# Http client

[Ugeopgave 1] [*]

Mirza Hasanbasic

## ABSTRACT

In this assignment we will be looking at how to implement a simple HTTP client, that will have a subset of the entire HTTP protocol. The HTTP client will still be able to communicate with servers. There will be performance and validation discussion about the HTTP client as well as the limitations and testing of the client.

## 1. THE CLIENT

I have written the HTTP client in python and you are able to run in from a terminal. The file should be executable and in order to run it you need to write two arguments where the first is the desired website you wish to send a request to and the second argument is the name the savefile should have. There is a possibility for a third argument, but this has not been implemented yet. The third argumentive would only have been useful to us since it would work like a debug mode where we can print, but due to time constraint we have not done this.

To run the HTTPclient you simply write the following in the console `./HTTPclient.py 'website' 'filename'` remember that if you download an `.exe` file, the filename should have an extension to match it.

The libraries I use are myparser, socket and sys in the HTTP client and in my parser I use itertools and unittest. The unittest is a framework I use in myparser to test if I get a correct output

The function that I use to split my url is `stringToUrl(myurl)`, where the variable name `myurl` the string I wish to parse. I start with splitting after `'//'` and then I check wether the user have given me `HTTPS` as input. If this is true, then I raise an error and tell the user that `HTTPS` is not implemented and that they should use `HTTP` instead. After this I check the length of my list. I the length is greater than 1 then the user have written `http://www.example.com` but if the length is 1 then the user have written `www.example.com` and if this is true, then I add the `HTTP` for the user and insert both the scheme and netloc into a list.[1] From this point I check wether the netloc contains `:` or `/`. If the netloc has `:` then it means that there is a port and I split it. After I split it I add `:` in between the netloc and the port number i.e. I now have [scheme // netloc : port]. If the netloc have any folder then in my list I have [... port/folder/folder2] and then I need to check if this is true. If there are any folders I split again and then I flatten the list, because I end up with a list of lists. In the end of the parser I add a `/` after the netloc and `//` before the netloc so the format should be as `http_URL = "http:" "//" host [ ":" port ] [ abs_path [ "?" query ]]` For now I only need to implement the query part, so input as - `en.wikipedia.org/wiki/Uniform_Resource_identifier#Examples` won't work

The HTTP client will check wether you have given a port as part of the connection string or not. If not, then the default port will be 80. As for now, I are using `HTTP/1.0` because it was not possible to connect to some sites with `HTTP/1.1`. This HTTP client can do redirect if you should get an error code 30x. It will use recursion to find the new location. If you should be able to get a status code 200, then the client will write to the filename the `GET` request.

### 1.1 Code structure

I have two files (`HTTPClient.py` and `myurlparser.py`). In the `HTTPClient.py` i have two functions. One is the clientconnect and the second one handle saving to files. In the `myurlparser.py` i only have an urlparser as function and this function handles the parse of the url

### 1.2 Performance and validation

The performance of the client is fine. I have manually tried to time the download speed, since I have not had time to implement this yet and the download speed matches the speed when I use google chrome with a 5% in difference. In short the validation matches, but only if you download an `.exe` file. See the subsection testing for more in detail description about this. The client can redirect the user if you should run into this.

### 1.3 Client limitations

For now, our urlparser doesn't handle a query in the link. This is a limitation in the urlparser the I have implemented. As I explained in the subsection "Performance and valida-

[*]

© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

[1]The scheme is 'http' where as the netloc is 'www.example.com'

tion" our client doesn't handle other files than `.exe` since we get some random input in the first line of a file if it is a `.txt`. In the subsection testing I have explained this and given an example.

## 1.4 Testing

For the urlparser I have unittesting to check wether the code is following the format `http_URL = "http:" "//" host [ ":" port ] [ abs_path [ "?" query ]]`. There are no error and the unittest can be found in urlparser60.py and in the appendix I have a picture of the unittest if you do not wish to run it.

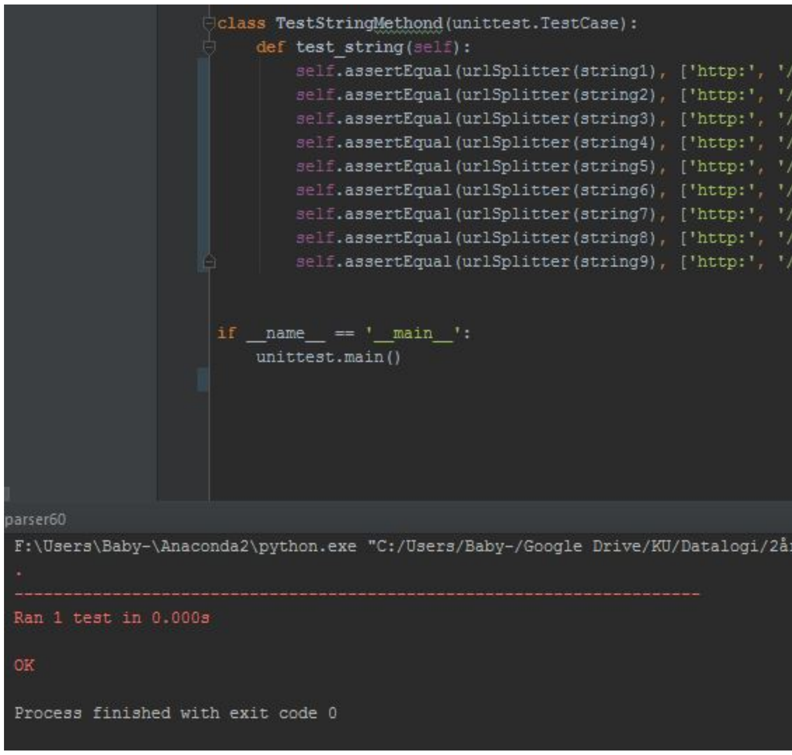
The HTTPclient should be able to download a file, this I have tried and after the file is downloaded I have checked the shasum against what it should be. The way I do this, is in the console where I write `./HTTPclient 'website' 'filename'`[2]. I check the checksum manually, so in my console I run `shasum -a 256 filename` and it matches the checksum. But this does only work with `.exe` files. If I try to check against a `.txt` or .html file then in the top of the file I get a random input in the file i.e. the file changes. I have not been able to solve this, so for now the HTTPclient will only work with an `.exe` file. As an example to the `.txt` file error I get, I have used this site. As you can see the first line is Site: ftp-master.debian.org but when I send a request with my HTTPclient and download the file my first line will be 1f90 or something else and the second line is `Site: ftp-master.debian.org` as in the link. So for now this HTTPclient only works for `.exe` files.

## APPENDIX

## A. HEADINGS IN APPENDICES

As seen in the picture below, the the unittest is running as it should, with no errors

[2]The sha256 sum and the file I download The executable



```
class TestStringMethond(unittest.TestCase):
    def test_string(self):
        self.assertEqual(urlSplitter(string1), ['http:', '/
        self.assertEqual(urlSplitter(string2), ['http:', '/
        self.assertEqual(urlSplitter(string3), ['http:', '/
        self.assertEqual(urlSplitter(string4), ['http:', '/
        self.assertEqual(urlSplitter(string5), ['http:', '/
        self.assertEqual(urlSplitter(string6), ['http:', '/
        self.assertEqual(urlSplitter(string7), ['http:', '/
        self.assertEqual(urlSplitter(string8), ['http:', '/
        self.assertEqual(urlSplitter(string9), ['http:', '/


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

```
F:\Users\Baby-\Anaconda2\python.exe "C:/Users/Baby-/Google Drive/KU/Datalogi/2å
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK

Process finished with exit code 0
```