Review Of Accessing APIs

Estimated time needed: **30** minutes

Objectives

After completing this lab, you will be able to:

- Understand HTTP
- Analyze HTTP Requests

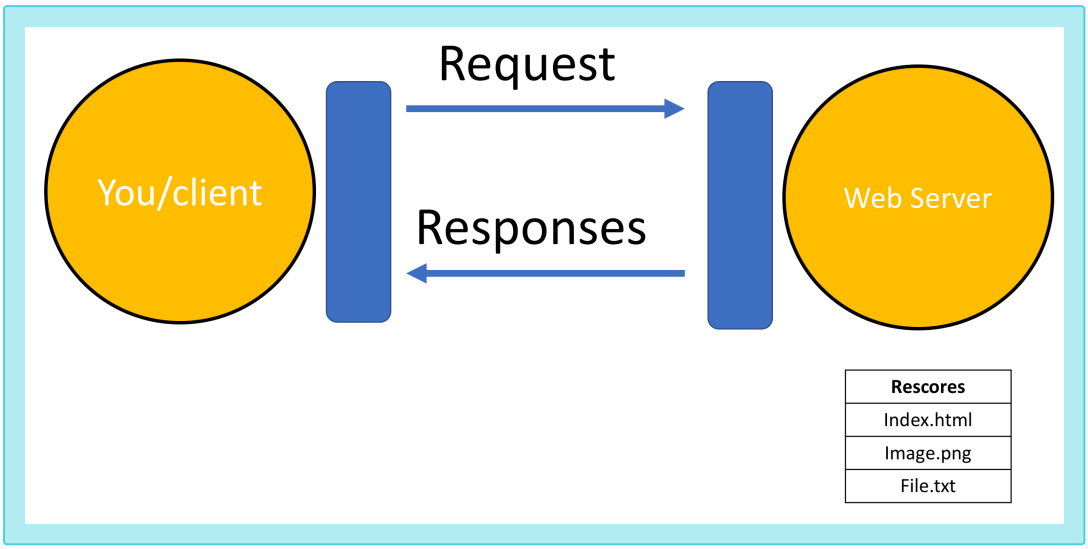
Table of Contents

Overview of HTTP

When you, the **client**, access a web page, your browser sends an **HTTP** request to the **server** where the page is hosted. The server tries to locate 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**, which 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 enables you to send and receive information through the web including webpages, images, and other web resources. In this lab, you will explore the Requests library for interacting with the `HTTP` protocol.



Uniform Resource Locator (URL)

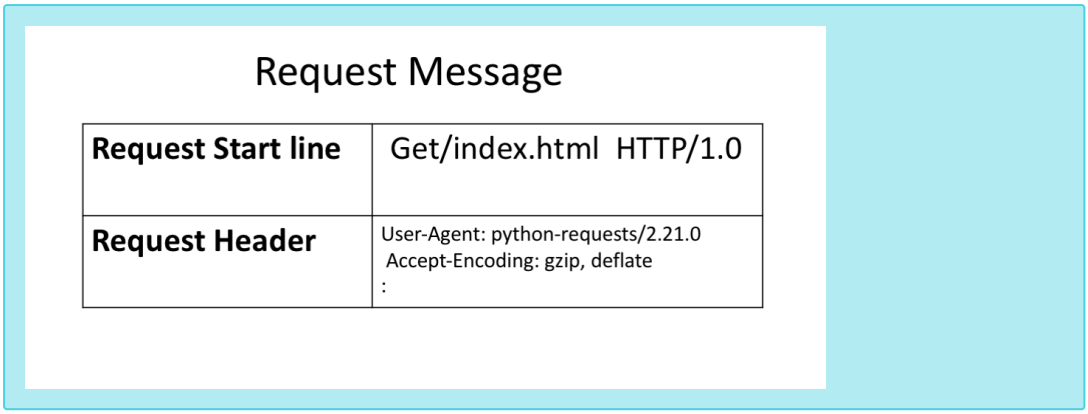
Uniform resource locator (URL) is the common way to find resources on the web. A URL can be broken down into three main parts:

- **Scheme**: The protocol used, which in this lab will always be `http://`
- **Internet address or Base URL**: Used to locate the resource. Examples include `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, which refers to 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 review more examples later.

HTTP Methods	Description
GET	Retrieves data from the server
POST	Submits data to the server
PUT	Updates data already on the server
DELETE	Deletes data from the server

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.

Response Message

Response Start line	HTTP/1.0 200 OK
Response Header	Server: Apache- Cache: UNCACHEABLE
Response Body	<!DOCTYPE html> <html> <body> My First Heading My first paragraph. </body> </html>

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 for more descriptions.

1XX	Informational
2xx	Success
200	OK
3XX	Redirection
300	Multiple Choices
4XX	Client Error
401	Unauthorized
403	Forbidden
404	Not Found

Install the required libraries

In [ ]:
```
!pip install requests
!pip install pillow
```

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:

In [ ]:
```
import requests
```

We will also use the following libraries:

In [ ]:
```
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:

In [ ]:
```
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. You can view the status code using the attribute `status_code`.

In [ ]:
```
r.status_code
```

You can view the request headers:

In [ ]:
```
print(r.request.headers)
```

You can view the request body, in the following line, as there is nobody for a get request we get `None`:

In [ ]:
```
print("request body:", r.request.body)
```

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

In [ ]:
```
header=r.headers
print(r.headers)
```

You can obtain the date the request was sent using the key `Date`.

In [ ]:
```
header['date']
```

`Content-Type` indicates the type of data:

In [ ]:
```
header['Content-Type']
```

You can also check the `encoding`:

In [ ]:
```
r.encoding
```

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

In [ ]:
```
r.text[0:100]
```

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

In [ ]:
```
# Use single quotation marks for defining string
url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeve
```

You can make a get request:

In [ ]:
```
r=requests.get(url)
```

You can look at the response header:

In [ ]:
```
print(r.headers)
```

You can see the `'Content-Type'`

In [ ]:
```
r.headers['Content-Type']
```

An image is a response object that contains the image as a bytes-like object. As a result, we must save it using a file object. First, you specify the file path and name

In [ ]:
```
path=os.path.join(os.getcwd(),'image.png')
```

You 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`:

In [ ]:
```
with open(path,'wb') as f:
    f.write(r.content)
```

You can view the image:

In [ ]:
```
Image.open(path)
```

Question: Download a file

Consider the following URL:

`URL = <https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/Example1.txt`

Write the commands to download the txt file in the given link.

In [ ]:
```
## Write your code here
```

▶ Click here for the solution

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. As 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:

Base URL	Route
httpbin.org	/get
httpbin.org/get	

The Base URL is for `http://httpbin.org/`. It is a simple HTTP Request and Response service. The `URL` in Python is given by:

In [ ]:
```
url_get='http://httpbin.org/get'
```

A query string is a part of a uniform resource locator (URL), it 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 `&`.

Start of Query	Parameter Name		Value		Parameter Name		Value
?	name	=	Joseph	&	ID	=	123
http://httpbin.org/get? Name=Joseph&ID=123							

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

In [ ]:
```
payload={"name":"Joseph","ID":"123"}
```

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

In [ ]:
```
r=requests.get(url_get,params=payload)
```

You can print out the `URL` and see the name and values.

In [ ]:
```
r.url
```

There is no request body.

In [ ]:
```
print("request body:", r.request.body)
```

You can print out the status code.

In [ ]:
```
print(r.status_code)
```

You can view the response as text:

In [ ]:
```
print(r.text)
```

You can look at the `'Content-Type'`.

In [ ]:
```
r.headers['Content-Type']
```

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

In [ ]:
```
r.json()
```

The key `args` has the name and values:

In [ ]:
```
r.json()['args']
```

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` you can change the route to `POST`:

In [ ]:
```
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`:

In [ ]:
```
try:
    response = requests.post(url_post, data=payload)
    if response.status_code == 200:
        print("Response JSON:", response.json())
    else:
```

```
        print(f"HTTP Error: {response.status_code} - {response.reason}")
except requests.exceptions.RequestException as e:
    print(f"An error occurred: {e}")
```

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

In [ ]:
```
print("POST request URL:",response.url )
print("GET request URL:",r.url)
```

You can compare the `POST` and `GET` request body, you see only the `POST` request has a body:

In [ ]:
```
print("POST request body:",response.request.body)
print("GET request body:",r.request.body)
```

You can view the form as well:

In [ ]:
```
response.json()['form']
```

There is a lot more you can do. Check out Requests for more.

Congratulations, you have completed your hands-on lab on Review Of Accessing APIs.

Authors

Joseph Santarcangelo

Other Contributors

Mavis Zhou