**Project Report #1: Email and Web Scraping with Python**

Daniel Hanna

Computer Networking (CSC 360)

The College of New Jersey

Department of Computer Science



**Part 0: Obtaining Familiarity with SMTP**

In attempting to understand how mail servers work, nslookup was used to determine which mail servers were used by TCNJ. It is clear from this command that there are two mail servers being used by TCNJ: mxgate1 and mxgate2. However, when attempting to utilize telnet in order to connect to either of these servers, the attempt times out, claiming that the resource is temporarily “unavailable.” However, this is expected, since as explained in the project outline, since the current computer’s host name does not match any registered host name on the DNS servers used by the college.

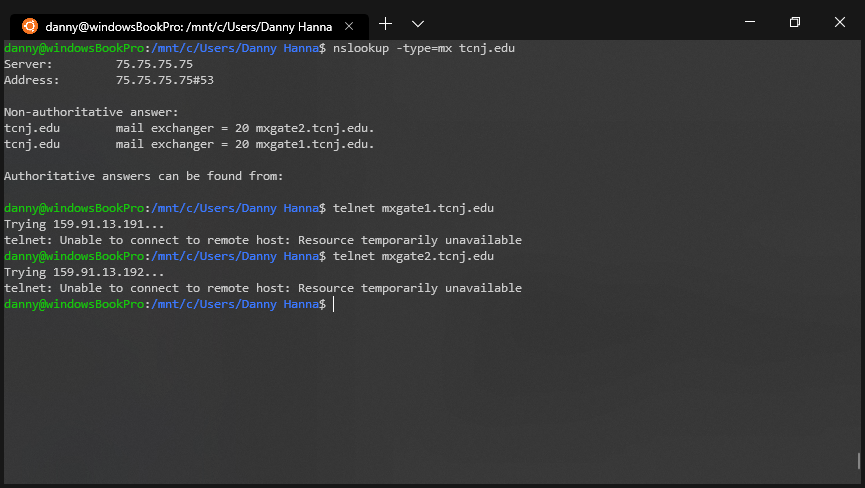


Figure 1: Using nslookup and telnet to access TCNJ’s mail servers

When attempting similar actions with Gmail’s servers, it can be seen that several mail servers exist, some of which are IMAP servers and some of which are SMTP servers. According to Google’s support documentation, Gmail’s SMTP servers use port number 587 for TLS (encrypted) protocol. Thus, openSSL must be utilized, as opposed to Telnet’s plaintext protocol.

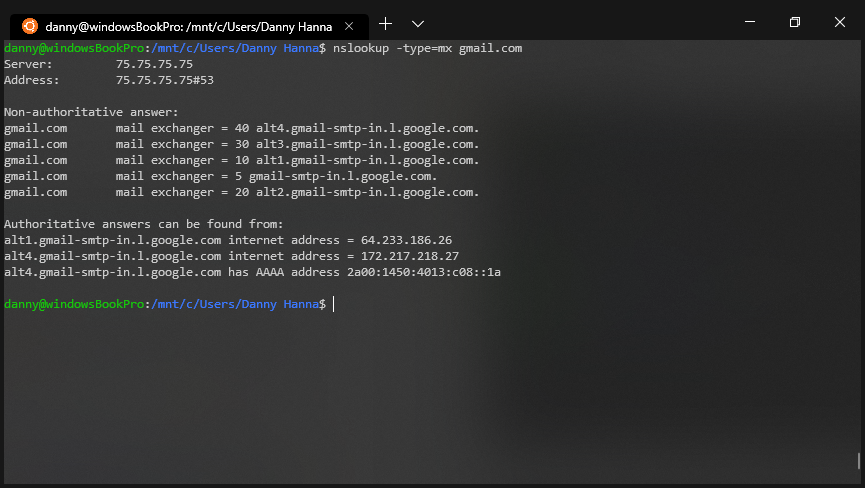


Figure 2: Using nslookup for Gmail's servers

In order to effectively connect to the SMTP mail server, a valid credential must be utilized. This can be done by Perl’s MIME, which encodes and decodes strings to and from a base64 encoding. This will provide an “authentication token” for connection. Figure 3 demonstrates the use of this command and the resulting output. Please note that some of the command has been covered for security.

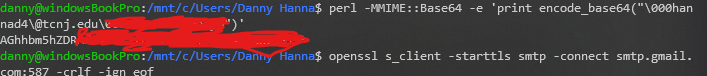


Figure 3: Perl MMIME to create a base64 encoding using login credentials

Connecting to the mail server and sending some mail then becomes a simple matter. OpenSSL is first used to connect to port 587 of the SMTP mail server.

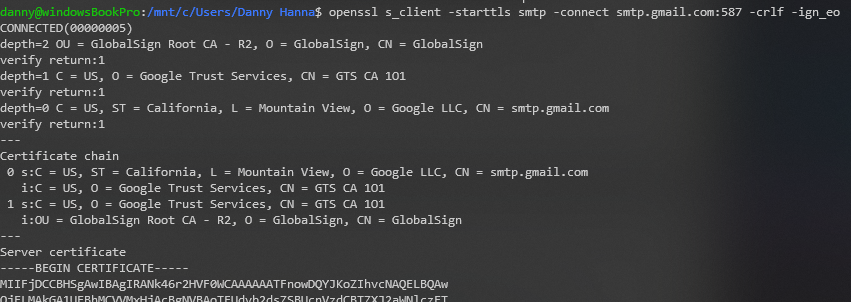


Figure 4: SSLing into server

Then authentication is performed by using the previously created base64 encoded sequence of characters. Note that for the purpose of security, the full character sequence is not included in the below figure.

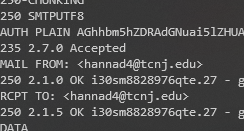


Figure 5: Authentication is Accepted

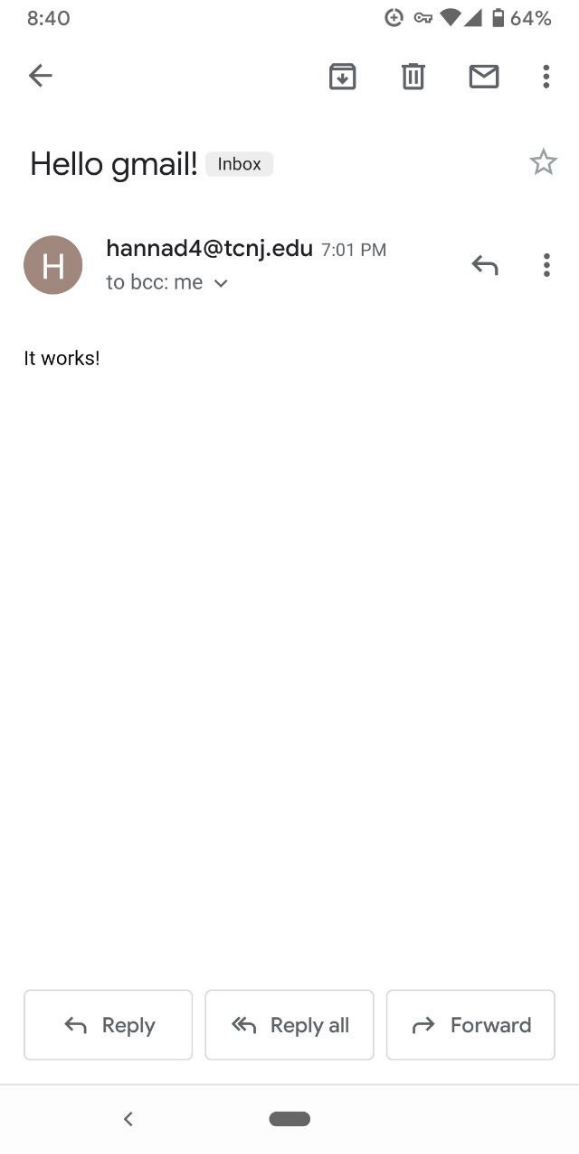
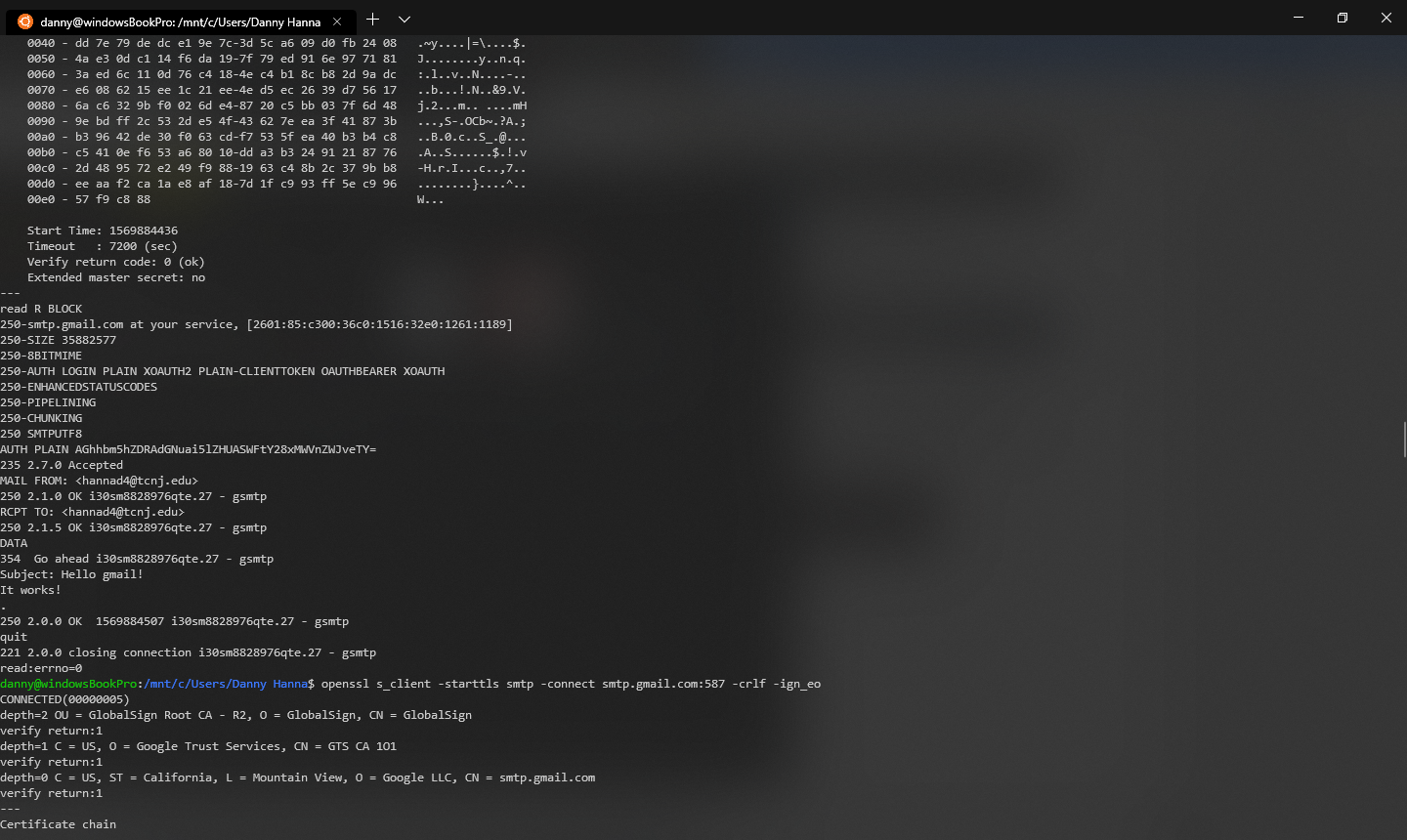
 It is then possible to send mail through the connection. Note the mail sent on the left half of the below figure, and the received mail on the right half.

Figure 6: Mail sent through SSL connection is received successfully

**Part 1: Extracting Stock Market Information**

As instructed via class and Canvas announcement, this portion of the project will be skipped since Yahoo Finance has stopped behaving as expected. Part 2 will continue the project process.

**Part 2: Scripting for Use With PAWS**

The goal of the second portion of the project is to automate the task of searching for classes in PAWS. This was done using Selenium and BeautifulSoup.

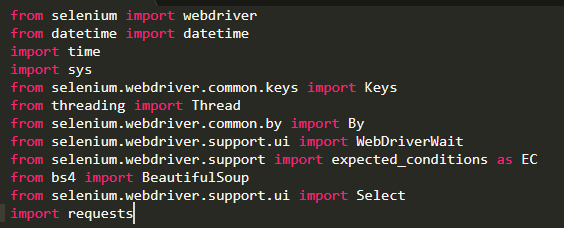


Figure 7: Imports which Include Selenium and BeautifulSoup

The pre-provided code acts as a good jumping off point by handling the steps that are needed to allow for a user to log into PAWS automatically. The only thing that needed to be modified with this code is the username and password fields that the script will use to successfully log the user in.

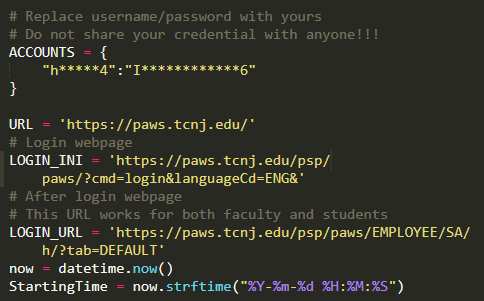


Figure 8: Changing Username and Password

It should be noted that the original starting time was a hard-coded date. In order to make this dynamically change, the datetime library was imported and utilized.

After the initial login, the next step required was to navigate to the Class Search page on PAWS and to enter the correct search terms so that the desired results may be obtained. In order to do this, the student center link was found and clicked first.

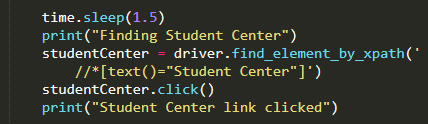
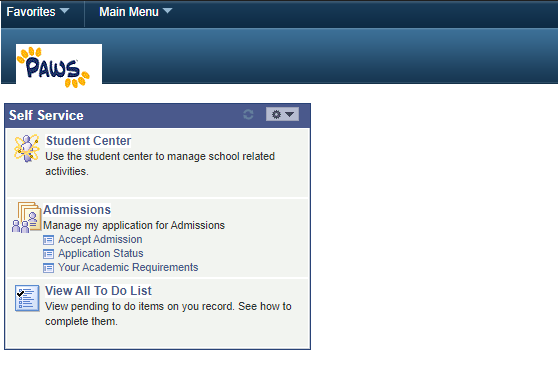
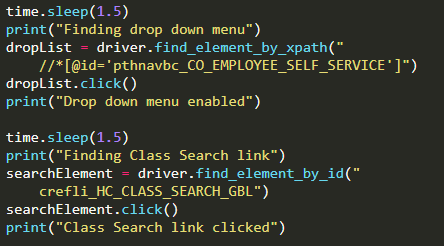


Figure 9: Clicking on Student Center

Once Student Center has been clicked, the next step is to navigate to the class search page. There are several links on the page that can do this. However, one of the links can be accessed through a drop-down menu. This element can be found via xpath without having to switch frames.

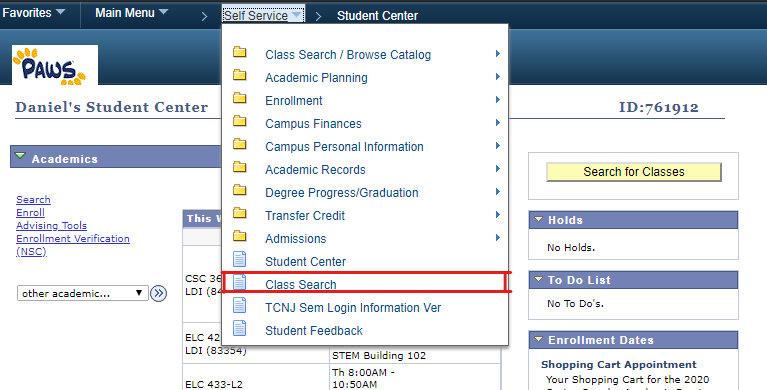


Figure 10: Clicking on Class Search

Note that a pause time of 1.5 seconds is periodically included between find element commands. This is to give PAWS time to load the page in so that the element may actually be found. When using a time of only 1 second, some runs of the script resulted in an error, since that was not enough time for everything to load in.

Once on the class search page, it is now necessary to enter the correct search terms. This includes selecting the correct semester as well as entering the class code. Unchecking the “Show Open Classes Only” must also be done. All of this can only be done after switching the correct frame. This must be done because all of the page content in PAWS is nested in an iframe tag. See Figure on next page.

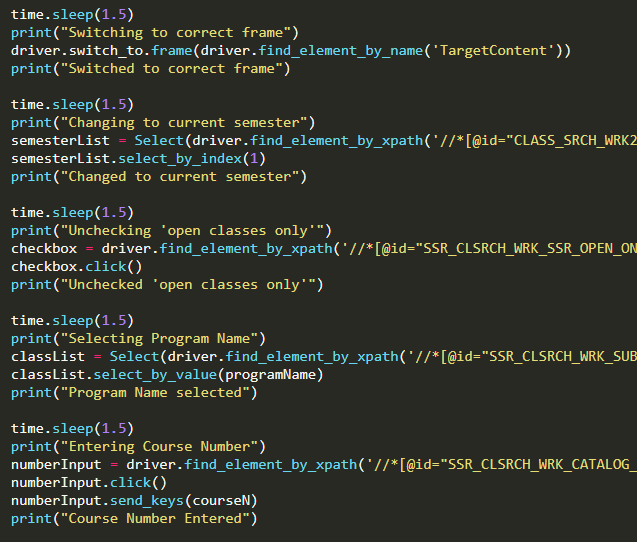
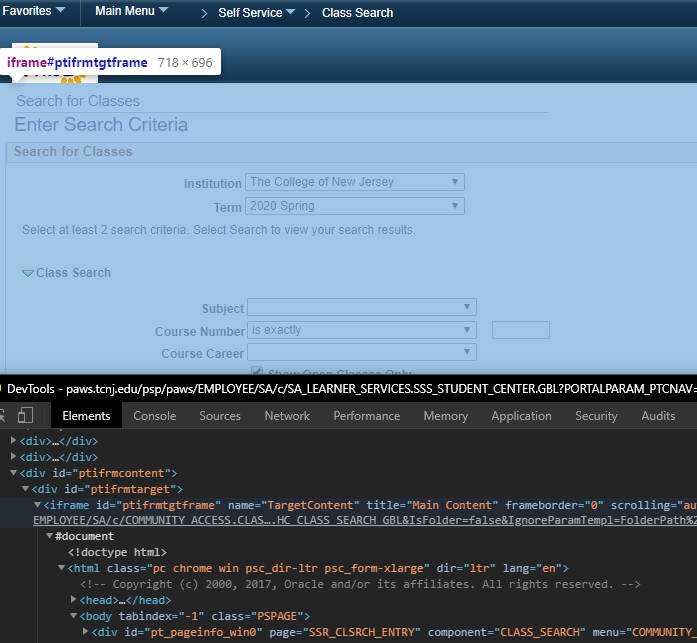


Figure 11: Changing to the Correct Frame

After the correct terms are selected and the search is submitted, the resulting page’s HTML content needs to be collected in order to print the results to the console. This is done using BeautifulSoup. There are 4 main pieces of information that is needed from the HTML content: if the class is open or closed, what the class code is, the hours in the week that the class will be, and the class’ teacher. Thus, four arrays are created to hold this information.

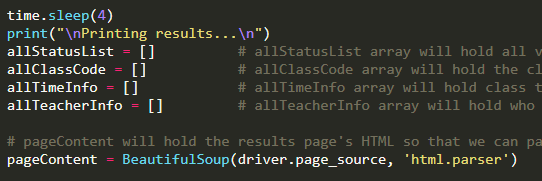


Figure 12: Using BeautifulSoup and Preparing Some Arrays

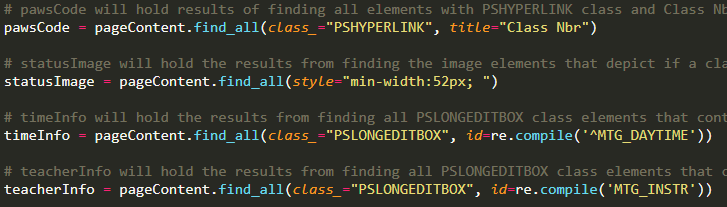
Now the HTML content must be sifted through. Some of HTML elements’ attributes are used as search criteria so that BeautifulSoup may find all the elements that are desired. These criteria include element class, title, style, and id. Style is particularly useful for finding whether a class is open or closed, since this is depicted via small images on the page. It is important to note that class time info for each section will have the same id concatenated with “$#” where # is a number. For example, the first class time information may have an id of “MTG\_DAYTIME$0” and the second class may have an id of “MTG\_DAYTIME$1”. For this reason, re.compile() is used to find all elements that contain “MTG\_DAYTIME” as part of the id. Similarly, re.compile() was also used to find all elements with an id containing “MTG\_INSTR”.

Figure 12: Using Element Attributes as Search Criteria

It is beneficial to extract the ‘alt’ attribute from the images that depict if classes are open or closed, because the alt attribute is marked either “Open” for an open class, or “Closed” for a closed class. This is done in the next figure.

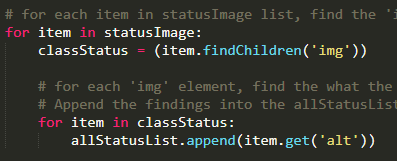


Figure 13: Getting the ‘alt’ Attribute

So far, all of the search criteria (except for the images) has returned a list of elements. However, for the purpose of this project, it is the text of these elements that is desired. Thus, for each list, the text can be extracted from the elements and appended to their respective arrays.

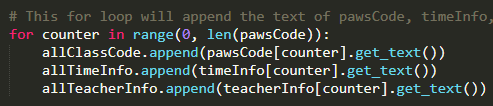


Figure 14: Appending Text to Arrays

Finally, it is now possible to output the results of the search to the user. The first condition to account for is if there is no class that matches the users search criteria. This can be done by checking if the allStatusList array is empty or not. This action is done in the next figure.

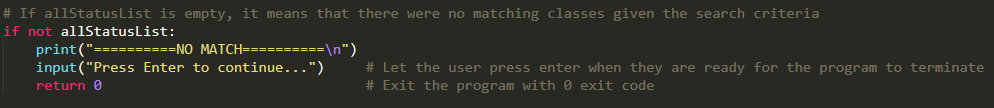


Figure 15: Accounting for No Match

If there is a match, Open Classes can be printed first, followed by closed classes. This can be done by iterating through the outputted status list twice – once for open array items, and again for closed ones. When doing so each time, the class code, class time, and teacher’s name(s) can be outputted, along with a little separator.

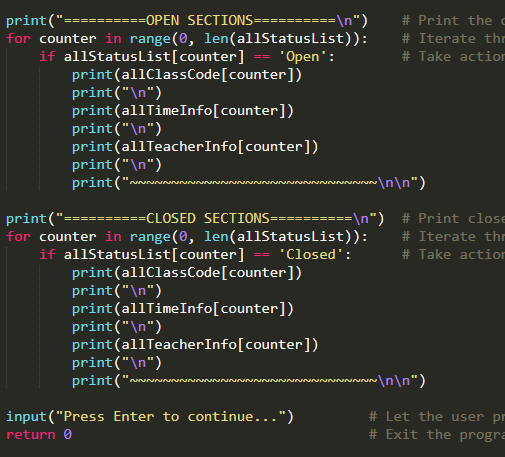
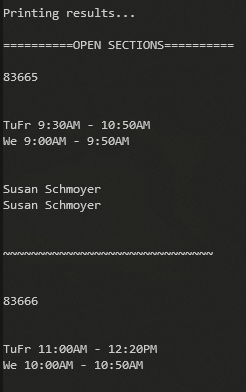
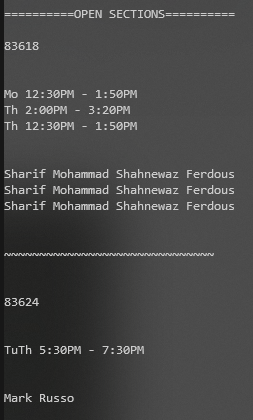


Figure 16: Printing Open and Closed Sections

Testing shows great results. See the following figures on the next page.



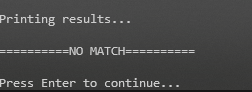


Figure 17: Entering “mat 127” for results on left. Entering “csc 220” for results on right. Entering “csc 220000” for bottom.

Error handling was also added for the condition in which the user enters a class code that does not exist. See below figure.

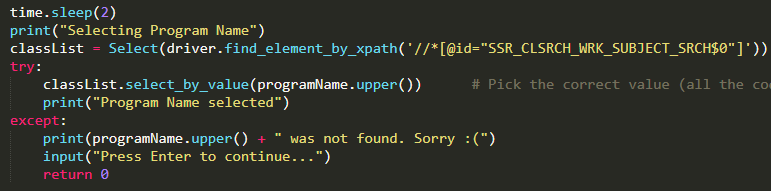




Figure 17-18: Exception Handling for Bogus Class Inputs (Such as “matewoifwoiefjwo 127”)