Create an IRC Server and Client

Submit Assignment

Due Oct 19 by 11:59pm **Points** 75 **Submitting** a file upload

In the second project, you'll be creating a stripped down version of an IRC (Internet relay chat) server and client. In IRC, clients communicate with each other through a distributed network of servers. Clients do not directly communicate to any other client, but route all communication through the servers. Servers are also distributed, such that clients can connect to different servers, yet still talk to each other, so long as the servers have established a connection with each other. Specifically, servers are connected in a spanning tree, such that no cycles are possible in the network. Each server creates its own spanning tree, where it lives at the root of the tree

In addition to defining the network architecture, the IRC protocol also defines a format procotol for messages, and a number of different messages governing interactions between clients and servers. We will be implementing a stripped down version of this protocol with 8 supported messages (at the server level) and 6 messages (at the client level). In addition to messages, IRC also defines numeric replies, which are similar to messages except that they are generated in response to a message, rather than initiating an action. We will be handling 12 numeric replies at the server and client levels (these are much simpler than messages).

The full IRC protocol definitions can be accessed here:

- General Protocol: https://tools.ietf.org/html/rfc1459. (https://tools.ietf.org/html/rfc1459)
- Architecture Protocol: https://tools.ietf.org/html/rfc2810 (https://tools.ietf.org/html/rfc2810)
- Server Protocol: https://tools.ietf.org/html/rfc2813 (https://tools.ietf.org/html/rfc2813)
- Client Protocol: https://tools.ietf.org/html/rfc2812 (https://tools.ietf.org/html/rfc2812)

I'm providing you with three python files, and a folder containing test cases.

- IRCNetworkLauncher.py
  - IRCNetworkLauncher is what you'll use to test your project. It contains a testing framework I've written to help you with testing and debugging your code. You can edit which tests are run by commenting out unwanted tests in the "tests" dictionary included at the end of the file. You can also write your own tests if desired and add them here, but this is not required. Do not edit IRCNetworkLauncher except for adding or removing test cases. We will test your code using a fresh copy of IRCNetworkLauncher, so no edits you make will be maintained.
  - When you run the IRCNetworkLauncher, you'll get a printout for each test indicating whether it passed (True), or failed (False). If the test fails, you'll also be given details about what went wrong. The tester operates by evaluating the final state of your network, including for the server: the list of adjacent users, all users, adjacent servers, all servers, and the channels (including their details). For clients, the tester looks at the channels the client has registered for and the messages the client has received from the server. The tester will inform you if an expected value is missing, and if an unexpected value is present.
- IRCServer.py
  - The bulk of what you need to implement is contained within IRCServer. This file is thoroughly documented with comments, and you should refer primarily to these comments for the details of how your code should

function. For your reference, my implementation of the server is 1326 lines of code (including comments), and the template contains 733 lines of code.

- IRCClient.py
    - You will also need to implement a stripped down IRCClient. The primary function of the IRCClient is to create and send messages to the server, and to receive responses and update the client's known state of the network based on responses by the IRCServer. For your reference, my implementation of the client is 439 lines of code (including commens), and the template is 363 lines of code.
    - You are not required to implement a client that accepts user input from a console, or that prints messages to a console, as all testing will be done using the IRCNetworkLauncher. If you want to implement this functionality, you are welcome to do so, and there are a few stubbed out functions that can be used for this purpose.

Your IRCServer will need to serve multiple connections at the same time. To accomplish this, **you MUST use a Selector**, as discussed in the slides Python for Network Programming - Select Statements and Delimiters (https://docs.google.com/presentation/d/1oLjS4sudpR5AVh4MI73tTu-UsTTwbZiTrfPMb1EaXJ8/edit?usp=sharing) . You will not need to use a Selector for the client, as it only needs to serve a single connection simultaneously. (IF you choose to implement an IRCClient that accepts input from the console, then you will likely need to introduce another form of parallelization, so as to support reading from the socket and from the client terminal simultaneously. This is NOT required for this project.)

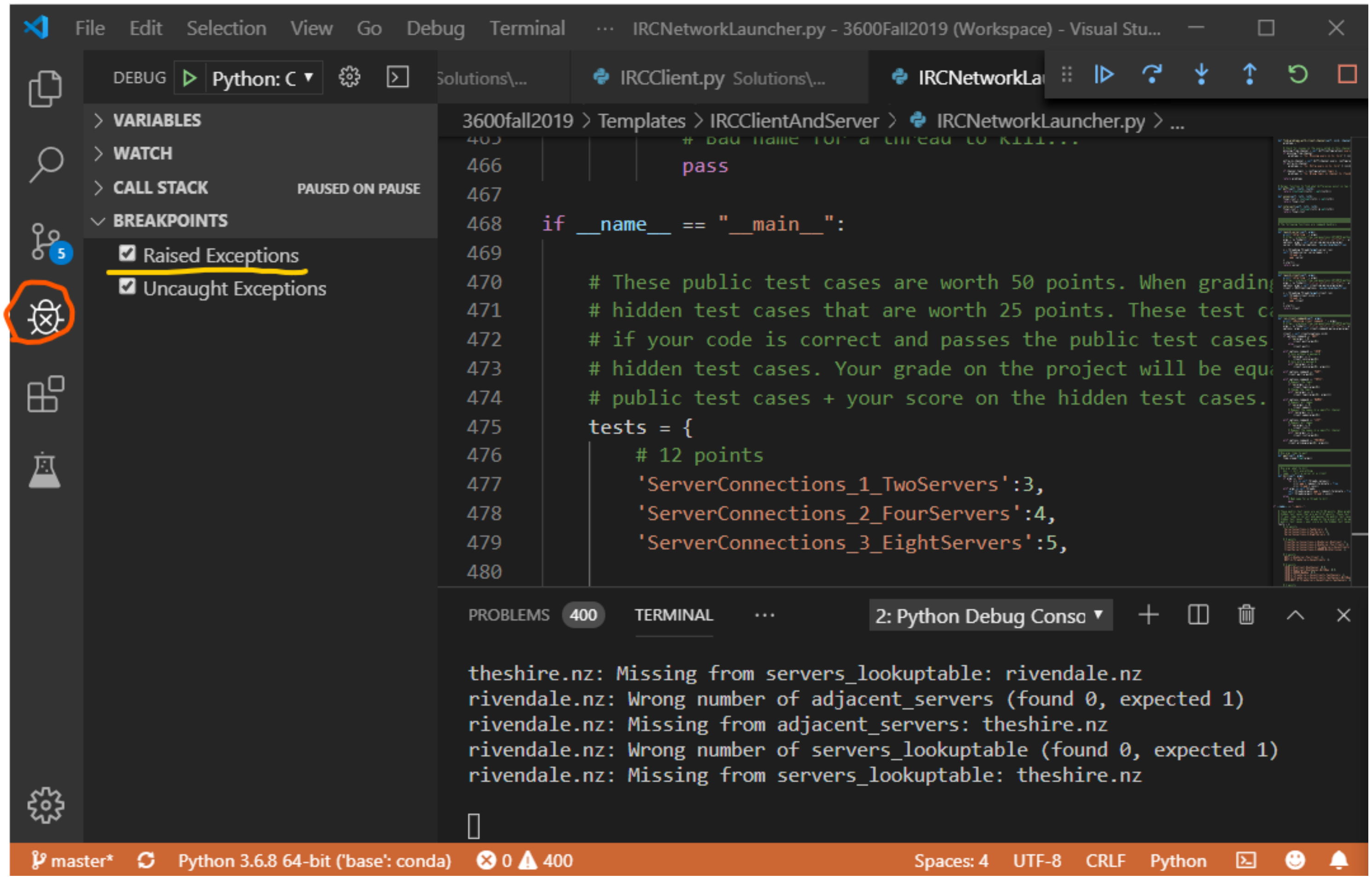
One of the main tasks you will need to accomplish for the IRCServer is determining WHO a message is from: is this a message from a client? a server? a new program requesting a connection, and you don't know if it's a client or a server? Is this a message coming from an adjacent client? Or is this a message that originated with a distant client and is being passed on by another server? Most of these questions can be answered by examining the syntax of the IRC message (hint: the prefix will be very helpful for this), but you'll also need to take advantage of your ability to associate data with a socket when registering it with your selector.

Another major task you will need to accomplish for the IRCServer is determining how to forward a message so that it reaches its intended destination. You will need to determine not only WHO the intended recipient of the message is, but also how to get a message there. If the intended recipient is not adjacent to a server, then the server needs to determine which adjacent server will eventually connect to the intended recipient. You will need to capture and store this information when users and servers register with a server so as to use it later. Take advantage of how IRC networks are structured as spanning trees. This means that if you receive a message from UserX over LinkB, then you must send a message into LinkB to get it to UserX.

The last major task you will need to accomplish is determining how to correctly parse IRC messages. With optional prefixes, variable numbers of parameters, and an optional trailing parameter, parsing IRC messages is more complex that simply splitting by a delimiter. If you're familiar with regular expressions, you're welcome to use them to split the message into its various components. Otherwise, you'll need to carefully think through the logic of how to determine when and where the message needs to be split.

Helpful Debugging Information

The IRCNetworkLauncher catches all exceptions by default as one check to see if your test passed. This can make debugging your code more difficult. If you are developing in Visual Studio Code, you can configure it to break on ALL exceptions, even ones that are caught later in execution. To do so, make sure Raised Exceptions is checked in the Breakpoints pane (this pane appears when you enter Debug mode, or when you select the icon circled in red)


File Edit Selection View Go Debug Terminal ··· IRCNetworkLauncher.py - 3600Fall2019 (Workspace) - Visual Stu...
DEBUG Python: C
Solutions\...
IRCClient.py Solutions\...
IRCNetworkLa
3600fall2019 > Templates > IRCClientAndServer > IRCNetworkLauncher.py > ...
VARIABLES
WATCH
CALL STACK PAUSED ON PAUSE
BREAKPOINTS
☑ Raised Exceptions
☑ Uncaught Exceptions
PROBLEMS 400
TERMINAL
2: Python Debug Conso
master* Python 3.6.8 64-bit ('base': conda) 0 400 Spaces: 4 UTF-8 CRLF Python

```
            pass

if __name__ == "__main__":

    # These public test cases are worth 50 points. When grading
    # hidden test cases that are worth 25 points. These test ca
    # if your code is correct and passes the public test cases
    # hidden test cases. Your grade on the project will be equa
    # public test cases + your score on the hidden test cases.
    tests = {
        # 12 points
        'ServerConnections_1_TwoServers':3,
        'ServerConnections_2_FourServers':4,
        'ServerConnections_3_EightServers':5,
```

```
theshire.nz: Missing from servers_lookuptable: rivendale.nz
rivendale.nz: Wrong number of adjacent_servers (found 0, expected 1)
rivendale.nz: Missing from adjacent_servers: theshire.nz
rivendale.nz: Wrong number of servers_lookuptable (found 0, expected 1)
rivendale.nz: Missing from servers_lookuptable: theshire.nz
```