GitHub Repository: <https://github.com/jaundickey/IS452FinalProject_Fall2018>

1. **Initialization and Research:**
   1. ***The tutorial for creating a GitHub*** found in the [IS 452 Fall 2018 Final Project Guidelines](https://courses.ischool.illinois.edu/pluginfile.php/369140/mod_assign/introattachment/0/FinalExamIS452-Fall18.pdf?forcedownload=1) was followed to create a public repository under my user account, jaundickey.
   2. ***Research for creating a connection to Gmail using Python programming*** code was performed and resulted in directions and boilerplate code obtained from the following:
      1. [*Read Gmail using Python*](https://pythonprogramminglanguage.com/read-gmail-using-python/), hosted by [pythonprogramminglanguage.com](https://pythonprogramminglanguage.com/).

This site was the initial starting place. However, after much trial and error it appears that the boilerplate code presented is out of date (i.e. Python did not recognize “string.split()” and “rfc.822.message” as further research stated these references were discontinues with the string module now being inherent, and the RFC822 components being part of the bundle email package import. Having said that, it is very likely that the initial instructions followed in the first few steps (i.e. “enable POP support in Gmail” which provides instructions to enable IMAP in your Gmail account and then activate the setting to Access for less secure apps) allowed subsequent attempts with updated code to be successful, therefore the reference to this page remains to ensure reproducibility.

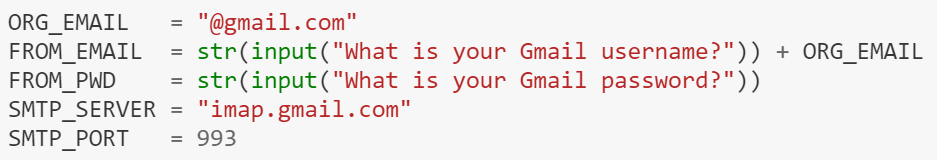
* + 1. [*How to Read Email from Gmail using Python*](https://codehandbook.org/how-to-read-email-from-gmail-using-python/), hosted by [codehandbook.org](https://codehandbook.org/how-to-read-email-from-gmail-using-python/).

Leveraged boilerplate code for the utility to read email, the named function reademail(), the variable names and the structure for defining the email IDs from the list of emails

* + 1. [*How to Read Emails using IMAP Download Attachments Python 3 for Beginners 2018*](https://www.youtube.com/watch?v=e-OZeAHFpkw) *and* [*Day 30: Fetch & Convert Email in Python Programming*](https://www.youtube.com/watch?v=bbPwv0TP2UQ), hosted by [YouTube.com](https://www.youtube.com/).

After initial unsuccessful attempts were made to generate proper server responses and email header or body content from the above boilerplate code it was determined further in-depth, step-by-step instruction was required to bring additional context for the functions and arguments used. Thus, a search for audio and visual capabilities of from scratch coding was needed.

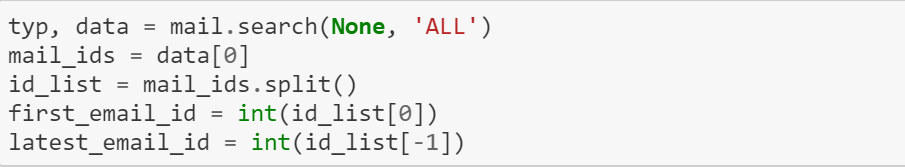
1. **Authoring Code:**
   1. ***The connection to Gmail*** employed boilerplate code from 1.2.2 and 1.2.3 above were used to successfully connect Python to the Gmail server and obtain more information about the mail objects (see the Connecting to the Gmail Server And Obtaining Inbox Index Information section of the [Gmail Inbox Analysis Jupyter Notebook](https://github.com/jaundickey/IS452FinalProject_Fall2018/blob/master/Gmail%20Inbox%20Analysis.ipynb) for detailed information).
      1. ***Methods Used (1)***
2. Boilerplate code: using the above resources provided a general base of code blocks to employ. However, there was not a one-stop resource that walked through how to successfully accomplish this to obtain inbox email header information. The code following code blocks were used multiple times across the resources and were found to be most successful in connecting to the Gmail server.





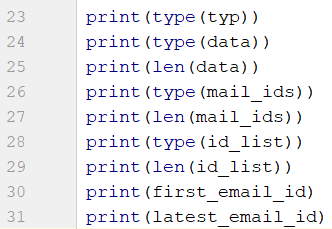
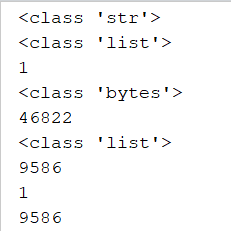
* + 1. ***Lessons Learned (2)***

1. Outdated resources: It was found that many of the resources were outdated and that Python has been updated within the last few versions to create more enhanced email object functions and methods that are inherent to the language; previously many of these had to be imported separately
2. Enable Gmail settings: the “enable POP support in Gmail” which provides instructions to enable IMAP in your Gmail account and activation of the setting to “Access for less secure apps” must be configured to allow Python to obtain server information
   * 1. ***Outcomes (1)***
3. Code run with 0 errors: After much trial and error of running the code above and receiving errors that state “please use browser to login” or mail.[method] doesn’t have attribute [.method] the block of code ran successfully without errors, there was no output, however validation was received in the following.
   1. ***Parsing and defining header information*** was accomplished again by employing boilerplate code from 1.2.2 above and converting the indexes numbers from parsed bytes to integers (see the Connecting to the Gmail Server And Obtaining Inbox Index Information section of the [Gmail Inbox Analysis Jupyter Notebook](https://github.com/jaundickey/IS452FinalProject_Fall2018/blob/master/Gmail%20Inbox%20Analysis.ipynb) for detailed information).
      1. ***Methods Used (2)***
4. Boilerplate code: using the above resources provided a general base of code blocks to employ. However, additional understanding of the variables was needed to further construct the analysis appropriately due to the use of appropriate operands because of the outdated code in the resources.
5. Type conversion: the type conversion to integer is needed here to convert the index numbers from the latter half of a parsed byte to an integer number for later processing during the analysis code block.



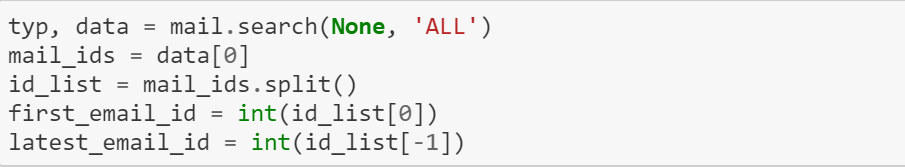
* + 1. ***Lessons Learned (2)***

1. Email directory information is done in bytes: even though the data variable had a length of 1, it contained a single element of 46K bytes.
2. Foreign bytes class: the bytes class attributes and architecture was not known and presented challenges when determining when to use the variables as bytes, when to convert them to integers, or even that the split() function could be used. Even though this was boilerplate code from 1.2.2 the rest of the boilerplate code using for loops to iterate through the emails did not work for me. Therefore, I had to investigate what bytes were and how they were handled (and not handled in Python).
   * 1. ***Outcomes***
3. First email ID: The code block parsed the mail id list to provide the first email in the Inbox (exemplified below as 1)
4. Latest email ID: The code block parsed the mail id list to provide the latest email in the Inbox (exemplified below as 9,586)

* 1. ***Obtaining user input for analysis and creating result variables*** was accomplished by creating variables with Python’s inherent features of assignment and input (see the Analyzing Inbox Index Information section of the [Gmail Inbox Analysis Jupyter Notebook](https://github.com/jaundickey/IS452FinalProject_Fall2018/blob/master/Gmail%20Inbox%20Analysis.ipynb) for detailed information).
     1. ***Methods Used (3)***

1. Input from user: the input function is used to collect data from the user for the number of emails and days back to be analyzed. In other words, starting with the latest email received and subtracting the number input from the user. The same is true for days back, starting with today and subtracting the number input from the user.
2. Type conversion: the type conversion to integer is needed here to convert the index numbers from the latter half of a parsed byte to an integer number for later processing during the analysis code block.
3. Date/time references and deltas: the type conversion to integer is needed here to convert the index numbers from the latter half of a parsed byte to an integer number for later processing during the analysis code block.



* + 1. ***Lessons Learned (2)***

1. Email directory information is done in bytes: even though the data variable had a length of 1, it contained a single element of 46K bytes.
2. Foreign bytes class: the bytes class attributes and architecture was not known and presented challenges when determining when to use the variables as bytes, when to convert them to integers, or even that the split() function could be used. Even though this was boilerplate code from 1.2.2 the rest of the boilerplate code using for loops to iterate through the emails did not work for me. Therefore, I had to investigate what bytes were and how they were handled (and not handled in Python).
   * 1. ***Outcomes***
3. First email ID: The code block parsed the mail id list to provide the first email in the Inbox (exemplified below as 1)
4. Latest email ID: The code block parsed the mail id list to provide the latest email in the Inbox (exemplified below as 9,586)

