Requests in Python (Guide)

by Venmani A D (https://www.machinelearningplus.com/author/venmani/) |

Contents

Introduction to Requests Library

Requests is an elegant and simple Python library built to handle HTTP requests in python easily.

But what is a HTTP request?

HTTP is a set of protocols designed to enable communication between clients and servers. A client is typically a local computer or device similar to what you are using to view this page.

A HTTP request is the message sent (or received) from your local computer to a web server hosting, typically hosting a website.

For example, when you go to any internet website from your browser, the browser is sending a HTTP request and receives an appropriate 'response' from the host server.

Requests is an easy-to-use library with a lot of features ranging from passing additional parameters in URLs, sending custom headers, SSL Verification, processing the received response etc.

A GET request is used to request data from a specific server. It is the most common type of request. This is synonymous to you visiting the homepage of a

Recent Posts

Requests in Python (Guide) (https://www.machinelearningplus.com/python/req in-python/)

Matplotlib Pyplot (https://www.machinelearningplus.com/plots/matp pyplot/)

Python Boxplot (https://www.machinelearningplus.com/plots/pytho boxplot/)

Bar Plot in Python (https://www.machinelearningplus.com/plots/bar-plot-in-python/)

data.table in R – The Complete Beginners Guide (https://www.machinelearningplus.com/data-manipulation/datatable-in-r-complete-guide/)

Augmented Dickey Fuller Test (ADF Test) – Must Read Guide (https://www.machinelearningplus.com/time-series/augmented-dickey-fuller-test/)

KPSS Test for Stationarity (https://www.machinelearningplus.com/time-series/kpss-test-for-stationarity/)

101 R data.table Exercises (https://www.machinelearningplus.com/data-manipulation/101-r-data-table-exercises/)

P-Value – Understanding from Scratch (https://www.machinelearningplus.com/statistics/p-value/)

101 Python datatable Exercises (pydatatable) (https://www.machinelearningplus.com/data-manipulation/101-python-datatable-exercises-pydatatable/)

Vector Autoregression (VAR) – Comprehensive Guide with Examples in Python (https://www.machinelearningplus.com/time-series/vector-autoregression-examples-python/)

Mahalonobis Distance – Understanding the math with examples (python) (https://www.machinelearningplus.com/statistics/m distance/)

datetime in Python – Simplified Guide with Clear Examples (https://www.machinelearningplus.com/python/dat python-examples/)

Feedback

What is a GET and POST request?

website from your browser.

Another common type of request is the POST request, which is used to send data to a host server for further processing, like, to update a resource, such as a database.

What is this synonymous to in real world?

For example, most data that you submit through forms in various websites is sent and processed as a POST request.

Besides this, using requests you can add additional content like header information, form data, multipart files, and parameters via simple Python libraries. You don't need to manually append the query strings to your URLs.

What does that practically mean?

For example, if you search for something, say the string 'babies' in google, your browser sends a GET request to Google server by appending the query string to the url. So if you check the url in your address bar, you should see something like: https://www.google.com/search?q=babies (http://www.google.com/search?q=babies)

Sometimes, there are more information making the query strings complex to construct.

With `requests` library, you don't have to explicity construct such query strings. But rather pass it as additional parameter to `requests.get()`.

What makes requests really stand out is, the received response is packages as a standardized `Response` object.

It will contain all the response data (status, content, cookies etc). This makes further inspection and manipulation reliable and convenient.

Let's start by downloading the requests library using: `pip install requests`.

```
!pip install requests
```

Then import the library to use it in your program use the `import requests` command.

```
import requests
from pprint import pprint  # prettyprint
```

Now let's look into the details of request library and some of the important features.

GET Method

Principal Component Analysis (PCA) – Better Explained (https://www.machinelearningplus.com/machine-learning/principal-components-analysis-pca-better-explained/)

Python Logging – Simplest Guide with Full Code and Examples (https://www.machinelearningplus.com/python/pyt logging-guide/)

Matplotlib Histogram – How to Visualize Distributions in Python (https://www.machinelearningplus.com/plots/matp histogram-python-examples/)

ARIMA Model – Complete Guide to Time Series Forecasting in Python (https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/)

Time Series Analysis in Python – A Comprehensive Guide with Examples (https://www.machinelearningplus.com/time-series/time-series-analysis-python/)

Matplotlib Tutorial – A Complete Guide to Python Plot w/ Examples (https://www.machinelearningplus.com/plots/matp tutorial-complete-guide-python-plot-examples/)

Topic modeling visualization – How to present the results of LDA models? (https://www.machinelearningplus.com/nlp/topic-modeling-visualization-how-to-present-results-lda-models/)

Top Posts & Pages

ARIMA Model - Complete Guide to Time Series Forecasting in Python (https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/)

Time Series Analysis in Python - A Comprehensive Guide with Examples (https://www.machinelearningplus.com/time-series/time-series-analysis-python/)

Top 50 matplotlib Visualizations - The Master Plots (with full python code) (https://www.machinelearningplus.com/plots/top-50-matplotlib-visualizations-the-master-plots-python/)

Cosine Similarity - Understanding the math and how it works (with python codes) (https://www.machinelearningplus.com/nlp/cosine-similarity/)

Parallel Processing in Python - A Practical Guide with Examples (https://www.machinelearningplus.com/python/par processing-python/)

Topic Modeling with Gensim (Python) (https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/)

Feedback

Lets try to get some information from the official python website – https://www.python.org/ [https://www.python.org/].

You can use any public site. All you need to do is call `requests.get(url_name)`.

When you ping a website or portal for information it's considered as 'making a request'.

`requests.get()` is used exactly for this purpose.

You need to specify the web address that you need to ping as an argument to the function.

```
r = requests.get('https://www.python.org/ (https://www.python.org/)')
```

The information that we got from the website will be stored in the `Response` object we created `r`.

You can extract many features from this response object, like if you need to get the cookies that server sent, all you need to do is print `r.cookies`.

Now as I have requested the data from the website, to check whether it has worked properly, let's try priniting the `r` value.

```
print(r)
```

```
<Response [200]>
```

You can see that we have got `Resonse [200]`. Let's look into what this means.

STATUS Code

Status codes are issued by a server in response to a client's request made to the server.

Use the `r.status_code` command to return the status code for your request.

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

```
200
```

We have got a response of 200 which means the request is success.

A response of 200 means Success.

A response of 300 means Redirected.

A response of 400 means Client Error.

A response of 500 means Server Error.

101 Pandas Exercises for Data Analysis [https://www.machinelearningplus.com/python/101-pandas-exercises-python/]
Complete Introduction to Linear Regression in R [https://www.machinelearningplus.com/machine-learning/complete-introduction-linear-regression-r/]
Matplotlib Histogram - How to Visualize Distributions in Python [https://www.machinelearningplus.com/plots/matp histogram-python-examples/]
101 NumPy Exercises for Data Analysis (Python) [https://www.machinelearningplus.com/python/101-numpy-exercises-python/]

Tags
Bigrams [https://www.machinelearningplus.com/tag/bigrams/] Classification [https://www.machinelearningplus.com/tag/c Corpus [https://www.machinelearningplus.com/tag/corpus/] Cosine Similarity [https://www.machinelearningplus.com/tag/cosine-similarity/] data.table [https://www.machinelearningplus.com/tag/data-table/] Data Manipulation [https://www.machinelearningplus.com/ta manipulation/] Debugging [https://www.machinelearningplus.com/tag/debugging/] Doc2Vec [https://www.machinelearningplus.com/tag/doc2vec/] Evaluation Metrics [https://www.machinelearningplus.com/tag/evaluation-metrics/] FastText [https://www.machinelearningplus.com/tag/fasttext/] Feature Selection [https://www.machinelearningplus.com/tag/feature-selection/] Gensim [https://www.machinelearningplus.com/tag/gensi LDA [https://www.machinelearningplus.com/tag/lda/] Lemmatization [https://www.machinelearningplus.com/tag/lemmatization/] Linear Regression [https://www.machinelearningplus.com/tag/linear-regression/] Logistic [https://www.machinelearningplus.com/tag/logistic/] LSI [https://www.machinelearningplus.com/tag/lsi/] Matplotlib [https://www.machinelearningplus.com/ Multiprocessing [https://www.machinelearningplus.com/tag/multiprocessing/] NLP [https://www.machinelearningplus.com/tag/n NLTK [https://www.machinelearningplus.com/tag/nltk/] Numpy [https://www.machinelearningplus.com/tag/nump P-Value [https://www.machinelearningplus.com/tag/p-value/] Pandas [https://www.machinelearningplus.com/tag/pandas/] Parallel Processing

Feedback

A response of 404 means Page Not Found Error.

In general, any status code less than 400 means, the request was processed successfully. If it is 400 and above, some sort of error occurred.

You may need to use the if else statement about whether to further proceed into the code depending upon the status code you recieved from the server, as you don't need to further run your program after you have got an error.

Use the following code below in that case:

```
if r.status_code == 200:
    print('Success')
elif r.status_code == 404:
    print("Page not found")
```

```
Success
```

Contents of the Response object

dir(r) function is used to get details about what all useful information we can get from the data we retrived.

```
r_attribs = [c for c in dir(r) if not c.startswith("_")]
r_attribs
```

```
['apparent_encoding',
 'close',
 'connection',
 'content',
 'cookies',
 'elapsed',
 'encoding',
 'headers',
 'history',
 'is_permanent_redirect',
 'is_redirect',
 'iter_content',
 'iter_lines',
 'json',
 'links',
 'next',
 'ok',
 'raise_for_status',
 'raw',
 'reason',
 'request',
 'status_code',
 'text',
 'url']
```

You can see that there are several commands available such as headers, status_code, content, cookies etc.

(https://www.machinelearningplus.com/tag/parallel-processing/) Phraser (https://www.machinelearningplus.com/tag/phraser/) Practice Exercise (https://www.machinelearningplus.com/tag/practice-exercise/) Python (https://www.machinelearn R (https://www.machinelearningplus.cc Regex (https://www.machinelearningplus.com/tag/regex/) Regression (https://www.machinelearningplus.com/tag/regression/) Residual Analysis (https://www.machinelearningplus.com/tag/residual-analysis/) Scikit Learn (https://www.machinelearningplus.com/tag/scikit-learn/) Significance Tests (https://www.machinelearningplus.com/tag/signifi tests/) Soft Cosine Similarity (https://www.machinelearningplus.com/tag/soft-cosine-similarity/) spaCy (https://www.machinelearningplus.com/tag/spacy/) Stationarity (https://www.machinelearningplus.com/tag/statio Summarization (https://www.machinelearningplus.com/tag/summarization/) TaggedDocument (https://www.machinelearningplus.com/tag/taggeddocument/) TextBlob (https://www.machinelearningplus.com/tag/textblob/) TFIDF (https://www.machinelearningplus.com/tag/tfidf/) Time Series (https://www.machinelearningplus.com/ta series/) Topic Modeling (https://www.machinelearningplus.com/tag/topic-modeling/) Visualization (https://www.machinelearningplus.com/tag/visual Word2Vec (https://www.machinelearningplus.com/tag/word2vec/)

You can also use help(r) command to get more info about each of these. I'm showing some of the help info that's useful below.

```
print(help(r))
```

Feedback

```
Help on Response in module requests.models object:

class Response(builtins.object)
 |  The :class:`Response <Response>` object, which contains a
 |  server's response to an HTTP request.
 |
 |  Methods defined here:
 |
 |  __bool__(self)
 |      Returns True if :attr:`status_code` is less than 400.
 |
 |      This attribute checks if the status code of the response is between
 |      400 and 600 to see if there was a client error or a server error. If
 |      the status code, is between 200 and 400, this will return True. This
 |      is **not** a check to see if the response code is ``200 OK``.
 |
 |  __enter__(self)
 |
 |  __exit__(self, *args)
 |
 |  __getstate__(self)
 |
 |  __init__(self)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __iter__(self)
 |      Allows you to use a response as an iterator.
 |
 |  __nonzero__(self)
 |      Returns True if :attr:`status_code` is less than 400.
 |
 |      This attribute checks if the status code of the response is between
 |      400 and 600 to see if there was a client error or a server error. If
 |      the status code, is between 200 and 400, this will return True. This
 |      is **not** a check to see if the response code is ``200 OK``.
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  __setstate__(self, state)
 |
 |  close(self)
 |      Releases the connection back to the pool. Once this method has been
 |      called the underlying ``raw`` object must not be accessed again.
 |
 |      *Note: Should not normally need to be called explicitly.*
 |
 |  iter_content(self, chunk_size=1, decode_unicode=False)
 |      Iterates over the response data.  When stream=True is set on the
 |      request, this avoids reading the content at once into memory for
 |      large responses.  The chunk size is the number of bytes it should
 |      read into memory.  This is not necessarily the length of each item
 |      returned as decoding can take place.
 |
 |      chunk_size must be of type int or None. A value of None will
 |      function differently depending on the value of `stream`.
 |      stream=True will read data as it arrives in whatever size the
 |      chunks are received. If stream=False, data is returned as
 |      a single chunk.
 |
```

Feedback

```
 |      If decode_unicode is True, content will be decoded using the best
 |      available encoding based on the response.
 |  
 |  iter_lines(self, chunk_size=512, decode_unicode=False, delimiter=None)
 |      Iterates over the response data, one line at a time.  When
 |      stream=True is set on the request, this avoids reading the
 |      content at once into memory for large responses.
 |      
 |      .. note:: This method is not reentrant safe.
 |  
 |  json(self, **kwargs)
 |      Returns the json-encoded content of a response, if any.
 |      
 |      :param \*\*kwargs: Optional arguments that ``json.loads`` takes.
 |      :raises ValueError: If the response body does not contain valid json.
 |  
 |  raise_for_status(self)
 |      Raises stored :class:`HTTPError`, if one occurred.
 |  
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |  
 |  __dict__
 |      dictionary for instance variables (if defined)
 |  
 |  __weakref__
 |      list of weak references to the object (if defined)
 |  
 |  apparent_encoding
 |      The apparent encoding, provided by the chardet library.
 |  
 |  content
 |      Content of the response, in bytes.
 |  
 |  is_permanent_redirect
 |      True if this Response one of the permanent versions of redirect.
 |  
 |  is_redirect
 |      True if this Response is a well-formed HTTP redirect that could have
 |      been processed automatically (by :meth:`Session.resolve_redirects`).
 |  
 |  links
 |      Returns the parsed header links of the response, if any.
 |  
 |  next
 |      Returns a PreparedRequest for the next request in a redirect chain, if
 |  
 |  ok
 |      Returns True if :attr:`status_code` is less than 400, False if not.
 |      
 |      This attribute checks if the status code of the response is between
 |      400 and 600 to see if there was a client error or a server error. If
 |      the status code is between 200 and 400, this will return True. This
 |      is **not** a check to see if the response code is ``200 OK``.
 |  
 |  text
 |      Content of the response, in unicode.
 |      
 |      If Response.encoding is None, encoding will be guessed using
 |      ``chardet``.
 |      
```

Feedback

```
 |      The encoding of the response content is determined based solely on HTTP
 |      headers, following RFC 2616 to the letter. If you can take advantage of
 |      non-HTTP knowledge to make a better guess at the encoding, you should
 |      set ``r.encoding`` appropriately before accessing this property.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __attrs__ = ['_content', 'status_code', 'headers', 'url', 'history', '...

None
```

The Content

The output from the requests.get() , that is, the Response object contains many useful information.

Use the r.content command to get the access to the raw data we got. This is the raw output of the html content behind the URL we requested, which in this case is https://www.python.org/ [https://www.python.org/].

```
# Printing first 200 characters
print(r.content[:200])
```

```
b'<!doctype html>\n<!--[if lt IE 7]>   <html class="no-js ie6 lt-ie7 lt-ie8 lt-
```

While .content gives you access to the raw bytes of the response, you need to convert them into a string using a character encoding such as UTF-8.

You get that directly by using another stored attribute in the response called the r.text .

The Full HTML source as Text

Use the text() command to get the content from the website as a unicode response.

Unicode is a standard way for encoding characters.

Unicode string is a python data structure that can store zero or more unicode characters.

This output should match with what you see when you right click on the webpage and view its page source.

```
print(r.text[:800])
```

Feedback

```
<!doctype html>
<!--[if lt IE 7]>   <html class="no-js ie6 lt-ie7 lt-ie8 lt-ie9">   <![endif]--
<!--[if IE 7]>      <html class="no-js ie7 lt-ie8 lt-ie9">          <![endif]--
<!--[if IE 8]>      <html class="no-js ie8 lt-ie9">                 <![endif]--
<!--[if gt IE 8]><!--><html class="no-js" lang="en" dir="ltr">  <!--<![endif]-->

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <link rel="prefetch" href="//ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js (htt

    <meta name="application-name" content="Python.org">
    <meta name="msapplication-tooltip" content="The official home of the Python Programming L
    <meta name="apple-mobile-web-app-title" content="Python.org">
    <meta name="apple-mobile-web-app-capable" co
```

Retrieving an image from the website

Use the same requests.get() command to retrieve the image.

I am using the image from the url – https://www.python.org/static/img/python-logo.png (https://www.python.org/static/img/python-logo.png)

The received response is also a Response object. The image is stored in r.content, which you can write to a file. This means, whatever be the content of the received response, be it text or image, it is stored in the content attribute.

```
# Retrieve image and save in local
r = requests.get("https://www.python.org/static/img/python-logo.png (https:/

# 'wb' represents write byte mode
with open('file.png','wb') as f:
    f.write(r.content)
```

The image from the website will be downloaded to the folder in which you are running the program.

Headers

Most webpages you visite will contain header, which contains various metadata.

Use the r.headers command to access the information in the header of the page.

What does r.header contain?

If you look at the output from the r.header, youâ€™ll see that it is actually a serialized JSON content.

More information like metadata about the response, it is stored in the header.

Feedback

It gives you many information such as the content type of the response payload, a time limit on how long to cache the response, and more.

This will return you a dictionary-like object, allowing you to access header values by key.

```
 # Header contents
for key, value in r.headers.items():
  print(key, ":", value)
```

```
 Server : nginx
Content-Type : text/html; charset=utf-8
X-Frame-Options : DENY
Via : 1.1 vegur, 1.1 varnish, 1.1 varnish
Content-Length : 48918
Accept-Ranges : bytes
Date : Fri, 17 Apr 2020 05:55:13 GMT
Age : 1334
Connection : keep-alive
X-Served-By : cache-bwi5145-BWI, cache-dca17774-DCA
X-Cache : HIT, HIT
X-Cache-Hits : 3, 2
X-Timer : S1587102914.994481,VS0,VE0
Vary : Cookie
Strict-Transport-Security : max-age=63072000; includeSubDomains
```

As you can see, it gives information about the content type, last modified date, Age of the website etc..

You can access each one of these by considering the ouput from the function as dictionary.

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

```
'48918'
```

Advanced Functions

Now as we have looked into the basics of the requests library, lets dive into some of the advanced functions

From now onwards, I will be using the website – http://httpbin.org/ (https://httpbin.org/) to retrive as well as send and store information.

Let's try the commands that you have learned so far.

How to Set Query String Parameters

Often the response given by the server differs based on the query you send. Like, you want to view the 2nd page of a 10 page article, instead of the first page. Or you want to search for a particular term in a website.

Feedback

In such cases, you will send additional parameters as part of the URL as a query. For example: https://www.google.com/search?q=babies (https://www.google.com/search?q=babies) will return the search results of 'babies'.

Depending on the device you are using, location, referring source, user etc, these queries can easily get complicated.

So instead of adding it in the url directly, using requests.get() , you can pass it as a separate parameter using the params argument.

Let's add couple of parameters to the query string, that is, page=5 and count=10 of httpbin.org . This essentially translates as "http://httpbin.org/?page=5&count=10 (https://httpbin.org/?page=5&count=10)".

```
# Setting parameters
parameter= {'page':5 , 'count':10}
r=requests.get('http://httpbin.org/ (http://httpbin.org/)', params=parameter)
print(r.text[:600])
```

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>httpbin.org</title>
    <link href="https://fonts.googleapis.com/css?family=Open+Sans:400,700|Source+Code+P
        rel="stylesheet">
    <link rel="stylesheet" type="text/css" href="/flasgger_static/swagger-ui.css">
    <link rel="icon" type="image/png" href="/static/favicon.ico" sizes="64x64 32x32 16x16" />
    <style>
        html {
            box-sizing: border-box;
            overflow: -moz-scrollbars-vertical;
            overflow-y: scroll;
        }

        *,
```

You can see that I have first created a dictionary for the parameters and then I passed it into the get() function.

And we got a json response back from the httpbin website.

To check whether the parameter passing has worked properly, use r.url to check the parameters we have passed.

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

```
http://httpbin.org/?page=5&count=10
```

It has worked properly as the page is set to 5 and the count is set to 10.

Feedback

You can also pass it as a tuple or a byte which will give the same output.

```
 # Setting parameters
parameter= (('page',5) , ('count',10))
r=requests.get('http://httpbin.org/ (http://httpbin.org/)', params=parameter)
print(r.text[:400])
```

```
 <!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>httpbin.org</title>
    <link href="https://fonts.googleapis.com/css?family=Open+Sans:400,700|Source+Code+P
        rel="stylesheet">
    <link rel="stylesheet" type="text/css" href="/flasgger_static/swagger-ui.css">
    <link rel="icon" type="image/png" href="/static/favicon.ico" si
```

POST Method

The POST method is used to submit data to be further handled by the server. The server typically understands the context and knows what to do with the data.

Generally it's used while submitting a web form or when uploading a file to the server.

The `requests.post()` function allows you to do this.

Let's look into an example with `httpbin.org` website.

```
 # POST request
param = { 'custname':'abcd', 'custemail': 'efgh@gmail.com'}
r = requests.post('http://httpbin.org/post (http://httpbin.org/post)', data=p

# As we are getting a json response, instead of using the text command, I am us
pprint(r.json())
```

Feedback

```
{'args': {},
 'data': '',
 'files': {},
 'form': {'custemail': 'efgh@gmail.com', 'custname': 'abcd'},
 'headers': {'Accept': '*/*',
             'Accept-Encoding': 'gzip, deflate',
             'Content-Length': '40',
             'Content-Type': 'application/x-www-form-urlencoded',
             'Host': 'httpbin.org',
             'User-Agent': 'python-requests/2.21.0',
             'X-Amzn-Trace-Id': 'Root=1-5e9c69e3-44d2f060dfeb7a401ffe7c28'},
 'json': None,
 'origin': '35.196.30.16',
 'url': 'http://httpbin.org/post (http://httpbin.org/post)'}
```

You can see in the form type the custname and the custemail has been recorded.

If you need to pass some form values into it, then you need to look into the source of the url and find out what kind of values the form expects.

To process the received json response, iterate through the contents of r.json() . Or if you know what the contents, you can access it directly as you would with a dict .

```
for k, v in r.json().items():
  print(k, ": ", v)
```

```
 args :  {}
data :  
files :  {}
form :  {'custemail': 'efgh@gmail.com', 'custname': 'abcd'}
headers :  {'Accept': '*/*', 'Accept-Encoding': 'gzip, deflate', 'Content-Lengtl
json :  None
origin :  35.196.30.16
url :  http://httpbin.org/post (http://httpbin.org/post)
```

Post function can be used to send large amount of data (text / binary) data.

PUT Method

The PUT method requests that the data you are sending to be stored under the supplied URL.

If the URL refers to an already existing resource, it is modified and if the URL does not point to an existing resource, then the server creates the resource with that URL.

As you can see, the PUT is somewhat similar in functionality to POST.

So what is the difference between PUT and POST?

Feedback

The difference is, the POST method sends data to a URI and the the receiving resource understands the context and knows how to handle the request. Whereas, in a PUT method, if there is a file in the given URI, it gets replaced. And if there isn't any, a file is created.

Besides, not matter how many times you execute a given PUT request, the resulting action is always the same. This property is called idempotence. Whereas, for a POST method, the response need not always be the same.

This makes the PUT method idempotent and the POST method is not.

To make a PUT request, use the requests.put() method.

```
import requests
r = requests.put('https://httpbin.org/put (https://httpbin.org/put)', data =
print(r)
print(r.content)
```

```
<Response [200]>
b'{\n  "args": {}, \n  "data": "", \n  "files": {}, \n  "form": {\n    "name": '
```

Generally in practice, put() function is used for updating operations and post() function is used for creating operations.

DELETE Method

The delete() method sends a DELETE request to the specified url.

DELETE requests are made for deleting the specified resource (file, record etc).

A successful response should be:

1. 200 (OK) if the response includes an entity describing the status.
2. 202 (Accepted) if the action has not yet been enacted
3. 204 (No Content) if the action has been enacted but the response does not include an entity.

```
import requests
r = requests.delete('https://httpbin.org/delete (https://httpbin.org/delete)
print(r)
print(r.json())
```

```
<Response [200]>
{'args': {}, 'data': '', 'files': {}, 'form': {'name': 'abcd'}, 'headers': {'Ac
```

The delete() function will request the server to delete a resource that you have specified in the URL.

Feedback

The client, however, cannot be guaranteed that the operation has been carried out.

PATCH Method

The PATCH method is a request method supported by the HTTP protocol for making partial changes to an existing resource.

The main difference between the PUT and PATCH method is that the PUT method uses the request URL to supply a modified version of the requested resource. And it replaces the original version of the resource.

Whereas, the PATCH method only supplies a set of instructions to modify the resource.

This means, the PATCH request needs to contain only the changes that needs to be applied to a resource and not the entire resource.

Although it resembles PUT, it typically contains a set of instructions that tell how a resource residing at a URI should be modified to produce a new version.

Use the `requests.patch()` command to implement this.

So when to use PATCH?

Whenever you want to make only partial changes to the resource.

```
 import requests
r = requests.patch('https://httpbin.org/patch (https://httpbin.org/patch)',
print(r)
pprint(r.json())
```

```
 <Response [200]>
{'args': {},
 'data': '',
 'files': {},
 'form': {'name': 'abcd'},
 'headers': {'Accept': '*/*',
             'Accept-Encoding': 'gzip, deflate',
             'Content-Length': '9',
             'Content-Type': 'application/x-www-form-urlencoded',
             'Host': 'httpbin.org',
             'User-Agent': 'python-requests/2.21.0',
             'X-Amzn-Trace-Id': 'Root=1-5e9c6fd2-fea9a805120b3ab9b59bf62a'},
 'json': None,
 'origin': '35.196.30.16',
 'url': 'https://httpbin.org/patch (https://httpbin.org/patch)'}
```

HEAD Method

`head()` function is useful for retrieving only the meta-information written in response headers, without having to transport the entire content like the `get()` command.

Feedback

This method is often used for testing hypertext links for validity, accessibility, and recent modification.

You can do this by using the `requests.head()` command with the web address and data as argument.

```
 import requests
r = requests.head('https://httpbin.org/ (https://httpbin.org/)', data ={'key'
print(r)
print(r.headers)
pprint(r.text)
```

```
 <Response [200]>
{'Date': 'Sun, 19 Apr 2020 15:45:25 GMT', 'Content-Type': 'text/html; charset=ut
''
```

Let's run the same as a GET request and see the difference.

```
 r = requests.get('https://httpbin.org/ (https://httpbin.org/)', data ={'key'
print(r)
print(r.headers)
pprint(r.text[:500])
```

```
 <Response [200]>
{'Date': 'Sun, 19 Apr 2020 15:49:24 GMT', 'Content-Type': 'text/html; charset=ut
('<!DOCTYPE html>\n'
 '<html lang="en">\n'
 '\n'
 '<head>\n'
 '    <meta charset="UTF-8">\n'
 '    <title>httpbin.org</title>\n'
 '    <link '
 'href="https://fonts.googleapis.com/css?family=Open+Sans:400,700|Source+Code+Pro:300,60
 '        rel="stylesheet">\n'
 '    <link rel="stylesheet" type="text/css" '
 'href="/flasgger_static/swagger-ui.css">\n'
 '    <link rel="icon" type="image/png" href="/static/favicon.ico" '
 'sizes="64x64 32x32 16x16" />\n'
 '    <style>\n'
 '        html {\n'
 '            box-sizing: border-box;\n'
 '          ')
```

Notice, we received only header content with `requests.head()`. Rest of the content is ignored. So, it will save time and resource if you are interested only in the header content.

Request Header

Feedback

A request header is an HTTP header that can be used in an HTTP request, and that doesn't relate to the content of the message.

To customize headers, you pass a dictionary of HTTP headers to `get()` using the `headers` parameter.

Through the 'Accept' header, the client tells the server what content types your application can handle.

```
import requests
r=requests.get("http://www.example.com/ (http://www.example.com/)", headers=
r
```

```
<Response [200]>
```

Inspecting the request made

When you make a request, the requests library prepares the request before actually sending it to the destination server.

Request preparation includes things like validating headers and serializing JSON content.

Only after preparing the request, the request will be sent to the destination server.

You can view the PreparedRequest by accessing the json.

```
# Inspect the URL
param= { 'username':'abcd', 'password': 'efgh'}

# We are passing this param as the data
r=requests.post('http://httpbin.org/post (http://httpbin.org/post)',data=par
r.request.url
```

```
'http://httpbin.org/post (http://httpbin.org/post)'
```

This helps you in getting access to informations like payload, URL, headers, authentication, and more.

```
# Inspect the Parameters
param= { 'username':'abcd', 'password': 'efgh'}
r=requests.post('http://httpbin.org/post (http://httpbin.org/post)',data=par

# As we are getting a json resposne, instead of using the text command, I am us
dict=r.json()
print(dict['form'])
```

Feedback

```
{'password': 'efgh', 'username': 'abcd'}
```

You can see that I have stored the ouput of the r.json in a dictionary and I can accesing different information from the dictionary separately.

Authentication

Authentication helps a service understand who you are.

You provide your credentials to a server by passing data through the Authorization header or a custom header defined by the service.

You need to use the auth command to do this.

```
r= requests.get('http://httpbin.org/basic-auth/abcd/efgh (http://httpbin.or
print(r.text)
```

```
{
  "authenticated": true,
  "user": "abcd"
}
```

What we are doing is we are giving our data to the server by passing data through the authorization header.

If we go into the httpbins website, you can see that the basic authentication format for the website is of the form http://httpbin.org/basic-auth/username/password (http://httpbin.org/basic-auth/username/password) .

In this the username and password will be what we have specified.

The authentication output comes out to be 'true' which means that our username and password is correct.

If our password is wrong, we wont be getting any output for authentication.

```
r= requests.get('http://httpbin.org/basic-auth/abcd/efgh (http://httpbin.or
print(r)
```

```
<Response [401]>
```

You can see that if I use the wrong username, I am getting an 401 error.

When you pass your username and password in a tuple to the auth parameter, requests is applying the credentials using HTTPâ€™s Basic access authentication scheme under the hood.

Time out

Feedback

When you send a request to a server, your system typically waits for a certain amount of time for the other server to respond.

If this takes too much times, then there is a possibility that your system will hang.

Time-out is set to make sure that the if the website is not responding for a certain amount of time it has to stop loading.

If we dont set the timeout then the website will be loading forever if the server is not responding.

Use the `timeout=` command to set the time limit in seconds.

```
r= requests.get('http://httpbin.org/basic-auth/abcd/efgh (http://httpbin.org
```

In the above case, I have set the time limit to 5 seconds.

You can also pass a tuple to timeout with the first element being a connect timeout (the timeout allows for the client to establish a connection to the server), and the second being a read timeout (the time it will wait on a response once your client has established a connection):

```
r=requests.get('http://httpbin.org/basic-auth/abcd/efgh (http://httpbin.org
```

This means that the request should establish a connection with server before 3 seconds and the data must be recieved within 7 seconds after the connection is established.

If the request times out, then the function will raise a Timeout exception.

You can handle this exception as well by importing `Timeout` from `requests.exceptions` .

```
from requests.exceptions import Timeout

try:
    response = requests.get('http://httpbin.org/basic-auth/abcd/efgh (http://
except Timeout:
    print('Request timed out')
else:
    print('Request went through')
```

SSL Certificate Vertification

SSL Certificates are small data files that digitally bind a cryptographic key to an organization details.

Feedback

An organization needs to install the SSL Certificate onto its web server to initiate a secure session with browsers. Once a secure connection is established, all web traffic between the web server and the web browser will be secure.

If the data you are trying to receive or send is sensitive, then it is done by establishing a encrypted connection using SSL.

requests library does this for you by default.

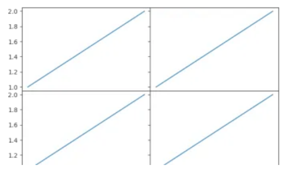
If you dont want to do this, then we can set the paramter verify to be false in the `get()` function.

```
requests.get('http://httpbin.org/basic-auth/abcd/efgh (http://httpbin.org/ba
```

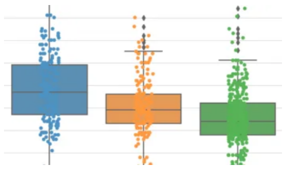
```
<Response [401]>
```

Related

[https://www.machinelearningplus.com/python/python-property/]
Python @Property Explained - How to Use and When? (Full Examples) [https://www.machinelearningplus.com/python/python-property/]
November 5, 2018
In "Python"



[https://www.machinelearningplus.com/plots/matplotlib-pyplot/]
Matplotlib Pyplot [https://www.machinelearningplus.com/plots/matplotlib-pyplot/]
April 18, 2020
In "Plots"



[https://www.machinelearningplus.com/plots/python-boxplot/]
Python Boxplot [https://www.machinelearningplus.com/plots/python-boxplot/]
April 16, 2020
In "Plots"