CS2505 – Python Lab 01
22.02.2022

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 **Japneet** or **Qirtas**, 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. However, if a student receives only 3 marks from the main assessment, they can use the additional assignment to obtain one mark to achieve the maximum of 4 marks for the 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 Canvas within the specified deadline. Please note that no late submission will be accepted by Canvas. If your solution files cannot run successfully, you will lose marks. So, make sure that there is no **syntax or run-time error**. You do not need to include your name or UCC ID in the name of the submitted files (Canvas recognises you by your account). **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 in Ubuntu or python in Windows. You can run your script by typing "python3 script.py" in Ubuntu or "python script.py" in Windows.
If you use your own machine (if you do not have Python 3 installed):
- 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
```

- MacOS comes with Python 2 pre-installed, so you will need to go to python.org to download Python 3 and install it.
- another very good solution is usage of the *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 1:

In this Lab you will learn the basic of python Socket Programming.

Go to the Laboratory Exercises on the classes Canvas site and download the files: client.py and server.py, save them in a folder called "CS2505_lab1".

This lab is a very simple example of a python TCP socket between a server and client. There is no exception control and minimal code.

Running the Code:

Without really reading what the code does, let's look at how we run the code:

1. The Server must always run first, so open a terminal/cmd window and move to the "CS2505_lab1" folder and type 'python ./server.py' (python3 on Ubuntu) or and press the return button. This should open an Output window at the end of the screen, on which you should see:

```
 *** Server is starting up on localhost port 10000 ***
*** Waiting for a connection ***
```

   You should also note that at the bottom of the screen, the program has not stopped, but is still running. This is fine for the Server, as its job is to wait for connections from Clients. To stop the server program, we need to press both the "ctrl" and "c" buttons.

2. Next open a second terminal window in the same folder location and type 'python ./client.py' and press the return button.

   On the terminal running the client file you should see:

```
connecting to server at localhost port 10000
sending "This is the message.  It will be repeated."
received "This is the mess"
received "age.  It will be"
received " repeated."
closing socket
```

   While on the terminal running the server file you should see:

```
*** Server is starting up on localhost port 10000 ***
*** Waiting for a connection ***
connection from ('127.0.0.1', 53283)
received "This is the mess"
sending data back to the client
received "age.  It will be"
sending data back to the client
received " repeated."
sending data back to the client
received ""
no more data from ('127.0.0.1', 53283)
*** Waiting for a connection ***
```

Note:
The client program has ended, while the server program is still waiting for connections.

4. You have now built and run your first python Sockets program. Take some time to read the lines of code and their comments within the Client and Server files. Next change the texts being passed between the Client and Server, see how it works.

It's time to do your own work now, so complete the following and submit on Canvas within the deadline. Name the files as client_solution.py and server_solution.py.

1. Update the Client and Server code, so rather than hardcoding "localhost" as the HostName, determine the domain name of the machine and the IP of the machine and print that out to the terminal. Look at the methods defined in the socket API in https://docs.python.org/3.7/library/socket.html to help you.
2. Update the Client code, so that the input sentence being sent to the Server is not static, but dynamic. Update the code to read from the command line, i.e. the terminal, and send the input sentence to the server.
3. Update the Server code, so that the input sentence from the client is saved to a file along with the time and date it was received, thus creating a form of log file. Update the sentence being returned to the client, so it informs the client the time the Server logged the sentence it sent- as a form of acknowledgement. The log file will be located in the same folder as the source code on your machine's hard-drive. There is no need to submit the log file.

Additional Coding Assignment (handling of multiple clients simultaneously):

The additional assignment this week, should you attempt it, is to take enhance the client and server code to allow the server to handle multiple connections. Name the files of your additional solution as client_solution_additional.py and server_solution_additional.py and when finished also upload on Canvas.

a. Increase the number of connections the server will accept to two or more and for each connection print out the client details, i.e. “connection from client port <port> and host name <host>”.

b. Add exception control to both the Client and Server code.

c. Implement functionality of serving multiple connections simultaneously by the server. How to check if the server can do it? Run the server, then run the client1 (it should ask for a message), without typing anything into client1 run client2 and use client2 in order to send a message. If server can receive message from the client2 while waiting something from the client1 then server can handle many connections simultaneously.

There are many ways in the modern Python 3.7+ to implement a server which can handle multiple clients simultaneously. Last year all students had been using only one thread. So, we decided to describe others here. We encourage you to try the async/await way (added in September 2015 in Python 3.5) or any other. However, you can use threads if you want to.

Among possible ways are

1) Using Thread (https://docs.python.org/3.7/library/threading.html) or Process (https://docs.python.org/3.7/library/multiprocessing.html).

2) Using asyncio module (it was added in the standard library of Python 3.4, https://docs.python.org/3.4/library/asyncio.html)

a) It is possible to use generators (watch this nice talk about generators: https://www.youtube.com/watch?v=D1twn9kLmYg) in order to construct server capable of serving many connections. It is not the easiest way, but you may want to try. Very good explanation is given in the 12 chapter (Concurrency) of the book Python Cookbook by David Beazley.

b) Protocol class can be used (from asyncio module), which is supposed to help in developing of network protocols. You can find documentation here: https://docs.python.org/3.4/library/asyncio-protocol.html#protocol-examples.
Here (https://www.youtube.com/watch?v=E-1Y4kSsAFc) is a great talk where you can find even more details.

3) Another method is async/await technique (added in standard library of Python 3.5, see:
https://docs.python.org/3.5/library/asyncio-task.html and
https://docs.python.org/3.5/library/asyncio-eventloop.html and
https://www.python.org/dev/peps/pep-0492/.
Here (https://www.youtube.com/watch?v=ZzfHjytDceU) is a very nice video with explanations of how it is working. You may find useful these talks about async/await as well:
https://www.youtube.com/watch?v=m28fiN9y_r8 and
https://www.youtube.com/watch?v=WSq0S7UvI8E.