Technical University of Moldova
Software Engineering and Automatics department
Study program in Software Engineering

WEB Programming
Laboratory Work Nr.2

Author:
Botnaru Victor FAF-202

Supervisior:
Alexei Șerșun

2023

Contents

1 Story

Imagine, you're a developer at TwoLines GmbH, a startup from Zurich, Switzerland. The idea of the startup is pretty simple - you develop best-in-class browser experience, with privacy and security in mind. One day, you receive a message from the CTO, Mike.

"' Hey there!

Tomorrow we'll have a presentation for a really biiiig investor. They say they don't need PPTs. They want at least some proof-of-concept for secure browsers for their toasters. Can you share with me the executable we used last time for demo?

Cheers, Mike "'

Yeah, the catch is toasters don't have any browser exeprience! Nonetheless, you can't remember where you've saved the executable, so you start writing it from scratch.

Yeah, the catch is toasters don't have any browser exeprience! Nonetheless, you can't remember where you've saved the executable, so you start writing it from scratch.

2 Task

Your task for this lab is:

- You have to write a command line program, using [go2web](go2web) executable as a starting point;
- The program should implement at least the following CLI: go2web -u ¡URL¿ make an HTTP request to the specified URL and print the response go2web -s ¡search-term¿ make an HTTP request to search the term using your favorite search engine and print top 10 results go2web -h show this help
- The responses from request should be human-readable (e.g. no HTML tags in the output)

3 Grading

Points:

- executable with ‘-u‘ or ‘-s‘ options - ‘+5‘ points
- executable with ‘-u‘ and ‘-s‘ options - ‘+6‘ points

  You can get ‘+1‘ extra point:

- if results/links from search engine can be accessed (using your CLI);
- for implementing HTTP request redirects

  You can get ‘+2‘ extra points:

- for implementing an HTTP cache mechanism;
- for implementing content negotiation e.g. by accepting and handling both JSON and HTML content types.

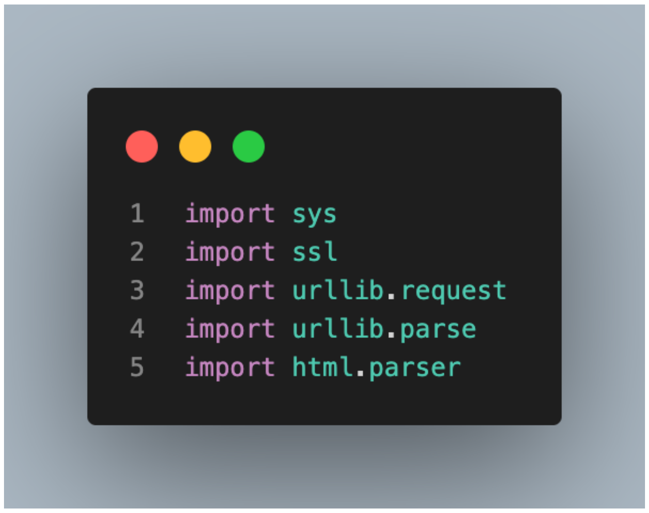
You can get ‘+2‘ bonus points for an adapted design for mobile devices.

4 Hints

- Before opting for some language, make sure you have the right tools: CLI parser, HTML/JSON parser, support for TCP sockets;
- For CLI you can use built-in libraries or even Bash built-in
- Use third-party libraries for parsing HTML and presenting it;
- While developing, you can launch a local server using ‘python3 -m http.server‘ to test debug the app.
- For HTTP cache you'll need either an in-memory store or file access.
- To avoid ‘if-else‘s in content negotiation, use your knowledge of OOP.
- ‘¡search-term¿‘ can either be a single word or all words following ‘-s‘ argument.

5 Implementation

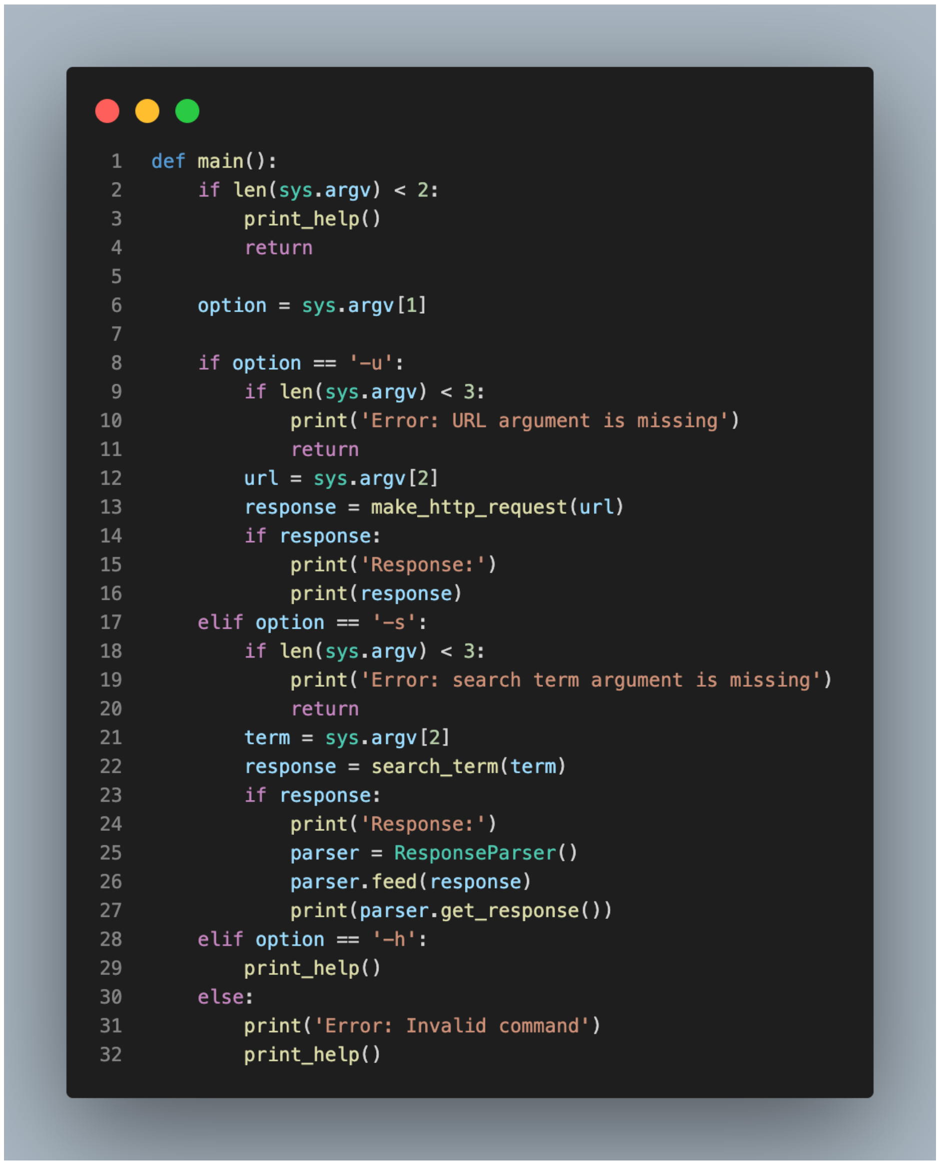
I've implemented this tasks in python. in order to implement them I've used this set of libraries:



```python
import sys
import ssl
import urllib.request
import urllib.parse
import html.parser
```

Figure 1: —

When you are writing a new request in CLI the response is prepared by a a set of functions starting with main function :



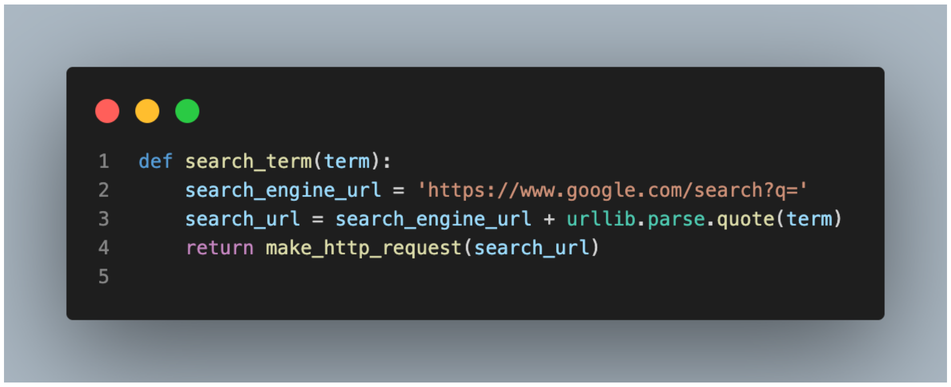
```python
def main():
    if len(sys.argv) < 2:
        print_help()
        return

    option = sys.argv[1]

    if option == '-u':
        if len(sys.argv) < 3:
            print('Error: URL argument is missing')
            return
        url = sys.argv[2]
        response = make_http_request(url)
        if response:
            print('Response:')
            print(response)
    elif option == '-s':
        if len(sys.argv) < 3:
            print('Error: search term argument is missing')
            return
        term = sys.argv[2]
        response = search_term(term)
        if response:
            print('Response:')
            parser = ResponseParser()
            parser.feed(response)
            print(parser.get_response())
    elif option == '-h':
        print_help()
    else:
        print('Error: Invalid command')
        print_help()
```

Figure 2: —

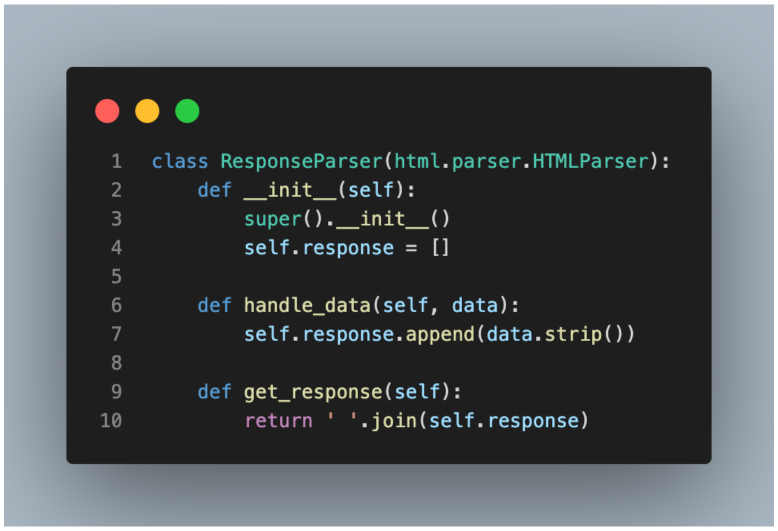
After this in dependence if request type the request is sent to a function that will execute it:



```python
def search_term(term):
    search_engine_url = 'https://www.google.com/search?q='
    search_url = search_engine_url + urllib.parse.quote(term)
    return make_http_request(search_url)

```

Figure 3: —

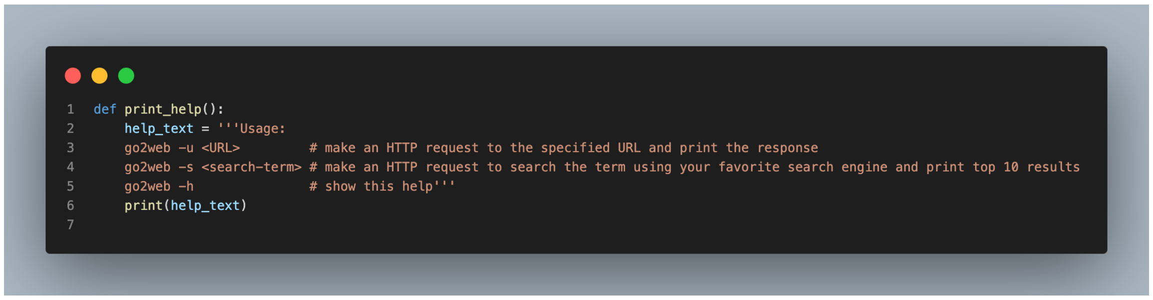


```python
class ResponseParser(html.parser.HTMLParser):
    def __init__(self):
        super().__init__()
        self.response = []

    def handle_data(self, data):
        self.response.append(data.strip())

    def get_response(self):
        return ' '.join(self.response)
```

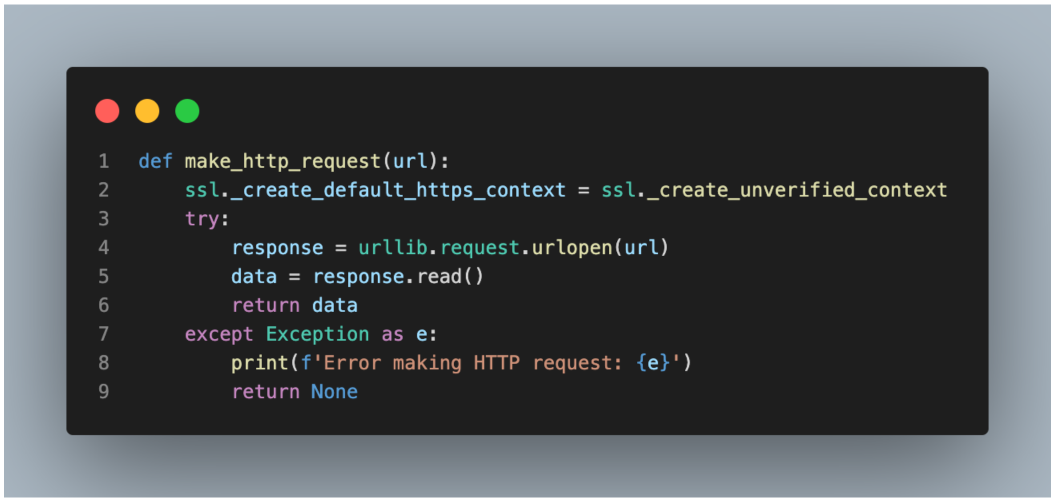
Figure 4: —



```python
def print_help():
    help_text = '''Usage:
    go2web -u <URL>         # make an HTTP request to the specified URL and print the response
    go2web -s <search-term> # make an HTTP request to search the term using your favorite search engine and print top 10 results
    go2web -h               # show this help'''
    print(help_text)

```

Figure 5: —



```python
def make_http_request(url):
    ssl._create_default_https_context = ssl._create_unverified_context
    try:
        response = urllib.request.urlopen(url)
        data = response.read()
        return data
    except Exception as e:
        print(f'Error making HTTP request: {e}')
        return None
```

Figure 6: —

6 Conclusion

In conclusion, this laboratory work focused on creating a simple command-line tool called "go2web" that allows users to make HTTP requests to URLs or search for terms using their favorite search engine. The program utilized the urllib module to handle the HTTP requests and the html.parser module to parse the response data.

The main function of the program checked the command-line arguments provided by the user and executed the corresponding functionality based on the option specified. If the user chose the "-u" option, the program made an HTTP request to the specified URL and printed the response. If the user chose the ”-s” option, the program made an HTTP request to search for the specified term and printed the top 10 results. The ”-h” option displayed the help information.

Throughout the implementation, error handling was implemented to catch exceptions and provide meaningful error messages. The SSL module was also used to handle SSL certificates for HTTPS requests.

Overall, this laboratory work provided hands-on experience with HTTP requests, URL encoding, HTML parsing, and command-line argument processing. It served as a practical introduction to web scraping and demonstrated the basic concepts of interacting with web resources programmatically.

References

GitHub Repository `https://github.com/Victoras23/PW`