Q1

A

We used Apache server, as we are ubuntu users. To install Apache server we should first update the local package index using the command:

```
sudo apt update
```

Then, install the `apache2` package:

```
sudo apt install apache2
```

To check the service is running, just run the command:

```
sudo systemctl status apache2
```

the output should be something similar to:

```
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled;
  Drop-In: /lib/systemd/system/apache2.service.d
           └─apache2-systemd.conf
   Active: active (running) since Sun 2019-04-21 20:06:23 EEST;
  Process: 935 ExecStart=/usr/sbin/apachectl start (code=exited,
 Main PID: 1017 (apache2)
    Tasks: 55 (limit: 4327)
   CGroup: /system.slice/apache2.service
           ├─1017 /usr/sbin/apache2 -k start
           ├─1018 /usr/sbin/apache2 -k start
           └─1019 /usr/sbin/apache2 -k start
```

Now the Apache Server is running.

B

We make the website, four pages with the names:

```
index.html
```

```
p1.html
```

```
p2.html
```

```
p3.html
```

Page `index.html` is the home page.
Next we place the four pages in the directory:
`/var/www/html/second/Q1/index.html`

C

To browse the website using the browser, we type these URLs in the browser:

Home page

page1

page2

page3

D

The solution for this part is in the `code.py` file.
First we import the libraries `socket` and `urllib` the server address is `lo calhost` port `80` the client connects to the server and sends `GET` request, to retrieve the home page, the request header will be:

```
GET /second/Q1/ HTTP/1.0
```

then the client receives the response, the response will be:

```
HTTP/1.1 200 OK
Date: Sun, 21 Apr 2019 20:03:20 GMT
Server: Apache/2.4.29 (Ubuntu)
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
Last-Modified: Sun, 21 Apr 2019 11:25:33 GMT
ETag: "12d-587089c32eaa9"
Accept-Ranges: bytes
Content-Length: 301
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <title>Home page</title>
  </head>
  <body>
    <h1>This is the home page.</h1>
    <p><a href="p1.html">page 1</a></p>
    <p><a href="p2.html">page 2</a></p>
    <p><a href="p3.html">page 3</a></p>
  </body>
</html>
```

And we do the same for `page1`
To retrieve the page using `urllib` we use:

```
url = 'http://localhost/second/Q1/p1.html'
page = urllib.urlopen(url).read()
```

When we print the `page` we get:

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
```

```
    <title>P1</title>
  </head>
  <body>
    <h2>This is page 1</h2>
    <p><a href="index.html">Home page</a></p>
  </body>
</html>
```

E

To extract the links in the homepage, we need to import `BeautifulSoup` from `bs4`
We used the `HTML parser` included in Python's standard library and passed it with the `page` variable to the `BeautifulSoup` constructor. then used the parser to extract the links using:

```
links = soup.find_all('a')
```

Q2

We first initialize the socket to establish the connection, when some client connects to the server he has the option to login or register, when the user registers new account first we check if there is a user with this username before, then the username and password are added to the `users.txt` file, the password is not stored as a plain text, we only store the `hash` value of the password, and compare the two `hash` values when the user attempts to login.
Next, the user chooses the test to make, there are two tests math and python test, each with five questions, and each question with four possible answers.
We used `Question` class with 6 variables:

```
class Question(object):
"""docstring for question."""
def __init__(self, question, choiceA,
    choiceB, choiceC, choiceD, correct_answer):
    # super(Qquestion, self).__init__()
    self.question = question
    self.choiceA = choiceA
    self.choiceB = choiceB
```

```
        self.choiceC = choiceC
        self.choiceD = choiceD
        self.correct_answer = correct_answer
```

Next we are serializing the `Question` object to `json` format using `__dict__` attribute.
Before sending the `json` file using socket, it is transformed to dictionary and then to `str` object. The client receives the `str` object and transforms it again to dictionary using the `eval` method.
Finally, the grade is calculated and added to the `log.txt` file, the `log.txt` file is formatted to look like a grid.