Q&A: Assignment 1

Q1: Are we allowed to use urllib.parse for splitting URL strings into components?

Ans: No. Also note that any python package or module not supported by linux.csc.uvic.ca should not be used.0

Q2: Is there a password protected site we could test our code on? I was trying to find one but had no such luck.

Ans: One example is https://httpbin.org/basic-auth/myuser/mypass.

Q3: If I have found out that a server supports http/2, should I change the *http version field* in the HTTP request messages to http/2?

Ans: No! Note that http/2 does not change the message syntax of HTTP/1.1. Refer to RFC 7540 (https://httpwg.org/specs/rfc7540.html#discover-http).

Q4: If a website supports HTTP/2, can I still send it an HTTP query using HTTP/1.1?

Ans: Yes. If a website supports HTTP/2, you can typically still send requests using HTTP/1.1. Most websites' servers typically support both HTTP/1.1 and HTTP/2 for backward compatibility, allowing clients to continue using the HTTP/1.1 protocol. Note that for this assignment, **we only need to check whether or not a server supports HTTP/2**. Using HTTP/2 for further communication is not mandatory.

Q5: Is there a limit to the redirects that this assignment should have. Assume that redirects could happen multiple times, eg: 301 X Perm move to Location: x / 302 x move to Location: y / 301 y move .... I saw this on Google: "Don't use **more than 3 redirects** in a redirect chain. Google Bot will not follow 301 redirects over multiple hubs." With this in mind should I limit the redirects to 3, or do we want it to go on for even longer.

Ans: From the viewpoint of coding, it is much more easier to recursively call a function of handling redirects than putting a limit on the number of times of handling redirect. You can also use a *while* loop to handle redirection without checking the times of redirections, i.e., the *while* loop terminates if the response code is not 3xx. In this way, you do not need to worry about how many times redirection may happen.

Note that if your code is correct, it would be impossible to get stuck in the infinite loop of 302/301 redirect. The looping redirects do not exist in correctly-configured real-world web systems.

Q6: Should we be trying to decompress gzipped responses?

Ans: No need.

Q7: I used s.connect(("www.uvic.ca",80)) but it generates error 302. What do I do next?

Ans: Use https protocol to request file. When you use port 443 you will have to look into SSL package. Here is the documentation for SSL. https://docs.python.org/3/library/ssl.html

Q8: When I run my code with 'uvic.ca' as the input URI and send a HTTP 1.1 request, I get back a HTTP 1.0 response:

```
HTTP/1.0 302 Found
Location: https://www.uvic.ca/
Server: BigIP
Connection: Keep-Alive
Content-Length: 0
```

Therefore this has my output for supporting HTTP 1.1 request fails if I simply use the version number listed in the 301/302 response.

However, if I use 'www.uvic.ca' as my input with the same code, I do get a 'HTTP/1.1 200 OK' response.

Are we supposed to, in the case of **301/302 responses**, use the location given back to us and create a new connection in order to get the correct HTTP responses, or can we assume that our program will be run only with URI where the content lives, and not alias URIs such as 'uvic.ca'?

Ans: You do not need to handle the alias further. In this case, your code may get different answers for uvic.ca and www.uvic.ca, and this is accepted as correct. However, if your code handles the re-route and sends further queries to check the re-routes, it is even better. This is definitely accepted as correct. To help TA mark your assignments, it is strongly suggested that you output intermediate results rather than just your final answer.

Overall, handling different scenarios caused by different DNS settings is not the focus of this assignment.

Q9: For error handling, would you prefer that we set up custom exceptions and raise them with a useful error message when an error does occur, or just handle all errors, print an error message and exit?

Ans: While we did not specify the granularity of reporting errors, please follow common sense and reasonable programming practice. In a practical sense, output "Error! Exit" for any type of exceptions is not useful, compared to output like "Error, the URI address is invalid and its IP address cannot be found." All we care is that you handle exceptions and have meaningful output.

Q10: I have received some “weird” response. Why does this happen?

Ans: This can be explained with several possible reasons:

(1) Nowadays, HTTP-based attacks are prevalent. Many popular web servers use application-layer firewalls (also called web-application firewall (WAF)), which can check for tampering of cookies and invalid character encoding and much more.

(2) WAF might selectively filter out some http primitives based on its security policy defined in WAF rules.

(3) Your message encoding may not follow exactly the HTTP message format (e.g., lack \r\n, lack \r, lack \n, lack :, mis-spelling a head attribute), thus triggering the action of the WAF.

Q11: Should we print out or remove duplicate cookies. For example, for www.google.com I get 2 cookies from https and 2 from http\1.1 that seem to be identical. Should I print all of them or remove the duplicates?

Ans: This requirement is not listed in Assignment specification and will not be tested. You will not lose marks if you do not print out duplicate cookies, and you will not get extra bonus if you print out duplicate cookies.

Q12: If a return code is in the 4xx and 5xx range, do we consider that supported by that protocol?
Ans: 4xx and 5xx level are mostly caused by incorrect query message format or incorrect object names. You cannot judge whether or not a server supports http1.1 or http2 based on these error responses.

Q13: If I am checking for h2 using the selected_alpn_protocol() do we just assume from that that it will also support https since I am having to wrap the socket?

Ans: Yes. When you use wrap the socket, you will use the method *set_alpn_protocols()* to set protocols that might be used in the secured connection. When the connection is established, you can use *selected_alpn_protocol()* to check which protocols are supported.