Requests_HTTP

September 14, 2021

HTTP and Requests

Estimated time needed: **15** minutes

0.1 Objectives

After completing this lab you will be able to:

- Understand HTTP
- Handle HTTP Requests

Table of Contents

```
<ul>
    <li>
        <a href="#index">Overview of HTTP </a>
        <ul>
            <li><a href="#HTTP">Uniform Resource Locator:URL</a></li>
             <li><a href="slice">Request</a></li>
            <li><a href="stride">Response</a></li>
        </ul>
    </li>
    <li>
        <a href="#RP">Requests in Python  </a>
        <ul>
            <li><a href="#get">Get Request with URL Parameters</a></li>
            <li><a href="#post">Post Requests </a></li>
```

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

    <img src="https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperS

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:

    <img src="https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperS

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.

    <img src="https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperS

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.

    <img src="https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperS

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.

    <img src="https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperS

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:

[1]:
```python
import requests
```

We will also use the following libraries:

```
[2]: 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:

```
[3]: 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.

```
[4]: r.status_code
```

```
[4]: 200
```

You can view the request headers:

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

```
{'User-Agent': 'python-requests/2.25.1', 'Accept-Encoding': 'gzip, deflate',
'Accept': '*/*', 'Connection': 'keep-alive', 'Cookie': 'bm_sz=AE80EDD1EAD34977F5
311DC9572C56E4~YAAQo50ZuKF5z1J6AQAArhw4Vgy3dcUcEuRRSn/MQve14istIJ4ous7PJP/GZ0GB0
gWZdrghfJzLwj9j+HkCvuHx6duuS5uFc/AviuJcNpXV7xv1Hz8NKoG1uZ6djaWUzEAFB5j79eU6W4qtG
3K87dvyVOSjPEygpX8IuS58dx7XaV/390nlvgjzvwj6; _abck=0141514AE8E329240031F54BCCEB8
1C9~-1~YAAQo50ZuKJ5z1J6AQAArhw4VgafgYlPbqqub5ovuPrIjRjlRS5Z8oijBIEWMXnfDiSATCEsN
KLUlmxTkU+S2X+/0AuhfAS+NOHOGXANdAXqF5UxX97+5TRSyGOj1xpetQV1G6Lmq5IqV/cqQreQ1Jr0e
+yZ4DfjiwTRDPTanntqhOio2MXMyzFpm2MJbHncKKihXFHFhdkm6NutBGGcPdHXOmmihEwoDyeD1lB/p
W0bjaz7Nkd24uCYZx4IXGOEI06bmBEUB76Rv6CL+5wKE4xQXKZUzBSSitoSuctc5eP5DKBWs9s4Fw28l
uyixhmWmRoVtecoTjzBHxpIUr7bm2Nw4R0lkJmxZxSVIAOQ4nmPqZbWD1M=~-1~-1~-1'}
```

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

```
[6]: 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.

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

```
{'Cache-Control': 'max-age=301', 'Expires': 'Mon, 28 Jun 2021 12:52:39 GMT',
'Last-Modified': 'Fri, 25 Jun 2021 18:05:17 GMT', 'ETag':
'"17dd2-5c59afb733963"', 'Accept-Ranges': 'bytes', 'Content-Encoding': 'gzip',
'Content-Type': 'text/html', 'X-Akamai-Transformed': '9 18051 0 pmb=mTOE,1',
'Date': 'Tue, 29 Jun 2021 05:22:35 GMT', 'Content-Length': '18129',
'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

```
[9]: header['date']
```

```
[9]: 'Tue, 29 Jun 2021 05:22:35 GMT'
```

Content-Type indicates the type of data:

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

```
[10]: 'text/html'
```

You can also check the encoding:

```
[11]:  r.encoding
```

```
[11]: '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:

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

```
[12]: '<!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:

```
[13]: # Use single quotation marks for defining string
      url='https://gitlab.com/ibm/skills-network/courses/placeholder101/-/raw/master/
       ↪labs/module%201/images/IDSNlogo.png'
```

We can make a get request:

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

We can look at the response header:

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

```
{'Date': 'Tue, 29 Jun 2021 05:24:33 GMT', 'Content-Type': 'image/png', 'Content-
Length': '21590', 'Connection': 'keep-alive', 'Cache-Control': 'max-age=60,
public', 'Content-Disposition': 'inline', 'Etag':
'W/"c26d88d0ca290ba368620273781ea37c"', 'Permissions-Policy': 'interest-
cohort=()', 'Vary': 'Accept, Accept-Encoding', 'X-Content-Type-Options':
'nosniff', 'X-Download-Options': 'noopen', 'X-Frame-Options': 'DENY',
'X-Permitted-Cross-Domain-Policies': 'none', 'X-Request-Id':
```

```
'01F9B3KTCESGFF17A9BYXZC82A', 'X-Runtime': '0.052865', 'X-Ua-Compatible':
'IE=edge', 'X-Xss-Protection': '1; mode=block', 'Strict-Transport-Security':
'max-age=31536000', 'Referrer-Policy': 'strict-origin-when-cross-origin',
'GitLab-LB': 'fe-12-lb-gprd', 'GitLab-SV': 'web-30-sv-gprd', 'CF-Cache-Status':
'EXPIRED', 'Accept-Ranges': 'bytes', 'cf-request-id':
'0af7d3015e00005ed26b9ea000000001', 'Expect-CT': 'max-age=604800, report-
uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"', 'Server':
'cloudflare', 'CF-RAY': '666cbaaefc6d5ed2-IAD'}
```

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')
path
```

[17]:
```
'/resources/labs/PY0101EN/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]:

IBM Developer

SKILLS NETWORK

Question 1: write wget

In the previous section, we used the wget function to retrieve content from the web server as shown below. Write the python code to perform the same task. The code should be the same as the one used to download the image, but the file name should be `Example1.txt'.

```
!wget -O /resources/data/Example1.txt https://cf-courses-data.s3.us.cloud-object-storage.appdom
```

```
[ ]:
```

Click here for the solution

```
url='https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSkillsNetw
path=os.path.join(os.getcwd(),'example1.txt')
```

```
r=requests.get(url)
with open(path,'wb') as f:
    f.write(r.content)
```

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:

    <img src="https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperS

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

[20]:
```
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 &.

    <img src="https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperS

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

[21]:
```
payload={"name":"Joseph","ID":"123"}
```

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

[22]:
```
r=requests.get(url_get,params=payload)
```

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

[23]:
```
r.url
```

[23]:
```
'http://httpbin.org/get?name=Joseph&ID=123'
```

There is no request body

[24]:
```
print("request body:", r.request.body)
```

```
request body: None
```

We can print out the status code

[25]:
```
print(r.status_code)
```

```
200
```

We can view the response as text:

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

```
{
  "args": {
    "ID": "123",
    "name": "Joseph"
  },
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.25.1",
    "X-Amzn-Trace-Id": "Root=1-60daaecc-69f33e3c5304866d00d1a26b"
  },
  "origin": "52.116.125.104",
  "url": "http://httpbin.org/get?name=Joseph&ID=123"
}
```

We can look at the `Content-Type'.

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

```
[27]: 'application/json'
```

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

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

```
[28]: {'args': {'ID': '123', 'name': 'Joseph'},
       'headers': {'Accept': '*/*',
        'Accept-Encoding': 'gzip, deflate',
        'Host': 'httpbin.org',
        'User-Agent': 'python-requests/2.25.1',
        'X-Amzn-Trace-Id': 'Root=1-60daaecc-69f33e3c5304866d00d1a26b'},
       'origin': '52.116.125.104',
       'url': 'http://httpbin.org/get?name=Joseph&ID=123'}
```

The key args has the name and values:

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

```
[29]: {'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:

```
[30]: 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 :

```
[31]: 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.

```
[32]: print("POST request URL:",r_post.url )
      print("GET request URL:",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)
```

We can view the form as well:

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

```
[33]: {'ID': '123', 'name': 'Joseph'}
```

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

0.2 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.

0.2.1 Other Contributors

Mavis Zhou

0.3 Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2021-12-20	2.1	Malika	Updated the links
2020-09-02	2.0	Simran	Template updates to the file

##