**What Is Requests Module?**

Requests is a Python module that you can use to send all kinds of HTTP requests. It is an easy-to-use library with a lot of features ranging from passing parameters in URLs to sending custom headers and SSL Verification. In this tutorial, you will learn how to use this library to send simple HTTP requests in Python.

Requests allow you to send HTTP/1.1 requests. You can add headers, form data, multi-part files, and parameters with simple Python dictionaries, and access the response data in the same way.

**Making a GET Request**

It is fairly straightforward to send an HTTP request using Requests. You start by importing the module and then making the request. Check out the example:

|  |  |
| --- | --- |
| 1  2 | import requests  req = requests.get('<https://www.google.co/>') |

So, all the information is stored somewhere, correct?

Yes, it is stored in a Response object called as **req.**

Let’s say, for example, you want the encoding of a web-page so that you can verify it or use it somewhere else. This can be done using the **req.encoding**property.

An added plus is that you can also extract many features like the status code for example (of the request). This can be done using the **req.status\_code** property.

|  |  |
| --- | --- |
| 1  2 | req.encoding # returns 'utf-8'  req.status\_code # returns 200 |

We can also access the cookies that the server sent back. This is done using **req.cookies,**as straightforward as that! Similarly, you can get the response headers as well. This is done by making use of **req.headers.**

Do note that the **req.headers** property will return a case-insensitive dictionary of the response headers. So, what does this imply?

This means that **req.headers[‘Content-Length’]**, **req.headers[‘content-length’]** and **req.headers[‘CONTENT-LENGTH’]**will all return the value of the just the **‘Content-Length’** response header.

We can also check if the response obtained is a well-formed HTTP redirect (or not) that could have been processed automatically using the **req.is\_redirect** property. This will return **True** or **False**based on the response obtained.

You can also get the time elapsed between sending the request and getting back a response using another property. Take a guess? Yes, it is the **req.elapsed** property.

Remember the URL that you initially passed to the **get()**function? Well, it can be different than the final URL of the response for many reason and this includes redirects as well.

And to see the actual response URL, you can use the **req.url** property.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | import requests  req = requests.get('<http://www.google.co/>')    req.encoding # returns 'utf-8'  req.status\_code # returns 200  req.elapsed # returns datetime.timedelta(0, 1, 666890)  req.url # returns '<https://google.co/>'    req.history  # returns [&lt;Response [301]&gt;, &lt;Response [301]&gt;]    req.headers['Content-Type']  # returns 'text/html; charset=utf-8' |

Don’t you think that getting all this information about the webpage is nice? But, the thing is that you most probably want to access the actual content, correct?

If the content you are accessing is text, you can always use the **req.text** property to access it. Do note that the content is then parsed as unicode only. You can pass this encoding with which to decode this text using the **req.encoding** property like we discussed earlier.

In the case of non-text responses, you can access them very easily. In fact it’s done in binary format when you use **req.content.** This module will automatically decode **gzip**and **deflate** transfer-encodings for us. This can be very helpful when you are dealing directly with media files. Also, you can access the JSON-encoded content of the response as well, that is if it exists, using the **req.json()**function.