HTTP Request/Response Cycle - Codealong ¶

Introduction

When developing a Web application, as we saw in the previous lesson, the request/response cycle is a useful guide to see how all the components of the app fit together. The request/response cycle traces how a user's request flows through the app. Understanding the request/response cycle is helpful to figure out which files to edit when developing an app (and where to look when things aren't working). This lesson will show how this setup works using python.

Objectives

You will be able to:

- Explain the HTTP request/response cycle
- List the status codes of responses and their meanings
- Obtain and interpret status codes from responses
- Make HTTP GET and POST requests in python using the `requests` library

The `requests` Library in Python

Dealing with HTTP requests could be a challenging task any programming language. Python with two built-in modules, `urllib` and `urllib2` to handle these requests but these could be very confusing and the documentation is not clear. This requires the programmer to write a lot of code to make even a simple HTTP request.

To make these things simpler, one easy-to-use third-party library, known as `Requests` , is available and most developers prefer to use it instead or urllib/urllib2. It is an Apache2 licensed HTTP library powered by urllib3 and httplib. Requests is add-on library that allows you to send HTTP requests using Python. With this library, you can access content like web page headers, form data, files, and parameters via simple Python commands. It also allows you to access the response data in a simple way.

Below is how you would install and import the requests library before making any requests.

```
# Uncomment and install requests if you don't have it already
# !pip install requests

# Import requests to working environment
import requests
```

In [2]:
```
# Code here
import requests
```

The `.get()` Method

Now we have requests library ready in our working environment, we can start making some requests using the `.get()` method as shown below:

```
### Making a request
resp = requests.get('https://www.google.com')
```

In [3]:
```
# Code here
resp = requests.get('https://www.google.com')
```

GET is by far the most used HTTP method. We can use GET request to retrieve data from any destination.

Status Codes

The request we make may not be always successful. The best way is to check the status code which gets returned with the response. Here is how you would do this.

```
# Check the returned status code
resp.status_code == requests.codes.ok
```

In [4]:
```
# Code here
resp.status_code == requests.codes.ok
```

Out[4]:
```
True
```

So this is a good check to see if our request was successful. Depending on the status of the web server, the access rights of the clients and the availability of requested information. A web server may return a number of status codes within the response. Wikipedia has an exhaustive details on all these codes. Check them out here (https://en.wikipedia.org/wiki/List_of_HTTP_status_codes).

Response Contents

Once we know that our request was successful and we have a valid response, we can check the returned information using `.text` property of the response object.

```
print (resp.text)
```

In [5]:
```
# Code here
print (resp.text)
```

```
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang
="en"><head><meta content="Search the world's information, including webpage
s, images, videos and more. Google has many special features to help you find
exactly what you're looking for." name="description"><meta content="noodp" na
me="robots"><meta content="text/html; charset=UTF-8" http-equiv="Content-Typ
e"><meta content="/images/branding/googleg/1x/googleg_standard_color_128dp.pn
g" itemprop="image"><title>Google</title><script nonce="tuo2_B5HJROoHQiU8L4HJ
g">(function(){window.google={kEI:'1hP-YqCMDuaiiLMP6ICYkAI',kEXPI:'0,1302536,
56873,6058,207,4804,2316,383,246,5,5367,1123753,1197749,380742,16111,17447,11
240,17572,4859,1361,9290,3023,17586,4998,13228,3847,10622,22741,5081,1593,127
9,2742,149,1103,840,1983,4315,107,3406,606,2023,1777,520,14670,3227,2845,9,29
072,4696,1851,15324,432,3,1590,1,5445,803,10668,2652,4,1528,2304,7039,20309,1
714,3050,2658,4163,3194,9814,3844,2980,16808,1435,5770,2587,4094,17,4035,3,35
41,1,11942,27100,1,3111,2,8984,1,5037,2373,342,3533,7868,11623,5679,1021,238
0,28742,4568,6258,23418,1246,5841,14968,4332,2204,2083,1803,1394,445,2,2,1,10
957,6349,9326,8155,6582,100,699,2,3,1396,78,13201,2162,5179,4447,10008,7,192
2,9803,6518,346,10751,3001,1139,1108,834,701,6899,1749,1624,1460,5762,192,198
0,273,3393,3880,552,984,122,700,4,1,2,2,2,2,1645,5439,7,1,761,548,2965,147,59
9,563,1195,46,85,614,509,4650,678,14,2,3789,48,1580,267,653,277,1707,14,82,32
```

So this returns a lot of information which by default is not really human-understandable due to data encoding, HTML tags and other styling information that only a web browser can truly translate. In later lessons, we'll learn how we can use ***Regular Expressions*** to clean this information and extract the required bits and pieces for analysis.

Response Headers

The response of an HTTP request can contain many headers that holds different bits of information. We can use `.header` property of the response object to access the header information as shown below:

```
# Read the header of the response - convert to dictionary for displaying
k:v pairs neatly
dict(resp.headers)
```

In [6]:
```
# Code here
dict(resp.headers)
```

Out[6]:
```
{'Date': 'Thu, 18 Aug 2022 10:26:30 GMT',
 'Expires': '-1',
 'Cache-Control': 'private, max-age=0',
 'Content-Type': 'text/html; charset=ISO-8859-1',
 'P3P': 'CP="This is not a P3P policy! See g.co/p3phelp for more info."',
 'Content-Encoding': 'gzip',
 'Server': 'gws',
 'X-XSS-Protection': '0',
 'X-Frame-Options': 'SAMEORIGIN',
 'Set-Cookie': '1P_JAR=2022-08-18-10; expires=Sat, 17-Sep-2022 10:26:30 GMT; pa
th=/; domain=.google.com; Secure, AEC=AakniGMKFb5stQZCuR-nB3z7kLdAO2H7l2VNYZK5Y
NO8HkRYNLrhX8IXcg; expires=Tue, 14-Feb-2023 10:26:30 GMT; path=/; domain=.googl
e.com; Secure; HttpOnly; SameSite=lax, NID=511=HRwMStFohDl8OZ7tAzP49qEjZaPv8ixA
j2l_BImcccbWmRhNI9-ACHbfs56CEcPxCEdWgxclTn-HD_-C4gWMp6yywSZLJUKvpIRtqNa4aI7dIEm
sA9lrH-P_JNvBJAEBOkPVP8tkTz5WYwSST5FrBK_cLa6iBwENx6eMwzirzXU; expires=Fri, 17-F
eb-2023 10:26:30 GMT; path=/; domain=.google.com; HttpOnly',
 'Alt-Svc': 'h3=":443"; ma=2592000,h3-29=":443"; ma=2592000,h3-Q050=":443"; ma=
2592000,h3-Q046=":443"; ma=2592000,h3-Q043=":443"; ma=2592000,quic=":443"; ma=2
592000; v="46,43"',
 'Transfer-Encoding': 'chunked'}
```

The content of the headers is our required element. You can see the key-value pairs holding various pieces of information about the resource and request. Let's try to parse some of these values using the requests library:

```
print(resp.headers['Date'])  # Date the response was sent
print(resp.headers['server'])   # Server type (google web service - GWS)
```

In [7]:
```
# Code here
print(resp.headers['Date'])
print(resp.headers['server'])
```

```
Thu, 18 Aug 2022 10:26:30 GMT
gws
```

Try `httpbin`

`httpbin.org` is a popular website to test different HTTP operations and practice with request-response cycles. Let's use httpbin/get to analyze the response to a GET request. First of all, let's find out the response header and inspect how it looks.

```
r = requests.get('http://httpbin.org/get')

response = r.json()
print(r.json())
print(response['args'])
print(response['headers'])
print(response['headers']['Accept'])
print(response['headers']['Accept-Encoding'])
print(response['headers']['Host'])
print(response['headers']['User-Agent'])
print(response['origin'])
print(response['url'])
```

In [8]:
```
r = requests.get('http://httpbin.org/get')

response = r.json()
print(r.json())
print(response['args'])
print(response['headers'])
print(response['headers']['Accept'])
print(response['headers']['Accept-Encoding'])
print(response['headers']['Host'])
print(response['headers']['User-Agent'])
print(response['origin'])
print(response['url'])
```

```
{'args': {}, 'headers': {'Accept': '*/*', 'Accept-Encoding': 'gzip, deflate, b
r', 'Host': 'httpbin.org', 'User-Agent': 'python-requests/2.26.0', 'X-Amzn-Trac
e-Id': 'Root=1-62fe13e0-20c65dc130eac19d02209514'}, 'origin': '3.236.232.217',
'url': 'http://httpbin.org/get'}
{}
{'Accept': '*/*', 'Accept-Encoding': 'gzip, deflate, br', 'Host': 'httpbin.or
g', 'User-Agent': 'python-requests/2.26.0', 'X-Amzn-Trace-Id': 'Root=1-62fe13e0
-20c65dc130eac19d02209514'}
*/*
gzip, deflate, br
httpbin.org
python-requests/2.26.0
3.236.232.217
http://httpbin.org/get (http://httpbin.org/get)
```

In [13]:
```
# Code here
r = requests.get('http://httpbin.org/get')
response = r.json()
print(r.json())
print(response['args'])
print(response['headers'])
print(response['headers']['Accept'])
print(response['headers']['Accept-Encoding'])
print(response['headers']['Host'])
print(response['headers']['User-Agent'])
print(response['origin'])
print(response['url'])
```

```
{'args': {}, 'headers': {'Accept': '*/*', 'Accept-Encoding': 'gzip, deflate, b
r', 'Host': 'httpbin.org', 'User-Agent': 'python-requests/2.26.0', 'X-Amzn-Trac
e-Id': 'Root=1-62fdb59a-4155f0167d9fa0a34ae27119'}, 'origin': '3.236.232.217',
'url': 'http://httpbin.org/get'}
{}
{'Accept': '*/*', 'Accept-Encoding': 'gzip, deflate, br', 'Host': 'httpbin.or
g', 'User-Agent': 'python-requests/2.26.0', 'X-Amzn-Trace-Id': 'Root=1-62fdb59a
-4155f0167d9fa0a34ae27119'}
*/*
gzip, deflate, br
httpbin.org
python-requests/2.26.0
3.236.232.217
http://httpbin.org/get (http://httpbin.org/get)
```

Let's use `requests` object structure to parse the values of headers as we did above.

```
print(r.headers['Access-Control-Allow-Credentials'])
print(r.headers['Access-Control-Allow-Origin'])
print(r.headers['CONNECTION'])
print(r.headers['content-length'])
print(r.headers['Content-Type'])
print(r.headers['Date'])
print(r.headers['server'])
```

In [14]:
```
# Code here
print(r.headers['Access-Control-Allow-Credentials'])
print(r.headers['Access-Control-Allow-Origin'])
print(r.headers['CONNECTION'])
print(r.headers['content-length'])
print(r.headers['Content-Type'])
print(r.headers['Date'])
print(r.headers['server'])
```

```
true
*
keep-alive
310
application/json
Thu, 18 Aug 2022 03:44:26 GMT
gunicorn/19.9.0
```

Passing Parameters in GET

In some cases, you'll need to pass parameters along with your GET requests. These extra parameters usually take the the form of query strings added to the requested URL. To do this, we need to pass these values in the `params` parameter. Let's try to access information from `httpbin` with some user information.

Note: The user information is not getting authenticated at `httpbin` so any name/password will work fine. This is merely for practice.

```
credentials = {'user_name': 'FlatironSchool', 'password': 'learnlovecod
e'}
r = requests.get('http://httpbin.org/get', params=credentials)

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

In [18]:
```
# Code here
credentials = {'user_name': 'FlatironSchool', 'password': 'learnlovecode'}
r = requests.get('http://httpbin.org/get', params=credentials)

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

```
http://httpbin.org/get?user_name=FlatironSchool&password=learnlovecode (http://
httpbin.org/get?user_name=FlatironSchool&password=learnlovecode)
{
  "args": {
    "password": "learnlovecode",
    "user_name": "FlatironSchool"
  },
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate, br",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.26.0",
    "X-Amzn-Trace-Id": "Root=1-62fdb7be-491c659f3815ffdd39b81d8f"
  },
  "origin": "3.236.232.217",
  "url": "http://httpbin.org/get?user_name=FlatironSchool&password=learnlovecod
e"
}
```

HTTP POST method

Sometimes we need to send one or more files simultaneously to the server. For example, if a user is submitting a form and the form includes different fields for uploading files, like user profile picture, user resume, etc. Requests can handle multiple files on a single request. This can be achieved by putting the files to a list of tuples in the form (`field_name, file_info`) .

```
import requests

url = 'http://httpbin.org/post'
file_list = [
    ('image', ('fi.png', open('images/fi.png', 'rb'), 'image/png')),
    ('image', ('fi2.jpeg', open('images/fi2.jpeg', 'rb'), 'image/png'))
]

r = requests.post(url, files=file_list)
print(r.text)
```

In [20]:
```
# Code here
import requests

url = 'http://httpbin.org/post'
file_list = [
    ('image', ('fi.png', open('images/fi.png', 'rb'), 'image/png')),
    ('image', ('fi2.jpeg', open('images/fi2.jpeg', 'rb'), 'image/png'))
]

r = requests.post(url, files=file_list)
print(r.text)
```


```
{
  "args": {},
  "data": "",
  "files": {
    "image": "
86UAAADAFBMVEX///9VVVXF7Pay5vPg4ODv7+/i4uLe3t7KysqqqqqV3e5FREQ8wN/i9vpJSUlRUV
HGxsaurq61tLTT09OdnJyioqKenp51dHS5ubmJiIjBwcGtrKyVlJS9vb1hYGC6urqmpqahoaGXlpa
BgIBu0OixsLBdXFxVyOR5eHiDgoK2trby+/2Ojo6ampp3dnaNjIxNTExxcXGLioqF1+ud3+/y8vLM
zMy/v7+lpaVycnKZmJhlZWXZ2NiMi4uysrJ/fn7+/v45ODg1NDQ3Njb8/Pz5+fnx8fH7+/vt7e36+
vo+PT09PDz39/c0MzPn5+fV1dVDQkJPTk44Nzfq6upfXl7R0NDr6+s7OjpkY2M2NTU6OTnNzMzFxM
SUk5Ps7Ozc3NxcW1twb2/4+Pj29vZ+fn5OTU3Z2dlsa2s8OzuAf394d3eQj4/w8PBCQUHm5ubFxcW
op6egn59UU1O8u7u+vr5oZ2elpKRlZGTd3d2RkJDd3Nza2trl5OSvr6/k5OTR0dHY2NiZmZlmZWX0
9PSIh4dgX19jYmJzcnJ0c3NaWVl9fHyzs7PQ8PfV1NTExMRhYWFtbW3Q0NBZWVlIR0dsz+dXVlbAv
7+bm5tnZmZ8e3tQT0/U1NSYl5f0+/20s7PNzc2H2OtdXV0Vs9lMS0suu92N2u1NxeKl4vHT8fh40+
mFhIS86vT+//8/Pj4XtNnp+PvB6/Xv+vzt+fz6/v5Fw+EWs9l91Orb29tEQ0P5/f656PR71OrZ8vn
D7PWr5PGQ2+1gy+Wh4fBqaWkxvN4ht9uK2eypqKiT3O3q+PxizObX19dJxOLPz8+o4/EdttpZWFhd
y+UluNw1vt79/f110unDw8Pp6OiB1urJ7fbl5eVbWlqEg4N7enoYtNmTkpJAweBtbGxy0ejN7/fl
9/u3t7f19fVRx+P2/P4ZtNqjo6PV8vjIx8dozuZBQECHhoZZyeSrq6v8/v9vbm7u+fwqutxlzeY6v
9+t5fLz8/Pd9PkzMjK25/NAPz+Y3u4btdrw+v2DIODLAAAZMElEQVR42u1dCVxVxf4fcE0lRMhUwC
```


This was a brief introduction to how you would send requests and get responses from a web server, while totally avoiding the web browser interface. Later we'll see how we can pick up the required data elements from the contents of the web page for analytical purposes.

Summary

In this lesson, we provided an introduction to the `requests` library in python. We saw how to use the get method to send requests to web servers, check server status, look at the header elements of a web page and how to send extra parameters like user information.