

PY0101EN-5.3_Requests_HTTP

November 18, 2023

HTTP and Requests

Estimated time needed: **30** minutes

0.1 Objectives

After completing this lab you will be able to:

- Understand HTTP
- Handle HTTP Requests

Table of Contents

Overview of HTTP

Uniform Resource Locator:URL

Request

Response

Requests in Python

Get Request with URL Parameters

Post Requests

0.2 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. </p>

\n<html lang="en-us">\n<head>\n \n \n \n \n \n <meta charset="UTF-8"/>\n <met'
```

```
[13]: # Use single quotation marks for defining string
url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
 ↪IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/IDSNlogo.png'
```

```
{'Date': 'Sat, 18 Nov 2023 19:48:25 GMT', 'X-Clv-Request-Id':
'42da673d-d90a-4f46-a071-c1877bf2b854', 'Server': 'Cleversafe',
```

```
'X-Clv-S3-Version': '2.5', 'Accept-Ranges': 'bytes', 'x-amz-request-id':
'42da673d-d90a-4f46-a071-c1877bf2b854', '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'

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

```
[16]: 'image/png'
```

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, we specify the file path and name

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

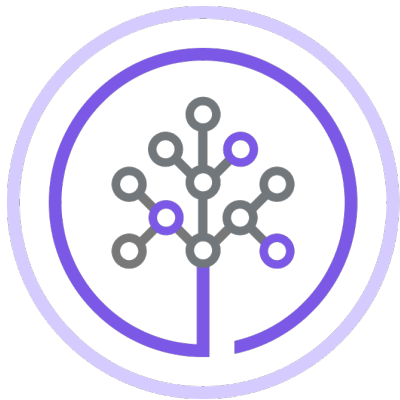
We save the file, in order to access the body of the response we use the attribute content then save it using the open function and write method:

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

We can view the image:

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

```
[19]:
```



# Skills Network

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%3C/code>

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

```
[20]: url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/
↳IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/
↳Example1.txt'
path=os.path.join(os.getcwd(), 'example1.txt')
r=requests.get(url)
with open(path, 'wb') as f:
 f.write(r.content)
```

Click here for the solution

```
url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork/labs/Module%205/data/Example1.txt'
path=os.path.join(os.getcwd(), 'example1.txt')
r=requests.get(url)
with open(path, 'wb') as f:
 f.write(r.content)
```

## 0.7 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 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
```

There is no request body.

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

We can print out the status code.

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

We can view the response as text:

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

We can look at the 'Content-Type'.

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

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

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

The key `args` has the name and values:

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

## 0.8 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)
```

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)
```

We can view the form as well:

```
[]: r_post.json()['form']
```

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

## 0.9 Authors

Joseph Santarcangelo A Data Scientist at IBM, and holds a PhD in Electrical Engineering. His research focused on using Machine Learning, Signal Processing, and Computer Vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

### 0.9.1 Other Contributors

Mavis Zhou

## 0.10 Change Log

| Date (YYYY-MM-DD) | Version | Changed By       | Change Description           |
|-------------------|---------|------------------|------------------------------|
| 2023-11-02        | 2.4     | Abhishek Gagneja | Updated instructions         |
| 2023-06-07        | 2.3     | Akansha Yadav    | Spell Check                  |
| 2021-12-20        | 2.1     | Malika           | Updated the links            |
| 2020-09-02        | 2.0     | Simran           | Template updates to the file |

##

© IBM Corporation 2023. All rights reserved.

```
[]:
```