(https://cognitiveclass.ai/)

HTTP and Requests

Estimated time needed: **15** minutes

Objectives

After completing this lab you will be able to:

- Understand HTTP
- Handle the HTTP Requests

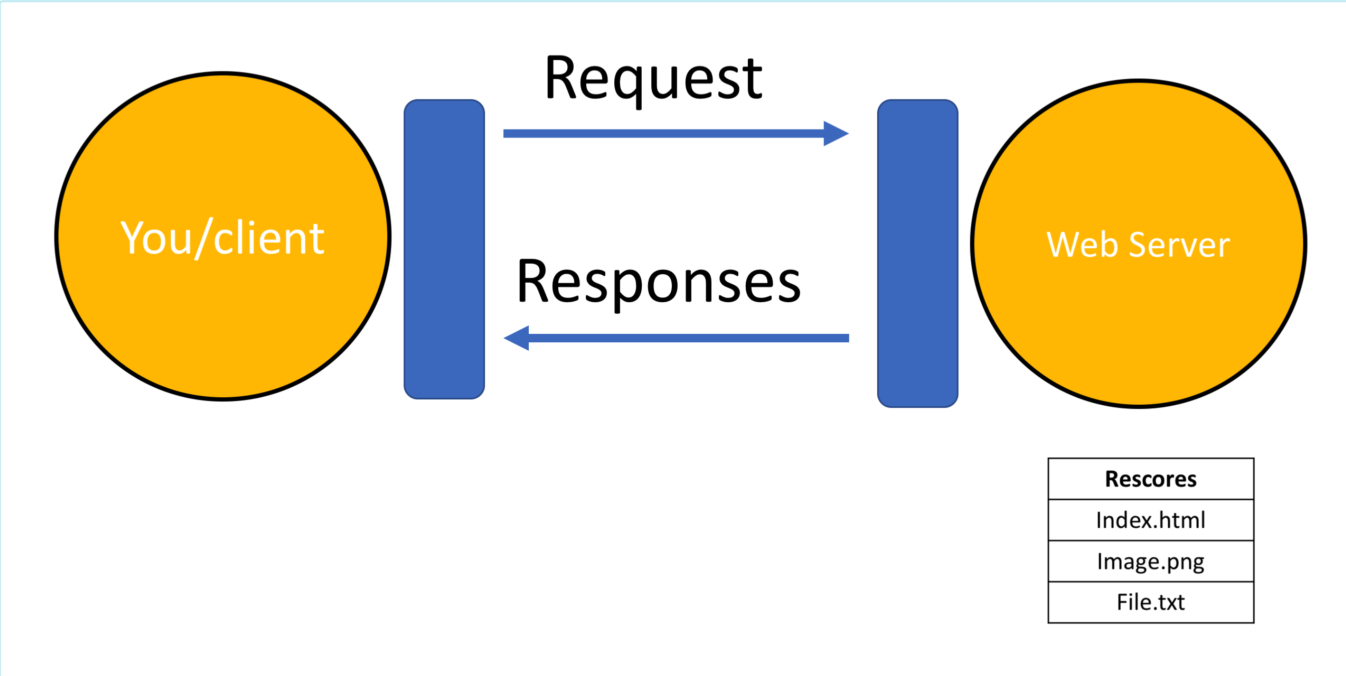
Table of Contents

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



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 here 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:

Request Message

Request Start line	Get/index.html HTTP/1.0
Request Header	User-Agent: python-requests/2.21.0 Accept-Encoding: gzip, deflate :

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.

HTTP METHODS	**Description**
GET	Retrieves Data from the server
POST	Submits data to server
PUT	Updates data already on server
DELETE	Deletes data from 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 request 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 (https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status) for more descriptions.

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

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 [1]:

```
import requests
```

We will also use the following libraries

In [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 (http://www.ibm.com?cm_mmc=Email_Newsletter-_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork-19487395&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newslette_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork-19487395&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newslette_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork-19487395&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newslette_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork-19487395&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newslette_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork-19487395&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newslette_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork-19487395&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newslette_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork-19487395&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newslette_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork-19487395&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newslette_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork-19487395&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newslette](http://www.ibm.com)

In [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`

In [4]:

```
r.status_code
```

Out[4]:

```
200
```

You can view the request headers:

In [5]:

```
print(r.request.headers)
```

```
{'User-Agent': 'python-requests/2.25.0', 'Accept-Encoding': 'gzip, deflat
e', 'Accept': '*/*', 'Connection': 'keep-alive', 'Cookie': '_abck=0CAD0B4F
052A53584F105FAFDE050C44~-1~YAAQnPNUuCSu23d3AQAAOAaVigUsjzAlY36yGk2PVAVpZu
ELpEeU+w9+ICHbDtNLVWVWC0KxHgkjASmADI9o/BgawF8/XyU1JqeFAY8LreVAea9f5trFTCrx
ETpOiIExKOnZ0YDTfVQ7I6QilCY3BPgMqmHvZ9jiuAC5lRQ+K+T+70Q5jd0rESMSP9govmXTJ2
BuVAEmwgNEv+N9Dqg+hPRuwBZLRKkmaJo1wuMcGdomr8IKaQtVDBoLuVH4NuyJgej5f9sQ+SbP
GAozvV/y76eObYestH53/BinB5KOhV4WjHO+XeTK~-1~-1~-1; bm_sz=3FD36EF8EE9AAD16B
9907F472EE87FFC~YAAQnPNUuCOu23d3AQAAOAaVigquUIkJkc9WoJuh9ErEBftpWkSw1wxRfu
3yifXP77nbHIYgAwEGr8cEgZ17811U/ueUkfYwIlZ6KhN4IMhWoW8KSX2TQ/nxoamduUwWJ3qq
DZvkUhjAwftHRMFQga41XgkRk3IyRyZQYd021dOu7fRpNieS3sSRSO5U'}
```

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

In [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.

In [7]:

```
header=r.headers
print(r.headers)
```

```
{'Server': 'Apache', 'x-drupal-dynamic-cache': 'UNCACHEABLE', 'Link': '<ht
tps://www.ibm.com/ca-en>; rel="canonical", <https://www.ibm.com/ca-en>; re
l="revision", <https://www.ibm.com/ca-en>; rel="revision", <//1.cms.s81c.c
om>; rel=preconnect; crossorigin, <//1.cms.s81c.com>; rel=dns-prefetch',
'x-ua-compatible': 'IE=edge', 'Content-Language': 'en-ca', 'x-generator':
'Drupal 8 (https://www.drupal.org)', 'x-dns-prefetch-control': 'on', 'x-dr
upal-cache': 'MISS', 'Last-Modified': 'Wed, 10 Feb 2021 02:56:41 GMT', 'ET
ag': '"1612925801"', 'Content-Type': 'text/html; charset=UTF-8', 'x-acquia
-host': 'www.ibm.com', 'x-acquia-path': '/ca-en', 'x-acquia-site': '', 'x-
acquia-purge-tags': '', 'x-varnish': '326699619 331318984', 'x-cache-hit
s': '10', 'x-age': '8893', 'Accept-Ranges': 'bytes', 'Content-Encoding':
'gzip', 'Cache-Control': 'public, max-age=300', 'Expires': 'Wed, 10 Feb 20
21 06:20:58 GMT', 'X-Akamai-Transformed': '9 12050 0 pmb=mTOE,1', 'Date':
'Wed, 10 Feb 2021 06:15:58 GMT', 'Content-Length': '12164', 'Connection':
'keep-alive', 'Vary': 'Accept-Encoding', 'x-content-type-options': 'nosnif
f', 'X-XSS-Protection': '1; mode=block', 'Content-Security-Policy': 'upgra
de-insecure-requests', 'Strict-Transport-Security': 'max-age=31536000', 'x
-ibm-trace': 'www-dipatcher: dynamic rule'}
```

We can obtain the date the request was sent using the key `Data`

In [8]:

```
header['date']
```

Out[8]:

```
'Wed, 10 Feb 2021 06:15:58 GMT'
```

`Content-Type` indicates the type of data:

In [9]:

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

Out[9]:

```
'text/html; charset=UTF-8'
```

You can also check the `encoding` :

In [10]:

```
r.encoding
```

Out[10]:

```
'UTF-8'
```

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:

In [11]:

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

Out[11]:

```
'<!DOCTYPE html>\n<html lang="en-ca" dir="ltr">\n  <head>\n    <meta chars
et="utf-8" />\n<script>digitalD'
```

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

In [12]:

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

We can make a get request:

In [13]:

```
r=requests.get(url)
```

We can look at the response header:

In [14]:

```
print(r.headers)
```

```
{'Date': 'Wed, 10 Feb 2021 06:19:24 GMT', 'Content-Type': 'image/png', 'Co
ntent-Length': '21590', 'Connection': 'keep-alive', 'Set-Cookie': '__cfdui
d=ddc0a594691a1130d80f1d9ae71d58aff1612937964; expires=Fri, 12-Mar-21 06:1
9:24 GMT; path=/; domain=.gitlab.com; HttpOnly; SameSite=Lax; Secure', 'Ca
che-Control': 'max-age=60, public', 'Content-Disposition': 'inline', 'Eta
g': 'W/"c26d88d0ca290ba368620273781ea37c"', 'X-Content-Type-Options': 'nos
niff', 'X-Download-Options': 'noopen', 'X-Frame-Options': 'DENY', 'X-Gitla
b-Feature-Category': 'source_code_management', 'X-Permitted-Cross-Domain-P
olicies': 'none', 'X-Request-Id': '01EY3S8QJK3R8RWP0CXPDK5EQS', 'X-Runtim
e': '0.096094', 'X-Ua-Compatible': 'IE=edge', 'X-Xss-Protection': '1; mode
=block', 'Strict-Transport-Security': 'max-age=31536000', 'Referrer-Polic
y': 'strict-origin-when-cross-origin', 'GitLab-LB': 'fe-08-lb-gprd', 'GitL
ab-SV': 'web-29-sv-gprd', 'CF-Cache-Status': 'HIT', 'Age': '47', 'Accept-R
anges': 'bytes', 'cf-request-id': '082c3141f500004003c785c000000001', 'Exp
ect-CT': 'max-age=604800, report-uri="https://report-uri.cloudflare.com/cd
n-cgi/beacon/expect-ct"', 'Vary': 'Accept-Encoding', 'Server': 'cloudflar
e', 'CF-RAY': '61f3b7e329d44003-YYZ'}
```

We can we can see the `'Content-Type'`

In [15]:

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

Out[15]:

```
'image/png'
```

An image is a response object that contains the image as a [bytes-like object (https://docs.python.org/3/glossary.html#term-bytes-like-object)](https://docs.python.org/3/glossary.html#term-bytes-like-object). As a result, we must save it using a file object. First, we specify the file path and name

In [16]:

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

Out[16]:

```
'/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` :

In [17]:

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

We can view the image:

In [18]:

```
Image.open(path)
```

Out[18]:

Question 1: write [`wget` (https://www.gnu.org/software/wget/)](https://www.gnu.org/software/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 `'example.txt'`.

```
!wget -O /resources/data/Example1.txt
```

In [19]:

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


```
{'Date': 'Wed, 10 Feb 2021 06:26:10 GMT', 'X-Clv-Request-Id': '85ae53dd-0913-4082-b486-32b96a49a4bc', 'Server': 'Cleversafe', 'X-Clv-S3-Version': '2.5', 'Accept-Ranges': 'bytes', 'x-amz-request-id': '85ae53dd-0913-4082-b486-32b96a49a4bc', 'ETag': '"522bc63a850f3569b8924c51131d5fe4"', 'Content-Type': 'text/plain', 'Last-Modified': 'Tue, 09 Feb 2021 14:48:30 GMT', 'Content-Length': '45'}
```


▶ 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. 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:

Base URL	Route
httbin.org	/get
httbin.org/get	

The Base URL is for `[http://httpbin.org/](http://httpbin.org?cm_mmc=Email_Newsletter-_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork-19487395&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newsletter.M_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork-19487395&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newsletter.M_-Developer_Ed%2BTech-_-WW_WW-_-SkillsNetwork-Courses-IBMDeveloperSkillsNetwork-PY0101EN-SkillsNetwork-19487395&cm_mmca1=000026UJ&cm_mmca2=10006555&cm_mmca3=M12345678&cvosrc=email.Newsletter.M` is a simple HTTP Request & Response Service. The `URL` in Python is given by:

In [20]:

```
url_get='http://httpbin.org/get'
```

A [query string (https://en.wikipedia.org/wiki/Query_string)](https://en.wikipedia.org/wiki/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 `&` .

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 [21]:

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

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

In [22]:

```
r=requests.get(url_get,params=payload)
```

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

In [23]:

```
r.url
```

Out[23]:

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

There is no request body

In [24]:

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

```
request body: None
```

We can print out the status code

In [25]:

```
print(r.status_code)
```

```
200
```

We can view the response as text:

In [26]:

```
print(r.text)
```


```
{
  "args": {
    "ID": "123",
    "name": "Joseph"
  },
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.25.0",
    "X-Amzn-Trace-Id": "Root=1-60237d2b-46e5f4d10f31922856e13ca3"
  },
  "origin": "169.53.161.204",
  "url": "http://httpbin.org/get?name=Joseph&ID=123"
}
```


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

In [27]:

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

Out[27]:

```
'application/json'
```

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

In [28]:

```
r.json()
```

Out[28]:


```
{'args': {'ID': '123', 'name': 'Joseph'},
 'headers': {'Accept': '*/*',
  'Accept-Encoding': 'gzip, deflate',
  'Host': 'httpbin.org',
  'User-Agent': 'python-requests/2.25.0',
  'X-Amzn-Trace-Id': 'Root=1-60237d2b-46e5f4d10f31922856e13ca3'},
 'origin': '169.53.161.204',
 'url': 'http://httpbin.org/get?name=Joseph&ID=123'}
```


The key `args` had the name and values:

In [29]:

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

Out[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` :

In [30]:

```
url_post='http://httpbin.org/post'
```

This endpont will expect data as a file or as a form, a from 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 [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.

In [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:

In [33]:

```
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:

In [34]:

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

Out[34]:

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

There is a lot more you can do check out Requests (https://requests.readthedocs.io/en/master/) for more.

Authors

Joseph Santarcangelo (https://www.linkedin.com/in/joseph-s-50398b136/)
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.

Other Contributors

Mavis Zhou (www.linkedin.com/in/jiahui-mavis-zhou-a4537814a)

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