CS2505 – Python Lab 03
07.03.2019

Let us start with the **Important** stuff first:

1. The grading for these Labs will take into account the **correctness** of your solution, the **approach** taken, and **comments**, which should be clear and concise. We will be checking carefully for plagiarism and penalties will be strictly applied.
2. If you don't understand a question, please ask Yusuf or Ahmed, the lab demonstrators, who will be only happy to help.
3. All Labs will consist of some Python programming and some questions. To maximise your Continuous Assessments marks, please answers all sections.
4. The Continuous Assessment labs are worth 20% of your final year mark for this module. Thus, each of the five labs is graded out of maximum of 4 marks. To offer students an additional means of maximising their Continuous Assessments marks, each week we will also offer an additional *Coding Assignment*, from which you can receive a maximum of 1 additional mark. This extra assignment is optional and will not affect your ability to receive the maximum 4 marks for the initial part of this lab. Note: irrespective of how many questions you attempt over the five weeks, the maximum number of marks that can be received from the Continuous Assessment aspect of this module is 20 marks.
5. We do not accept solutions which are written in Python 2 (https://pythonclock.org/). **Make sure your solutions work in Python 3**.

Your solutions for this Lab, including the solutions for the additional assignment should you decide to attempt same, must be submitted on Moodle within the specified deadline. Please note that no late submission will be accepted by Moodle. If your solution files cannot run successfully, you will lose marks. So, make sure that there is no **syntax, compilation or run-time error**. You do not need to include your name or UCC ID in the name of the submitted files (Moodle does that automatically). **Follow the file naming conventions** as mentioned in the description of assignment. ☺
We recommend (but not obligate) that you follow the official style guide for Python:
https://www.python.org/dev/peps/pep-0008/
The official Python 3.7 documentation is located here: https://docs.python.org/3.7/index.html

---

Where to get Python 3.7+?
In order to run Python 3.7 on machine in class room, type python3 (or python3.7 to be 100% sure) in Kubuntu or python in Windows. You can run your script by typing "python3.7 script.py" in Kubuntu or "python script.py" in Windows.
If you use your own machine:
- for Windows OS, go to python.org and download Python 3.7 or higher;
- Ubuntu usually has the latest Python installed by default
- for older versions of Ubuntu (for 14.04, for example) you get the latest Python version from https://launchpad.net/~fkrull/+archive/ubuntu/deadsnakes ppa:

```
sudo add-apt-repository ppa:fkrull/deadsnakes
sudo apt-get update
sudo apt-get install python3.7
```

- another very good solution is usage of conda package system(https://conda.io/docs/). Go to https://conda.io/docs/install/quick.html and read manual for your OS. With conda you can quickly install a package by typing “conda install numpy” and quickly update all installed packages by typing “conda upgrade –all” (upgrade is alias for update here).

Lab 3:

In this lab, you will interact with a python web server that handles one HTTP request at a time. The web server will accept and parse the HTTP request, get the requested file from the server’s file system, create an HTTP response message consisting of the requested file preceded by header lines, and then send the response directly to the client. You will notice that if the requested file is not present in the server, the server should send an HTTP “404 Not Found” message back to the client.

Create a new project folder called “CS2505_lab3” and download the following files from the module webpage: ‘WebServer.py’ and ‘HelloWorld.html’.

Running the Server:

Run the server program using ‘python3 ./WebServer.py’’. Determine the IP address of the host that is running the server (e.g., 128.238.251.26). Open a browser and provide the corresponding URL. For example: http://128.238.251.26:6789/HelloWorld.html

‘HelloWorld.html’ is the name of the file you placed in the server directory. Note also the use of the port number after the colon. The browser should now display the contents of HelloWorld.html. If you omit the port number, i.e. ":6789", the browser will assume port 80 and you will get the web page from the server only if your server is listening at port 80. Now try to get a file that is not present at the server, e.g. http://128.238.251.26:6789/GoodbyeWorld.html. You should get a “404 Not Found” message.

In this week’s lab:

1. Instead of using a browser, write your own HTTP client to test your server. Your client will connect to the server using a TCP connection, send an HTTP request to the server, and display the server’s response as an output. You can assume that the HTTP request sent is a GET method. (Hint: Check the message sent to the server by the browser to get an idea of the message syntax that your client should use.)

   The client should take command line arguments specifying the server IP address or host name, the port at which the server is listening, and the path at which the requested object is stored at the server. The following is an input command format to run the client.

   ```
   client.py server_host server_port filename
   ```

2. Test your code both locally via your server, using ‘localhost’ and remotely via another student’s server, using their IP address.

Questions:
As there are a lot of concepts to understanding in the coding section of this lab, all I require in the question section is to explain: the problems you found and the steps you took to design this Lab.

Submit the solution files as “client_solution.py” and “WebServer_solution.py”. Submit the answers to questions as “answers.txt” file.

Additional Coding Assignment:

Currently, the web server handles only one HTTP request at a time. The additional assignment this week is to implement a multithreaded server that is capable of serving multiple requests simultaneously. Using threading, first create a main thread in which your modified server listens for clients at a fixed port. When it receives a TCP connection request from a client, it will set up the TCP connection through another port and services the client request in a separate thread. There will be a separate TCP connection in a separate thread for each request/response pair. Submit the solution file as “WebServer_solution_additional.py”.