how to build or use such a connection or interface is called an API specification.

The OpenWeatherMap API https://openweathermap.org/api (https://openweathermap.org/api)

The OpenWeatherMap is a service that provides weather data, including current weather data, forecasts, and historical data to the developers of web services and mobile applications.

It provides an API with JSON, XML, and HTML endpoints and a limited free usage tier. Making more than 60 calls per minute requires a paid subscription starting at USD 40 per month. Access to historical data requires a subscription starting at 150 USD per month. Users can request current weather information, extended forecasts, and graphical maps (showing cloud cover, wind speed, pressure, and precipitation).

To use this current weather data API, one must need the API key.

Pricing and API Key

https://openweathermap.org/price (https://openweathermap.org/price)

In [1]:

```
import requests, json
```

In [2]:

```
api_key = input('Enter your API key: ')
city = input('Enter the city name: ')
request_url = "http://api.openweathermap.org/geo/1.0/direct?q=" + city + "&appid=" + api_
response = requests.get(request_url)
coordinates = response.json()
print(coordinates)
lat= coordinates[0]['lat']
long = coordinates[0]['lon']
```

```
Enter your API key: 81c148fded66cf6945b9fe227e878f6e
Enter the city name: Ahmedabad
[{'name': 'Ahmedabad', 'local_names': {'ascii': 'Ahmedabad', 'ja': 'アフマ
ダーバード', 'ar': 'أحمد آباد', 'de': 'Ahmedabad', 'te': 'అహ్మదాబాద్', 'mr':
'अहमदाबाद', 'eo': 'Ahmadabado', 'he': 'אחמדאבאד', 'ml': 'അഹമ്മദാബാദ്',
'ru': 'Ахмадабад', 'gu': 'અમદાવાદ', 'ta': 'அகமதாபாத்', 'pl': 'Ahmadaba
d', 'zh': '艾哈迈达巴德', 'feature_name': 'Ahmedabad', 'oc': 'Ahmadabad',
'kn': 'ಅಹ್ಮದಾಬಾದ್', 'hi': 'अहमदाबाद', 'or': 'ଅହମଦାବାଦ', 'uk': 'Ахмедаба
д', 'cs': 'Ahmadábád', 'pa': 'ਅਹਿਮਦਾਬਾਦ', 'ur': 'احمد آباد', 'en': 'Ahmedaba
d'}, 'lat': 23.0216238, 'lon': 72.5797068, 'country': 'IN', 'state': 'Guja
rat'}]
```

In [3]:

```
print(lat)
```

```
23.0216238
```

In [4]:

```
print(long)
```

```
72.5797068
```

Current Weather API https://openweathermap.org/current (https://openweathermap.org/current)

Used to obtain current weather details of any city

In [5]:

```
api_key = input('Enter your API key: ')
request_url = "https://api.openweathermap.org/data/2.5/weather?lat="+str(lat)+"&lon="+str
response = requests.get(request_url)
current_weather = response.json()
print(json.dumps(current_weather, indent = 5))
dscription = current_weather['weather'][0]['description']
temperature = current_weather['main']['temp']
pressure = current_weather['main']['pressure']
humidity = current_weather['main']['humidity']
wind_speed = current_weather['wind']['speed']
visibility = current_weather['visibility']
print('Current Weather')
print('Temperature', temperature)
print('Pressure', pressure)
print('Humidity', humidity)
print('Wind Speed', wind_speed)
print('Visibility', visibility)
```

```
Enter your API key: 81c148fded66cf6945b9fe227e878f6e
```

```
{
     "coord": {
          "lon": 72.5797,
          "lat": 23.0216
     },
     "weather": [
          {
               "id": 711,
               "main": "Smoke",
               "description": "smoke",
               "icon": "50d"
          }
     ],
     "base": "stations",
     "main": {
          "temp": 305.19,
          "feels_like": 310.94,
          "temp_min": 305.19,
          "temp_max": 305.19,
          "pressure": 1004,
          "humidity": 62
     },
     "visibility": 5000,
     "wind": {
          "speed": 4.12,
          "deg": 220
     },
     "clouds": {
          "all": 40
     },
     "dt": 1686371641,
     "sys": {
          "type": 1,
          "id": 9049,
          "country": "IN",
          "sunrise": 1686356617,
          "sunset": 1686405277
     },
     "timezone": 19800,
     "id": 1279233,
     "name": "Ahmedabad",
     "cod": 200
}
```

```
Current Weather
Temperature 305.19
Pressure 1004
Humidity 62
Wind Speed 4.12
Visibility 5000
```

Air Pollution API https://openweathermap.org/api/air-pollution (https://openweathermap.org/api/air-pollution)

Used to obtain current air pollution data, air pollution forecast and also past air pollution data of any city

##Current air pollution data

In [6]:

```
quality_indices = {1:'Good', 2:'Fair', 3:'Moderate', 4:'Poor', 5:'Very Poor'}
def display_current_air_pollution():
  api_key = input('Enter your API key: ')
  request_url = "http://api.openweathermap.org/data/2.5/air_pollution?lat=" + str(lat) +
  response = requests.get(request_url)
  current_pollution = response.json()
  print(json.dumps(current_pollution, indent = 5))
  index = current_pollution['list'][0]['main']['aqi']
  print('Current Air Quality: ', quality_indices[index])

display_current_air_pollution()
```

```
Enter your API key: 81c148fded66cf6945b9fe227e878f6e
{
     "coord": {
          "lon": 72.5797,
          "lat": 23.0216
     },
     "list": [
          {
               "main": {
                    "aqi": 1
               },
               "components": {
                    "co": 223.64,
                    "no": 0.32,
                    "no2": 1.23,
                    "o3": 52.21,
                    "so2": 1.36,
                    "pm2_5": 3.18,
                    "pm10": 7.36,
                    "nh3": 1.66
               },
               "dt": 1686371850
          }
     ]
}
Current Air Quality:  Good
```

Exercise: Generate Weather Forecast for the next 5 Days in Ahmedabad:

https://openweathermap.org/forecast5 (https://openweathermap.org/forecast5)

In [7]:

```
api_key = input('Enter your API key: ')
request_url = "http://api.openweathermap.org/data/2.5/forecast?lat=" + str(lat) + "&lon="
response = requests.get(request_url)
five_day = response.json()
print(json.dumps(five_day, indent = 5))
```

```
Enter your API key: 81c148fded66cf6945b9fe227e878f6e
```

```
{
     "cod": "200",
     "message": 0,
     "cnt": 40,
     "list": [
          {
               "dt": 1686376800,
               "main": {
                    "temp": 32.04,
                    "feels_like": 37.79,
                    "temp_min": 32.04,
                    "temp_max": 37.11,
                    "pressure": 1004,
                    "sea_level": 1004,
                    "grnd_level": 999,
                    "humidity": 62,
                    "temp_kf": -5.07
               },
```

In [9]:

```
D = {"date_time":[], "temp":[], "pressure":[], "humidity":[], "weather":[]}

for i in five_day['list']:
  D["date_time"].append(i["dt_txt"])
  D["temp"].append(i["main"]["temp"])
  D["pressure"].append(i["main"]["temp"])
  D["humidity"].append(i["main"]["humidity"])
  D["weather"].append(i["weather"][0]["description"])
import pandas as pd
df = pd.DataFrame(D)
df.head(40)
```

Out[9]:

	date_time	temp	pressure	humidity	weather
0	2023-06-10 06:00:00	32.04	32.04	62	scattered clouds
1	2023-06-10 09:00:00	35.06	35.06	49	scattered clouds
2	2023-06-10 12:00:00	37.39	37.39	40	few clouds
3	2023-06-10 15:00:00	34.28	34.28	49	clear sky
4	2023-06-10 18:00:00	32.07	32.07	58	clear sky
5	2023-06-10 21:00:00	30.65	30.65	60	clear sky
6	2023-06-11 00:00:00	30.00	30.00	63	clear sky
7	2023-06-11 03:00:00	32.03	32.03	56	clear sky
8	2023-06-11 06:00:00	36.93	36.93	40	few clouds
9	2023-06-11 09:00:00	38.94	38.94	33	few clouds
10	2023-06-11 12:00:00	35.85	35.85	40	few clouds
11	2023-06-11 15:00:00	31.95	31.95	56	clear sky
12	2023-06-11 18:00:00	30.31	30.31	64	scattered clouds
13	2023-06-11 21:00:00	29.85	29.85	66	overcast clouds
14	2023-06-12 00:00:00	29.27	29.27	71	broken clouds
15	2023-06-12 03:00:00	32.04	32.04	57	few clouds
16	2023-06-12 06:00:00	36.65	36.65	37	few clouds
17	2023-06-12 09:00:00	37.92	37.92	33	few clouds
18	2023-06-12 12:00:00	35.99	35.99	36	scattered clouds
19	2023-06-12 15:00:00	31.73	31.73	57	scattered clouds
20	2023-06-12 18:00:00	30.09	30.09	70	scattered clouds
21	2023-06-12 21:00:00	29.46	29.46	74	few clouds
22	2023-06-13 00:00:00	28.96	28.96	75	scattered clouds
23	2023-06-13 03:00:00	31.74	31.74	61	overcast clouds
24	2023-06-13 06:00:00	36.44	36.44	41	broken clouds
25	2023-06-13 09:00:00	37.72	37.72	37	few clouds
26	2023-06-13 12:00:00	35.88	35.88	43	clear sky
27	2023-06-13 15:00:00	31.15	31.15	67	clear sky
28	2023-06-13 18:00:00	30.21	30.21	71	scattered clouds
29	2023-06-13 21:00:00	29.61	29.61	72	few clouds
30	2023-06-14 00:00:00	29.17	29.17	75	scattered clouds
31	2023-06-14 03:00:00	31.53	31.53	62	light rain
32	2023-06-14 06:00:00	35.47	35.47	47	light rain
33	2023-06-14 09:00:00	36.96	36.96	43	broken clouds
34	2023-06-14 12:00:00	35.33	35.33	47	broken clouds
35	2023-06-14 15:00:00	32.80	32.80	58	broken clouds
36	2023-06-14 18:00:00	30.52	30.52	70	broken clouds

	date_time	temp	pressure	humidity	weather
37	2023-06-14 21:00:00	29.88	29.88	72	scattered clouds
38	2023-06-15 00:00:00	29.76	29.76	72	light rain
39	2023-06-15 03:00:00	32.25	32.25	60	light rain

```
In [ ]:
```