Python: how to use the urllib module to make HTTP requests

In this article we will discover the basics of HTTP requests in Python using the urllib core module.

When making an HTTP request, you must first verify that the protocol specified in the URL is HTTP or HTTPS. We will be using HTTPS in this example, but you may be required to use plain HTTP if the remote site does not have an SSL certificate.

`urllib.request` has the `Request` class which, as mentioned in the documentation, has the primary purpose of handling HTTP URLs.

Our HTTP request begins by creating an instance of this class and passing a URL to it.

```
from urllib.request import urlopen, Request

request_url = 'https://gabrieleromanato.com'
request = Request(request_url)
```

Now we need to actually open the remote URL using the `urlopen()` function which creates a stream. Being a stream, the recommended way to manage it is to use a context manager with the `with` operator.

```
with urlopen(request) as response:
    pass
```

`response` is an instance of `urllib.response` and has the properties `status`, which returns the integer HTTP status code returned by the server, and `headers` which returns the HTTP headers of the response as an instance of the `EmailMessage` class. The `EmailMessage.items()` method returns a list of tuples containing the HTTP header name and its value. We can also convert this list of tuples into a list of dictionaries using the `map()` function.

```
def create_single_header(data):
    key, value = data
    d = {}
    d[key] = value
    return d

request_url = 'https://gabrieleromanato.com'
request = Request(request_url)

with urlopen(request) as response:
    status_code = response.status
    res_headers = list(map(create_single_header,
response.headers.items()))
```

Instead, to find the body of the response returned by the server, i.e. the HTML document, we must bear in mind that within our context manager we are operating with a stream of bytes, therefore we can use the same approach that we use with files (`read()` and `decode()`).

```
request_url = 'https://gabrieleromanato.com'
request = Request(request_url)

with urlopen(request) as response:
    status_code = response.status
    res_headers = list(map(create_single_header,
response.headers.items()))
    body = response.read().decode('utf-8')
```

Our code, however, is missing a key part: exception handling. In fact, the URL may not be valid, the server may return an HTTP error or the connection may time out. We can add exception handling like this.

```
from urllib.error import HTTPError, URLError
from urllib.request import urlopen, Request



def create_single_header(data):
    key, value = data
    d = {}
    d[key] = value
    return d



def send_http_request(request_url):
    if not request_url.startswith('https://'):
        return False
    try:
        request = Request(request_url)
        with urlopen(request) as response:
            status_code = response.status
            res_headers =
list(map(create_single_header,
response.headers.items()))
            body = response.read().decode('utf-8')
            return {
                'status': status_code,
                'headers': res_headers,
                'body': body
            }
    except URLError as err:
        return err.reason
    except HTTPError as error:
        return error
    except TimeoutError as tm_err:
        return tm_err
```

Finally, we can use our code as follows:

```
def main():

print(send_http_request('https://gabrieleromanato.com
'))



if __name__ == '__main__':
    main()
```