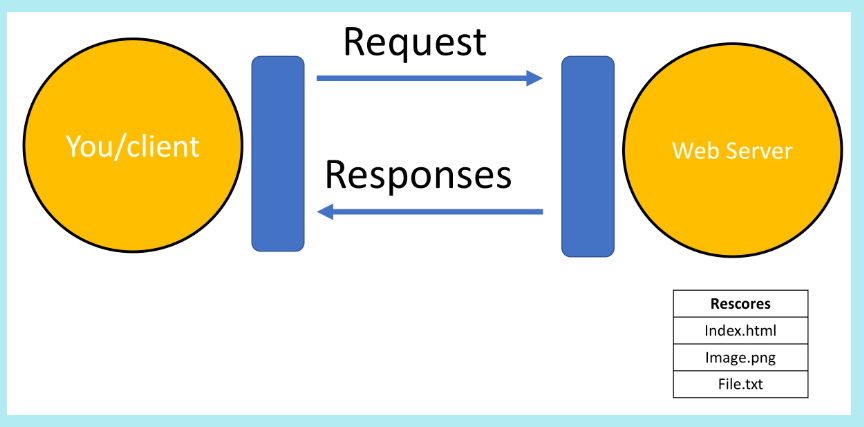
**HTTP and Requests**

## Overview of HTTP

When you, the **client**, use a web page your browser sends an **HTTP** request to the **server** where the page is hosted. The server tries to find the desired **resource** by default "index.html". If your request is successful, the server will send the object to the client in an **HTTP response**. This includes information like the type of the **resource**, the length of the **resource**, and other information.

The figure below represents the process. The circle on the left represents the client, the circle on the right represents the Web server. The table under the Web server represents a list of resources stored in the web server. In this case an HTML file, png image, and txt file .

The **HTTP** protocol allows you to send and receive information through the web including webpages, images, and other web resources. In this lab, we will provide an overview of the Requests library for interacting with the HTTP protocol.



## Uniform Resource Locator: URL

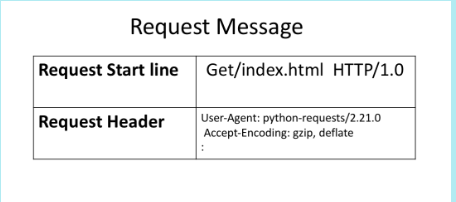
Uniform resource locator (URL) is the most popular way to find resources on the web. We can break the URL into three parts.

* **scheme** this is this protocol, for this lab it will always be http://
* **Internet address or Base URL**this will be used to find the location here are some examples: www.ibm.com and  www.gitlab.com
* **route** location on the web server for example: /images/IDSNlogo.png

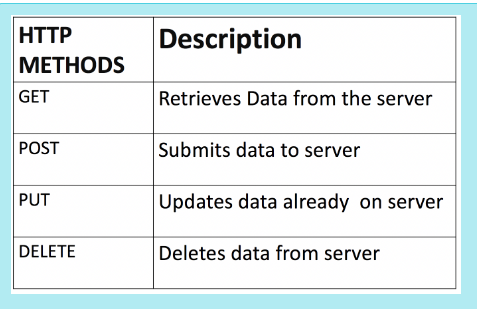
You may also hear the term Uniform Resource Identifier (URI), URL are actually a subset of URIs. Another popular term is endpoint, this is the URL of an operation provided by a Web server.

## Request

The process can be broken into the **request** and **response**process. The request using the get method is partially illustrated below. In the start line we have the GET method, this is an HTTP method. Also the location of the resource /index.html and the HTTP version. The Request header passes additional information with an HTTP request:

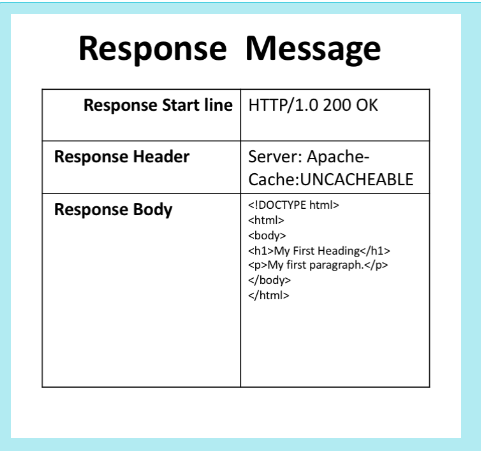


When an HTTP request is made, an HTTP method is sent, this tells the server what action to perform. A list of several HTTP methods is shown below. We will go over more examples later.

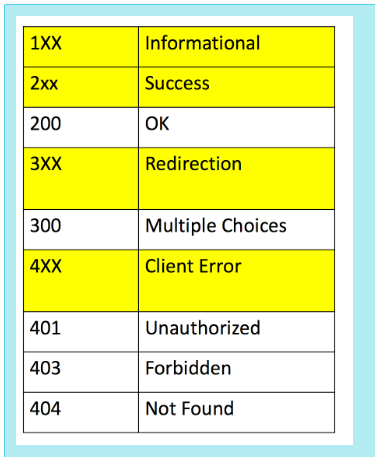


## Response

The figure below represents the response; the response start line contains the version number HTTP/1.0, a status code (200) meaning success, followed by a descriptive phrase (OK). The response header contains useful information. Finally, we have the response body containing the requested file, an  HTML  document. It should be noted that some requests have headers.



Some status code examples are shown in the table below, the prefix indicates the class. These are shown in yellow, with actual status codes shown in white. Check out the following [link](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork1005-2022-01-01)for more descriptions.



## Requests in Python

Requests is a Python Library that allows you to send HTTP/1.1 requests easily. We can import the library as follows:

import requests

We will also use the following libraries:

import os

from PIL import Image

from IPython.display import IFrame

You can make a GET request via the method get to [www.ibm.com](http://www.ibm.com/):

url='https://www.ibm.com/'

r=requests.get(url)

We have the response object r, this has information about the request, like the status of the request. We can view the status code using the attribute status\_code.

r.status\_code

200

You can view the request headers:

print(r.request.headers)

You can view the request body, in the following line, as there is no body for a get request we get a None:

print("request body:", r.request.body)

request body: None

You can view the HTTP response header using the attribute headers. This returns a python dictionary of HTTP response headers.

header=r.headers

print(r.headers)

{'Accept-Ranges': 'bytes', 'Content-Type': 'text/html', 'ETag': '"c1633ffa71e9f7915c50a7845e2c093e:1683595203.234443"', 'Last-Modified': 'Tue, 09 May 2023 00:02:09 GMT', 'Server': 'AkamaiNetStorage', 'Cache-Control': 'max-age=3600', 'Expires': 'Thu, 11 May 2023 14:14:06 GMT', 'X-Akamai-Transformed': '9 18381 0 pmb=mTOE,2', 'Content-Encoding': 'gzip', 'Date': 'Thu, 11 May 2023 13:14:06 GMT', 'Content-Length': '18572', 'Connection': 'keep-alive', 'Vary': 'Accept-Encoding', 'x-content-type-options': 'nosniff', 'X-XSS-Protection': '1; mode=block', 'Content-Security-Policy': 'upgrade-insecure-requests', 'Strict-Transport-Security': 'max-age=31536000'}

We can obtain the date the request was sent using the key Date

header['date']

'Thu, 11 May 2023 13:14:06 GMT'

Content-Type indicates the type of data:

header['Content-Type']

'text/html'

You can also check the encoding:

r.encoding

'ISO-8859-1'

As the Content-Type is text/html we can use the attribute text to display the HTML in the body. We can review the first 100 characters:

r.text[0:100]

'<!DOCTYPE html><html lang="en-US"><head><meta name="viewport" content="width=device-width"/><meta ch'

You can load other types of data for non-text requests, like images. Consider the URL of the following image:

*# Use single quotation marks for defining string*

url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/IDSNlogo.png'

We can make a get request:

r=requests.get(url)

We can look at the response header:

print(r.headers)

{'Date': 'Thu, 11 May 2023 13:45:50 GMT', 'X-Clv-Request-Id': '149740b9-bfb3-46c5-ba09-07969201e5fc', 'Server': 'Cleversafe', 'X-Clv-S3-Version': '2.5', 'Accept-Ranges': 'bytes', 'x-amz-request-id': '149740b9-bfb3-46c5-ba09-07969201e5fc', 'ETag': '"8bb44578fff8fdcc3d2972be9ece0164"', 'Content-Type': 'image/png', 'Last-Modified': 'Wed, 16 Nov 2022 03:32:41 GMT', 'Content-Length': '78776'}

We can see the 'Content-Type'

r.headers['Content-Type']

'image/png'

An image is a response object that contains the image as a [bytes-like object](https://docs.python.org/3/glossary.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork1005-2022-01-01#term-bytes-like-object). As a result, we must save it using a file object. First, we specify the file path and name

path=os.path.join(os.getcwd(),'image.png')

path

'/resources/labs/image.png'

We save the file, in order to access the body of the response we use the attribute content then save it using the open function and write method:

with open(path,'wb') as f:

f.write(r.content)

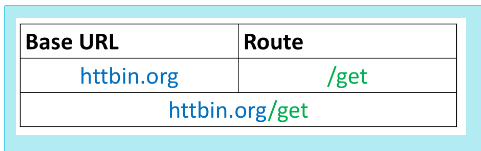
We can view the image:

Image.open(path)



## Get Request with URL Parameters

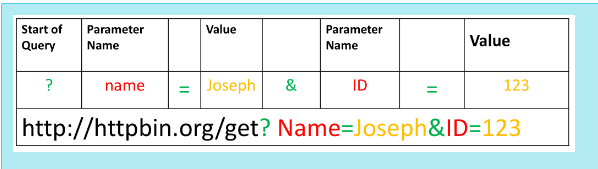
You can use the **GET** method to modify the results of your query, for example retrieving data from an API. We send a **GET** request to the server. Like before we have the **Base URL**, in the **Route** we append /get, this indicates we would like to preform a GET request. This is demonstrated in the following table:



The Base URL is for <http://httpbin.org/> is a simple HTTP Request & Response Service. The URL in Python is given by:

url\_get='http://httpbin.org/get'

A [query string](https://en.wikipedia.org/wiki/Query_string?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork1005-2022-01-01) is a part of a uniform resource locator (URL), this sends other information to the web server. The start of the query is a ?, followed by a series of parameter and value pairs, as shown in the table below. The first parameter name is name and the value is Joseph. The second parameter name is ID and the Value is 123. Each pair, parameter, and value is separated by an equals sign, =. The series of pairs is separated by the ampersand &.



To create a Query string, add a dictionary. The keys are the parameter names and the values are the value of the Query string.

payload={"name":"Joseph","ID":"123"}

Then passing the dictionary payload to the params parameter of the  get() function:

r=requests.get(url\_get,params=payload)

We can print out the URL and see the name and values

r.url

'<http://httpbin.org/get?name=Joseph&ID=123>'

There is no request body

print("request body:", r.request.body)

request body: None

We can print out the status code

print(r.status\_code)

200

We can view the response as text:

print(r.text)

{

"args": {

"ID": "123",

"name": "Joseph"

},

"headers": {

"Accept": "\*/\*",

"Accept-Encoding": "gzip, deflate, br",

"Host": "httpbin.org",

"User-Agent": "python-requests/2.29.0",

"X-Amzn-Trace-Id": "Root=1-645cfa2e-5dd76bf76b1e32ba54e4fb37"

},

"origin": "169.63.179.135",

"url": "<http://httpbin.org/get?name=Joseph&ID=123>"

}

We can look at the 'Content-Type'.

r.headers['Content-Type']

'application/json'

As the content 'Content-Type' is in the JSON format we can use the method json(), it returns a Python dict:

r.json()

{'args': {'ID': '123', 'name': 'Joseph'},

'headers': {'Accept': '\*/\*',

'Accept-Encoding': 'gzip, deflate, br',

'Host': 'httpbin.org',

'User-Agent': 'python-requests/2.29.0',

'X-Amzn-Trace-Id': 'Root=1-645cfa2e-5dd76bf76b1e32ba54e4fb37'},

'origin': '169.63.179.135',

'url': '<http://httpbin.org/get?name=Joseph&ID=123>'}

The key args has the name and values:

r.json()['args']

{'ID': '123', 'name': 'Joseph'}

## Post Requests

Like a GET request, a POST is used to send data to a server, but the POST request sends the data in a request body. In order to send the Post Request in Python, in the URL we change the route to POST:

url\_post='http://httpbin.org/post'

This endpoint will expect data as a file or as a form. A form is convenient way to configure an HTTP request to send data to a server.

To make a POST request we use the post() function, the variable payload is passed to the parameter  data :

r\_post=requests.post(url\_post,data=payload)

Comparing the URL from the response object of the GET and POST request we see the POST request has no name or value pairs.

print("POST request URL:",r\_post.url )

print("GET request [URL:",r.url](URL:%22,r.url))

POST request URL: <http://httpbin.org/post>

GET request URL: <http://httpbin.org/get?name=Joseph&ID=123>

We can compare the POST and GET request body, we see only the POST request has a body:

print("POST request body:",r\_post.request.body)

print("GET request body:",r.request.body)

POST request body: name=Joseph&ID=123

GET request body: None

We can view the form as well:

r\_post.json()['form']

There is a lot more you can do. Check out [Requests](https://requests.readthedocs.io/en/master/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=NA-SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork1005-2022-01-01)for more.