[1. Initialization and Research: 2](#_Toc532993260)

[1.1. The tutorial for creating a GitHub 2](#_Toc532993261)

[1.2. Research for creating a connection to Gmail using Python programming 2](#_Toc532993262)

[1.2.1. Read Gmail using Python 2](#_Toc532993263)

[1.2.2. How to Read Email from Gmail using Python 2](#_Toc532993264)

[1.2.3. Day 30: Fetch & Convert Email in Python Programming 2](#_Toc532993265)

[2. Authoring Code: 2](#_Toc532993266)

[2.1. The connection to Gmail 2](#_Toc532993267)

[2.1.1. Methods Used (1) 2](#_Toc532993268)

[2.1.2. Lessons Learned (2) 2](#_Toc532993269)

[2.1.3. Outcomes (1) 3](#_Toc532993270)

[2.2. Parsing and defining header information 3](#_Toc532993271)

[2.2.1. Methods Used (2) 3](#_Toc532993272)

[2.2.2. Lessons Learned (2) 3](#_Toc532993273)

[2.2.3. Outcomes (2) 3](#_Toc532993274)

[2.3. Obtaining user input for analysis and creating result variables 4](#_Toc532993275)

[2.3.1. Methods Used (4) 4](#_Toc532993276)

[2.3.2. Lessons Learned (2) 4](#_Toc532993277)

[2.3.3. Outcomes (1) 4](#_Toc532993278)

[2.4. Iterating over emails to return summary and detailed results 4](#_Toc532993279)

[2.4.1. Methods Used (9) 4](#_Toc532993280)

[2.4.2. Lessons Learned (5) 5](#_Toc532993281)

[2.4.3. Outcomes (2) 6](#_Toc532993282)

[2.5. Counting sender frequency into results summary 6](#_Toc532993283)

[2.5.1. Methods Used (4) 6](#_Toc532993284)

[2.5.2. Lessons Learned (2) 6](#_Toc532993285)

[2.5.3. Outcomes (4) 6](#_Toc532993286)

[2.6. Creating a detailed csv file 7](#_Toc532993287)

[2.6.1. Methods Used (4) 7](#_Toc532993288)

[2.6.2. Lessons Learned (0) 7](#_Toc532993289)

[2.6.3. Outcomes (1) 7](#_Toc532993290)

[3. End Products: 7](#_Toc532993291)

[3.1. Program console results 7](#_Toc532993292)

[3.2. CSV File results 8](#_Toc532993293)

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

# 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. [Day 30: Fetch & Convert Email in Python Programming](https://www.youtube.com/watch?v=bbPwv0TP2UQ), hosted by [YouTube.com](https://www.youtube.com/).

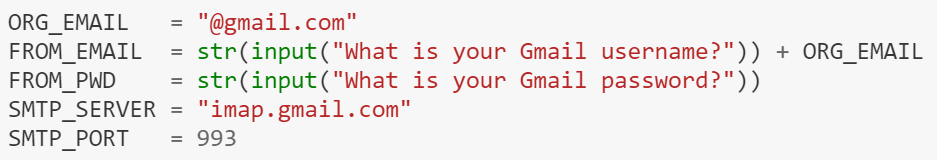
After initial unsuccessful attempts were made to generate proper server responses and email header or body content from the above boilerplate code it was determined further in-depth, step-by-step instruction was required to bring additional context for the functions and arguments used. Thus, a search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

# 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).

### Methods Used (1)

1. 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.





### Lessons Learned (2)

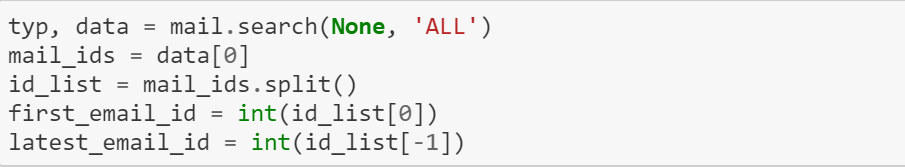
1. Outdated resources: It was found that many of the resources were outdated and that Python has been updated within the last few versions to create more enhanced email object functions and methods that are inherent to the language; previously many of these had to be imported separately
2. Enable Gmail 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

### Outcomes (1)

1. Connection string stored as variable: 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).

### Methods Used (2)

1. 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.
2. Type conversion (integer): 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.

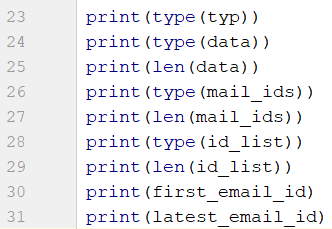
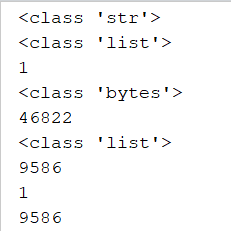


### Lessons Learned (2)

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).

### Outcomes (2)

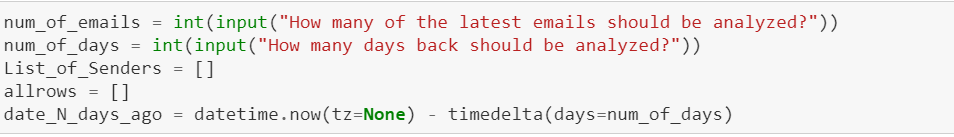
1. First email ID: The code block parsed the mail id list to provide the first email in the Inbox (exemplified below as 1)
2. Latest email ID: The code block parsed the mail id list to provide the latest email in the Inbox (exemplified below as 9,586)

* 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).

### Methods Used (4)

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 to be analyzed, starting with today and subtracting the number input from the user.
2. Type conversion (integer): the type conversion to integer is needed here to convert the input numbers to an integer number for later processing during the analysis code block.
3. Empty list variable creation: variables are created to store processed outputs from the analysis code block
4. Date/time references and deltas: the date/time functions were researched online using the [Python Standard Library>Data Types](https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior). In order to reference the current date and time without time zone awareness the parameter of tz was set to None. The timedelta function parameter is set to the number of days input by the user.



### Lessons Learned (2)

1. Calculating Date/Time fields: in the later analysis code block there is a statement that evaluates the date of each qualifying email against the date/time stored in the variable created here. This statement continued to error out because the date/time in this variable was time zone aware and the date/time in the email was naïve. Therefore the date/time created here had to be set to naïve as well. There were many different examples of how to do this and some sites recommended using pytz. After trial and error of using different parameters (i.e. timezone=None), importing the pytz module, etc. it was found the tz=None was correct.

### Outcomes (1)

1. Volume and timeframe from user: information obtained from the user on how many emails to analyze and within what timeframe
   1. Iterating over emails to return summary and detailed results was accomplished by creating a complex for loop to iterate over emails using the following outcomes: 2.1.3 connection variable, 2.2.3 first and last IDs, 2.3.3 volumes and timeframes, and the parsed IDs in a range. The loop (1) evaluated the ID against the range, (2) decoded and convert the email into an email object with recognized headers (dictionary keys), (3) created decision structures to format the date correctly, and (4) evaluated date for timeframe and appended corresponding information to lists for summation and csv output (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).

### Methods Used (9)

1. For loop range/unpacking: loop was used to iterate over emails based on criteria in the range function. Within the loop the headers of the emails are unpacked and evaluated for inclusion in the timeframe. Results are appended to lists.
2. Boilerplate code: coding statement from 1.2.3 above were used to leverage the connection variable method [.fetch] to retrieve the email according the passed ID in the ranged for loop. Additional coding statements were used to decode the results from the fetch method and create and email message object using the string type.
3. If/else decision structures: were employed to (1) evaluate the value of the date item for two types of date formats: [Day, DD Mon YYYY HH:MM:SS ~other info] and [DD Mon YYYY HH:MM:SS ~other info] in order to perform subsequent string parsing, and (2) evaluate the final formatted date against the timeframe delta obtained in 2.3 for inclusion in the data to be appended to the lists.
4. Dictionary items: the dictionary concepts of key-value pairs was used to reference the From, Date, and Subject headers of the email object
5. String parsing: provided a way to trim the date/time strings in order to exclude the [~other info] such as UTC or time zone qualifiers which were found to not to be standard.
6. Keyword “in”: was used to determine if the provided index number was in the range of emails obtained from the user and to determine if the first three letters of the date/time string were an abbreviated day
7. Type conversion (date/time): is used to convert the remaining date/time string post parsing to a date/time data type for calculation against the current date time
8. Variable assignment: used to capture each value of the specified item from the email object separately and in a list of tuples
9. List methods (append): adds each of the values from the key [‘From’] to an empty list variable; adds the resulting formatted date and the values from the keys [‘From’] and [‘Subject’] into a row list that mirrors the structure of a tuple; adds each of the row-list tuples as an element into a larger list (of tuples).



### Lessons Learned (5)

1. Utf-8 decode doesn’t always work: when the list of all emails (~10K) was initially ran through the for loop the program would hit an error stating that “utf 8 codec can’t decode byte 0x96 in position “x” invalid start byte.” Trial and error of a first few emails (limiting the list) worked until it hit a particular one (it was difficult to determine which email this was for troubleshooting). Therefore, the thought was to start at the end and work backwards. This was successful for 450 of the latest emails, thus it was concluded this was an old email using an out of date or illegal markup of some sort. This error introduced the need to limit the number of latest emails to analyze vs. performing analysis on the entire list. This is still a pitfall moving forward.
2. Unknown Date/Time function: research was conducted to uncover how to convert the date/time string to a date/time data type. Initially it was thought this would operate like excel parsing automatically, however it was quickly discovered that there were very specific formatting mnemonics that are needed to describe how the date/time is formatted in order to parse it correctly.
3. Dates are formatted differently: trial error of running the for loop resulted in errors with the date formatting mnemonics stating that either extra information remains that needs to be formatted or that the date/time string does not match the formatting mnemonics supplied. It was determined there were two formats used therefore prompting the use of a decision structure in which to handle both. This is another pitfall of this program if there are other unknown date/time formats
4. Naïve vs Aware Date/Time fields: once the date/time string was formatted into a date it was used to compare against the current date and time. Because the UTC and/or time zone qualifier was removed to format the date the time zone had to be removed from the current date/time in order to compare naïve dates/times.
5. Calculating Date/Time fields: in the later analysis code block there is a statement that evaluates the date of each qualifying email against the date/time stored in the

### Outcomes (2)

1. List of date, from, subject tuples: the for loop results in a list of tuples with header information from each email within the volume and timeframe provided
2. List of senders: the for loop results in a list of senders, one for each email evaluated within the volume and timeframe provided
   1. Counting sender frequency into results summary was accomplished using the counter function and the most common method thereof. The print function was used to display text back to the user on the total number of emails, the timeframe, the number of emails analyzed, and the top ten senders. General formatting was done to the top ten sender list to make it more readable (see the Returning Analysis Results in Summary Display and via a CSV File section of the [Gmail Inbox Analysis Jupyter Notebook](https://github.com/jaundickey/IS452FinalProject_Fall2018/blob/master/Gmail%20Inbox%20Analysis.ipynb) for detailed information).

### Methods Used (4)

1. Counter function and methods: the counter function was used to provide a frequency of each of the senders in a dictionary style counter object. The [.most\_common] method was employed to return the top ten senders in the list according to the frequency of each.
2. Print function: this was used to title the summary and clarify values of the variables in the summary, the “\n” character was used to create visible line breaks
3. For ranged loop: this loop iterates over the index number of each of the element tuples in the top ten senders list in order to create a line for each of the entries using the print function
4. String methods: the [.replace] method was used on the first tuple value of each of the entries in the top ten senders list to remove parentheses and quotes



### Lessons Learned (2)

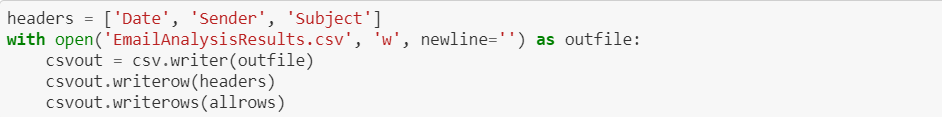
1. String methods are chainable: when performing the cleaning of the sender entry for each of the top ten, the initial approach was to create a variable to store the results and then perform the next [.replace] method. Chaining the replace method was attempted and worked.
2. Argument in print statement to remove space: it was discovered the “=sep” argument was needed to remove the whitespace between each of the elements in the print statement. This provided a clean line with the period after the rank and the appropriate space between the colon of the sender and the corresponding frequency.

### Outcomes (4)

1. Total number of emails in inbox: leverages the latest email id to retun the total number of emails in the inbox from the server
2. Timeframe analyzed: uses the variable that captures the number of days back input by the user to return a date and the date/time function of the current date/time to display a timeframe
3. Number of emails analyzed: employs the length of the senders list variable to return the number of emails that qualified for analysis between the timeframe, this cannot be more than the number of emails to be analyzed input by the user
4. Top ten senders: uses the for loop to print out the senders and their corresponding frequencies in ranked order
   1. Creating a detailed csv file was accomplished using the code block learned in class to write a single row as headers and subsequent rows as the detail (see the Returning Analysis Results in Summary Display and via a CSV File section of the [Gmail Inbox Analysis Jupyter Notebook](https://github.com/jaundickey/IS452FinalProject_Fall2018/blob/master/Gmail%20Inbox%20Analysis.ipynb) for detailed information).

### Methods Used (4)

1. Keyword “with”: leveraged around the file open and closing code statements to ensure the file stream remains managed within the statements
2. File open: employed to create a csv file object in which to write information to
3. CSV object methods: the [.writer] method of the csv object is used to create a variable in which to invoke the writerow and write row methods.



### Lessons Learned (0)

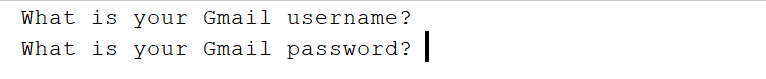
1. None: no additional knowledge was gained from this section

### Outcomes (1)

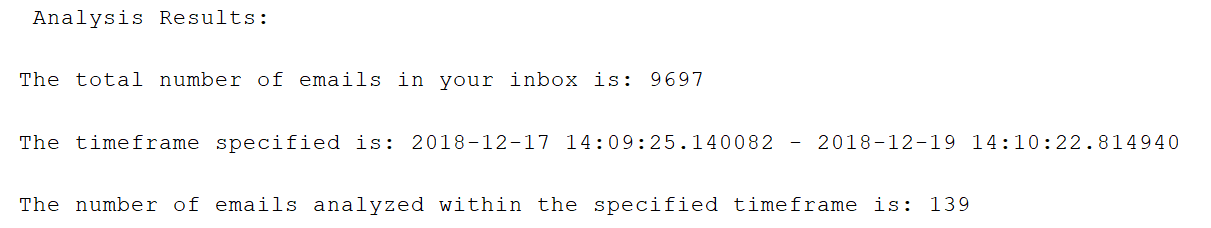
CSV file: a single row consisting of headers was created to categorize the remaining rows of tuples into additional rows of data that represent the date, sender, and subject of each of the emails analyzed

# End Products:

* 1. Program console results reflect inquiries to the user as described in 2.3.1 and result feedback statements to the user as described in 2.5.3 above.









* 1. CSV File results details described in 2.6.3 are written to the same directory in which the python programming coding script is running (see the Result Examples folder for full details).

