

Bridgewater State University – Computer Science Dept.

COMP390 – Software Engineering (Fall 2023)

Prof. Joseph Matta

Individual Coding Assignment 1.2 – Final Project

Overview and Project Requirements

For this final project, you will add functionality to your meteorite filtering program from Project 1.1. The new functionality will involve error checking and additional filter results output options. You will also incorporate “Clean Code” characteristics. Additionally, you will write unit tests for a few of your functions.

As in the first two projects, there are restrictions on which external libraries you may use. These are the external libraries you are ALLOWED to use for this project:

os

pathlib

xlwt (for Microsoft Excel tools)

datetime

pytest

NO OTHER EXTERNAL LIBRARIES/MODULES ARE ALLOWED. ALSO, NO OTHER PYTHON STANDARD LIBRARY IMPORTS! THIS MEANS YOU SHOULD HAVE NO OTHER ‘IMPORT’ STATEMENTS AT THE TOP OF YOUR FILES (EXCEPT FOR IMPORTS OF *MODULES LISTED ABOVE* AND/OR *MODULES YOU WRITE YOURSELF*).

This project should include most requirements for projects 1 and 1.1 in addition to the new requirements described in this document.

Project 1.2 Requirement Specifics

Many of the following requirements are similar or the same as the requirements for project 1.1. However, there are new requirements added for this project. Pay attention to the newly added requirements!

When the program starts there should be a welcome message printed to the terminal. The exact wording of the message is up to you, but it should give a basic introduction and some developer information. These items MUST be included in the welcome message:

- 1) A welcome message (“Welcome to the meteorite filtering program...”)
- 2) Very brief description of the program
- 3) The developer(s) name (your name)
- 4) A release date (usually just the month and year – ex. December 2023)

After the welcome message is printed, a series of user input prompts should be displayed in the terminal. Examples of these prompts at runtime are shown below.

```
Enter a valid file name (ex. "file_name.txt") with its file extension (if applicable) |or|
Enter ">q" or ">Q" to quit: meteorite_landings_data.txt

Target file: meteorite_landings_data.txt

What mode would you like to open the file with?
"r" - open for reading (default)
"w" - open for writing, truncating the file first. (WARNING: this mode will delete
      the contents of an existing file!)
"x" - open for exclusive creation, failing if the file already exists
"a" - open for writing, appending to the end of file if it exists
Enter ">q" or ">Q" to quit
Mode -> r

File mode: r

What attribute would you like to filter the data on?
1. meteor MASS (g)
2. The YEAR the meteor fell to Earth
3. QUIT
>> 1

Enter the LOWER limit (inclusive) for the meteor's MASS (g) ('Q' to QUIT): 2900000
Enter the UPPER limit (inclusive) for the meteor's MASS (g) ('Q' to QUIT): 1000000000000000
```

*** Notice the modified options for the file mode menu. Make this change to your code from Project 1.1. This change will simplify the file handling code, and it will make your menu more user-friendly. You should also include the warning shown in red text (you do not have to make the text print in red in your project solution; the text coloring is not a requirement). The purpose of this warning is to remind the user that if they open a file in 'write' mode, the contents of that file will be erased when the file is open. Thus, if 'write' mode is used on a file that already exists, any data in that file will be erased. This means that if 'write' mode is used to open 'meteorite landings data.txt', all 45,717 meteorite data entries will be deleted from the file, leaving 'meteorite landings data.txt' empty.*

The wording of your prompts does not need to be *exactly* the same as shown above, however, the prompts must be clear to the user and explicitly communicate what the user needs to do. Thus, your prompts should convey the intended instructions as shown in these examples. **You will lose points if your prompts are not clear and do not explicitly convey the appropriate instructions to the user.** Darker green text at the prompts indicates user input, and the text lines colored in brighter green between the prompt sections are confirmation messages. These confirmation messages **must** be displayed in the output. (The specific text coloring shown in this document is not required for your solution to the project.)

Another thing you'll notice is that each prompt has an option to quit the application. You may find it interesting that these "quit" options are executed differently by the user for each set of prompts. For the file access prompts (filename and file mode), the user is required to enter ">q" or ">Q" to quit the application. You may be curious as to why such a strange combination of characters is used to exit the app here. It turns out that there's a good reason for this. If you simply type "Quit" or "Q" at these prompts you may inadvertently open a file named "Quit" or "Q" in the current directory. Although it is unlikely that there is a file named "Quit," we must be very thoughtful when we design our applications. The ">" character is not allowed in a filename (Windows). Therefore, we can be sure that ">Q" and ">q" are not files in the current directory. (Note: This filename restriction may not be the same for other operating systems.)

For the *file mode* input, "q" could be confused as a mode option and will cause an exception without proper error checking. So, again, ">Q" and ">q" are used to quit the application at the file mode selection prompt.

For the *attribute selection menu*, we can simply use a menu option (choice 3) as our exit input.

A simple "q" or "Q" can be used at the *LOWER* and *UPPER limit prompts* because we are expecting a number here; the "q" and "Q" characters will not conflict with the expected input for the limits.

When the user enters the exit option at any of the prompts, the application should immediately exit and close after a short exit message. Examples are shown below. Each panel is a separate program run.

Exiting at the filename input prompt:

```
Enter a valid file name (ex. "file_name.txt") with its file extension (if applicable) |or|
Enter ">q" or ">Q" to quit: >q

The program is now exiting... GOODBYE!

Process finished with exit code 0
```

Exiting at the file mode input prompt:

```
What mode would you like to open the file with?
"r" - open for reading (default)
"w" - open for writing, truncating the file first. (WARNING: this mode will delete
      the contents of an existing file!)
"x" - open for exclusive creation, failing if the file already exists
"a" - open for writing, appending to the end of file if it exists
Enter ">q" or ">Q" to quit
Mode -> >Q

The program is now exiting... GOODBYE!

Process finished with exit code 0
```

Exiting at the attribute selection menu prompt:

...

```
File mode: r

What attribute would you like to filter the data on?
1. meteor MASS (g)
2. The YEAR the meteor fell to Earth
3. QUIT
>> 3

The program is now exiting... GOODBYE!

Process finished with exit code 0
```

Exiting at the LOWER limit input prompt (this is the same for the UPPER limit prompt). Notice that lowercase “q” does not work, only uppercase “Q” works to quit here:

```
Enter the LOWER limit (inclusive) for the meteor's MASS (g) ('Q' to QUIT): q
ERROR: Invalid range limit: "q"
Enter the LOWER limit (inclusive) for the meteor's MASS (g) ('Q' to QUIT): Q

The program is now exiting... GOODBYE!

Process finished with exit code 0
```

Your program MUST have this *exit/quit* functionality at each prompt, and there MUST be an *exit/GOODBYE* message printed before the application exits when the user chooses to exit this way.

New Filter Results Output Options

After the user supplies values for the prompts as shown above (without quitting), another menu will be presented. This menu will ask how the filtered results should be formatted:

```
How would you like to output the filter results?
1. On screen (in terminal)
2. To a TEXT file
3. To an EXCEL file
4. QUIT
>>
```

The requirements for each menu option are described below.

1. On screen (in terminal) – this option will print the filtered results to the terminal. This is like projects 1 and 1.1 where filtered results were printed to the terminal as neatly formatted tables. The table output for projects 1 and 1.1 only included the meteorite name and its mass or year; no other data points (i.e., id, nametype, GeoLocation, etc..) were included. **For this project (1.2)**, the table output to the terminal MUST include all twelve (12) data points for each meteorite in the result set. The list should be neatly formatted as in the list output for the previous projects. Sample output is shown below for filtering on mass 2,900,000 – 10E15 grams. The actual result table has 25 rows; only the top part of the table is shown here.

LEFT portion of the terminal output...:

	name	id	nametype	recclass	mass (g)	fall
=====	=====	=====	=====	=====	=====	=====
1	Jilin	12171	Valid	H5	4000000	Fell
2	Sikhote-Alin	23593	Valid	"Iron, IIAB"	23000000	Fell
3	Al Haggounia 001	44857	Valid	Aubrite	3000000	Found
4	Armanty	2335	Valid	"Iron, IIIIE"	28000000	Found
5	Bacubirito	4919	Valid	"Iron, ungrouped"	22000000	Found
6	Bendegiz	5815	Valid	"Iron, IC"	5360000	Found
7	Brenham	5136	Valid	"Pallasite, PMG-an"	4300000	Found
8	Campo del Cielo	5247	Valid	"Iron, IAB-MG"	50000000	Found
9	Canyon Diablo	5257	Valid	"Iron, IAB-MG"	30000000	Found
10	Cape York	5262	Valid	"Iron, IIIAB"	58200000	Found

...RIGHT portion of the terminal output:

year	reclat	reclong	GeoLocation	States	Counties
=====	=====	=====	=====	=====	=====
1976	44.05	126.16667	"(44.05, 126.16667)"		
1947	46.16	134.65333	"(46.16, 134.65333)"		
2006	27.5	-12.5	"(27.5, -12.5)"		
1898	47	88	"(47.0, 88.0)"		
1863	26.2	-107.83333	"(26.2, -107.83333)"		
1784	-10.11667	-39.2	"(-10.11667, -39.2)"		
1882	37.5825	-99.16361	"(37.5825, -99.16361)"	17	1238
1575	-27.46667	-60.58333	"(-27.46667, -60.58333)"		
1891	35.05	-111.03333	"(35.05, -111.03333)"	7	986
1818	76.13333	-64.93333	"(76.13333, -64.93333)"		

The complete table including the two portions shown above should be displayed in the terminal. In PyCharm's terminal, the entire table is displayed cohesively. You must scroll over to the right to see the right half of the table. The table must be neatly formatted as shown.

2. To a TEXT file – this option causes the filtered output data to be written to a text file. Like the table printed to the terminal, the text file will contain all twelve data points for each of the meteorites in the result set. Additionally, the text file must be formatted the same as the original “meteorite_landings_data.txt” text file: all data values are tab separated, and the first line of the file has the name of each meteorite attribute, also tab separated. Since the filtered output text file is formatted the same as the original meteorite data file, it should be possible to pass the filtered text file in as the starting data set. In other words, you should be able to run a filtering process on the filtered text file as if it was the original dataset. There is also a specific naming convention required for your filtered text file filename. You MUST use the current date and time to name your output text file. An example is shown below in green:

“2023-11-27_14_44_42_496754.txt”

‘2023-11-27’ is the year, month, and day the file was created

‘14_44_42_496754’ is the hour, minute, second, and microseconds at which the file was created

This function will help you get the required filename format:

```
from datetime import datetime

def get_clean_datetime_string():
    current_timestamp = datetime.now()
    current_timestamp.strftime("%Y-%m-%d %H-%M-%S")
    clean_timestamp_str = current_timestamp.__str__().replace(':', '_')
    clean_timestamp_str = clean_timestamp_str.replace('.', '_')
    clean_timestamp_str = clean_timestamp_str.replace(' ', '_')
    return clean_timestamp_str
```

The `get_clean_datetime_string()` function returns a string in the following format

“2023-11-27_14_44_42_496754”

This is the same format as the required file name *without the “.txt” extension*. You MUST include the TEXT filetype extension (.txt) in the file name:

“2023-11-27_14_44_42_496754.txt”

Sample output text file content is shown below for filtering on mass 2,900,000 – 10E15 grams. The actual result has 25 entries but only the top part of the file content is shown.

	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLocation	States	Counties
1	Jilin	12171	Valid	H5	4000000	Fell	1976	44.05	126.16667	"(44.05, 126.16667)"		
3	Sikhote-Alin	23593	Valid	"Iron, IIAB"	23000000	Fell	1947	46.16	134.65333	"(46.16, 134.65333)"		
4	Al Haggounia	001	44857	Valid	Aubrite	3000000	Found	2006	27.5	-12.5	"(27.5, -12.5)"	
5	Armanty	2335	Valid	"Iron, IIIE"	28000000	Found	1898	47	88	"(47.0, 88.0)"		
6	Bacubirito	4919	Valid	"Iron, ungrouped"	22000000	Found	1863	26.2	-107.83333	"(26.2, -107.83333)"		
7	Bendeg	5015	Valid	"Iron, IC"	5360000	Found	1784	-10.11667	-39.2	"(-10.11667, -39.2)"		
8	Brenham	5136	Valid	"Pallasite, PMG-an"	4300000	Found	1882	37.5825	-99.16361	"(37.5825, -99.16361)"	17	1238
9	Campo del Cielo	5247	Valid	"Iron, IAB-MG"	50000000	Found	1575	-27.46667	-60.58333	"(-27.46667, -60.58333)"		
10	Canyon Diablo	5257	Valid	"Iron, IAB-MG"	30000000	Found	1891	35.05	-111.03333	"(35.05, -111.03333)"	7	986
11	Cape York	5262	Valid	"Iron, IIIAB"	5820000	Found	1818	76.13333	-64.93333	"(76.13333, -64.93333)"		

Notice that output text file is formatted exactly the same as the original “meteorite_landings_data.txt” data file.

3. To an EXCEL file – for this option, the filtered results will be written to a Microsoft Excel spreadsheet. The way the spreadsheet is organized can be understood by comparing it to the way the output text file is written. Instead of being tab separated as in the text file, each meteorite data value will occupy its own cell in the spreadsheet. You will also include the attribute names in the first row of the spreadsheet. Each name should occupy its own cell.

A sample output excel sheet is show below for filtering on mass 2,900,000 – 10E15 grams. The actual result has 25 entries but only the top part of the sheet is shown. This sample is opened in the Microsoft Excel application:

	A	B	C	D	E	F	G	H	I	J	K	L
1	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLocation	States	Counties
2	Jilin	12171	Valid	H5	4000000	Fell	1976	44.05	126.16667	"(44.05, 126.16667)"		
3	Sikhote-Alin	23593	Valid	"Iron, IIAB"	23000000	Fell	1947	46.16	134.65333	"(46.16, 134.65333)"		
4	Al Haggounia	001	44857	Valid	Aubrite	3000000	Found	2006	27.5	-12.5	"(27.5, -12.5)"	
5	Armanty	2335	Valid	"Iron, IIIE"	28000000	Found	1898	47	88	"(47.0, 88.0)"		
6	Bacubirito	4919	Valid	"Iron, ungrouped"	22000000	Found	1863	26.2	-107.83333	"(26.2, -107.83333)"		
7	Bendeg	5015	Valid	"Iron, IC"	5360000	Found	1784	-10.11667	-39.2	"(-10.11667, -39.2)"		
8	Brenham	5136	Valid	"Pallasite, PMG-an"	4300000	Found	1882	37.5825	-99.16361	"(37.5825, -99.16361)"	17	1238
9	Campo del Cielo	5247	Valid	"Iron, IAB-MG"	50000000	Found	1575	-27.46667	-60.58333	"(-27.46667, -60.58333)"		
10	Canyon Diablo	5257	Valid	"Iron, IAB-MG"	30000000	Found	1891	35.05	-111.03333	"(35.05, -111.03333)"	7	986
11	Cape York	5262	Valid	"Iron, IIIAB"	5820000	Found	1818	76.13333	-64.93333	"(76.13333, -64.93333)"		
12	Chupaderos	5363	Valid	"Iron, IIIAB"	24300000	Found	1852	27	-105.1	"(27.0, -105.1)"		
13	Cranbourne	5463	Valid	"Iron, IAB-MG"	8600000	Found	1854	-38.1	145.3	"(-38.1, 145.3)"		
14	Gibeon	10912	Valid	"Iron, IVA"	26000000	Found	1836	-25.5	18	"(-25.5, 18.0)"		
15	Hoba	11890	Valid	"Iron, IVB"	60000000	Found	1920	-19.58333	17.91667	"(-19.58333, 17.91667)"		
16	Mbosi	15456	Valid	"Iron, ungrouped"	16000000	Found	1930	-9.11667	33.06667	"(-9.11667, 33.06667)"		
17	Morito	16745	Valid	"Iron, IIIAB"	10100000	Found	1600	27.05	-105.43333	"(27.05, -105.43333)"		

The excel file name should be generated in the same way as the text file name: use the *get_clean_datetime_string()* function to get a *datetime* string, *but instead of appending “.txt” to the string, append “.xls” for the excel file extension.*

This partially completed pseudocode-ish function should help you implement the excel spreadsheet generation. (Note: this function is too big to be a clean function and does not satisfy the clean code requirements for this project. You will have to make some adjustments and refactor accordingly. Remember, this is pseudocode, it's not entirely written in Python syntax; its purpose is to guide you in the implementation):

```
# -----

from xlwt import Workbook

def write_filtered_results_to_excel_file():
    # Workbook is created
    excel_workbook = Workbook()

    # add a sheet to the workbook
    filtered_data_sheet = excel_workbook.add_sheet('filteredMeteoriteData')

    # write the attribute titles to the top of the sheet
    index = 0
    for name in attribute_name_list ['name', 'id', 'nametype', ...]:
        # write top row of the Excel output sheet -- __.write(row, column, value)
        filtered_data_sheet.write(0, index, name)
        index++

    # loop through all the filtered meteorite objects
    for index in range(len(filtered_meteorite_object_list):
        # get the current meteorite object
        current_meteorite_record_obj = filtered_meteorite_object_list[index]

        # get a Python list with all 12 of the current meteorite object's attributes
        # extract_data_from_meteorite_object() should return a list of 12 values
        attribute_list = extract_data_from_meteorite_object()

        # loop through the attribute strings in the attribute list
```

```

        for attr_index in range(len(attribute_list)):
            # write each row of the Excel output sheet -- __.write(row, column, value)
            filtered_data_sheet.write(index + 1, attr_index, attribute_list[attr_index])

    clean_timestamp_str = get_clean_datetime_string()
    excel_workbook.save(f'{clean_timestamp_str}.xls')

    # confirmation message
    print(f'\n\033[92mFiltered output sent to "{clean_timestamp_str}.xls"\033[0m')

# -----

```

4. QUIT – This option should function like the other quit options in your program. If option “4” is chosen, the program will quit with a goodbye message:

```

How would you like to output the filter results?
1. On screen (in terminal)
2. To a TEXT file
3. To an EXCEL file
4. QUIT
>> 4

The program is now exiting... GOODBYE!

Process finished with exit code 0

```

Clean Code Requirements

The following clean code requirements MUST be addressed in your project. Failing to include one or more of these requirements will result in points being deducted from your project grade.

Documentation:

- 1) Every file, module, class, class method, and function you write must have a Python docstring associated with it, including the “main()” function, the “main.py” module, and all test functions and test modules. This will require you to think about the functions you write. Make your functions, classes, and modules focused and simple so that your docstrings can clearly explain what the code does.
- 2) Include a **README.md** file in your project. The README requirements are described below (a.-g.):
 - a. your first and last name
 - b. your COMP390 section. For example, if I am in section 001, I would write: COMP390-001
 - c. a brief overview of what your project does (don’t go deep into implementation details here, but provide a complete and succinct summary of your project)
 - d. **The URL of your project’s GitHub repository**
 - e. What third-party libraries must be installed for your project to run (i.e.. xlwt, pytest, etc...)
 - f. **a guide on how to use your program.** This section will be a bit extensive because you are adding a significant amount of functionality to your project. Make sure to explain how to use your program using terms and language appropriate for someone who does not have a computer science background. Also, assume that the reader has never used your program before.
 - g. a brief description of what does not work in your project and/or what you did not complete (if you completed all of the project requirements, simply write “all project requirements completed” for this part)

Clean Functions:

- 1) **No function can be more than ten (10) lines.** This line limit does NOT include comments, blank lines, and docstrings. For example, if a function has three lines of docstrings, three lines of comments, ten lines of code, and two blank lines, that function is OK: 10 (code lines). You will lose points for every function that does not adhere to this restriction. Hint: If a function is required to print a large amount of text, consider putting the text in a text file and creating a function that reads that file and displays the text to the screen. This is better than having a function be 20 lines long to display 20 lines of text.

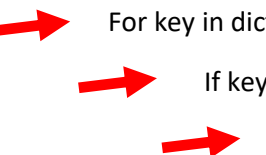
- 2) No function can have more than two levels of indentation. (The indentation for the function body does not count.)

For example:

This function is NOT acceptable:

bad_func():


```
For _ in list:
    For key in dict:
        If key ....
            return True
```



The function above is unacceptable because it has three levels of indentation.

good_func():

```
For _ in list:
    For key in dict:
        Helper_func(key)
```



The above function is acceptable because it only has two levels of indentation.

- 3) NOTE: Your unit test functions are exempt from the ten-line code limit. Your test functions may include more than ten lines to accommodate your tests' coverage.

Unit Tests:

- 1) Include a suite of at least **THREE** automated unit tests. These tests should only test functions that you write.
- 2) Use *pytest* to run your tests.
- 3) All test cases must PASS for all tests.
- 4) Your tests should have good coverage and test error conditions and edge cases. Error conditions/exceptions must be caught using the *pytest.raises()* function as a context manager (inside a "with" block) and then the error type must be asserted. If the function being tested allows certain errors/exceptions to be thrown, you MUST demonstrate this in your test.

- 5) If you test a function that accepts one or more parameter(s), make sure you test edge cases. For example, if you write a function that takes an *int* as a parameter, you must test "0" as a value. You should always test *None* as a parameter in functions that take parameters.
- 6) At least one test must test a function that accepts user input. User input must be mocked using the *pytest monkeypatch* fixture. If the function outputs (prints) anything to the terminal, the output must be captured and tested using the *pytest capfd* fixture. Of course, any possible error conditions must also be caught using *pytest.raises()*, and the error type must be asserted.

Error checking:

There MUST be appropriate error checking present in your project. For example, if you write a function that expects a user to input an integer in the terminal, make sure the value entered is indeed an integer. Your program should not crash if the user enters "9j6" instead of "96" at a prompt that expects an integer. **In fact, your program should check all errors such that it will NEVER crash with an error/exception no matter what the user enters at the prompts.** If the user enters an invalid value at one of the prompts, the program should print an error message to the terminal and re-prompt the user.

An example of this re-prompting behavior is shown below for the target data file name prompt. This prompt will only accept valid filenames of files that exist in the project folder. It will also accept ">q" or ">Q" to quit the application. If any invalid value is entered, the user is prompted again for a valid filename. (Red text coloration for the error messages is not required.)

```
Enter a valid file name (ex. "file_name.txt") with its file extension (if applicable) |or|
Enter ">q" or ">Q" to quit: 34v31rq2df
ERROR: TARGET FILE NAME "34v31rq2df" IS NOT VALID!

Enter a valid file name (ex. "file_name.txt") with its file extension (if applicable) |or|
Enter ">q" or ">Q" to quit:
ERROR: TARGET FILE NAME "" IS NOT VALID!

Enter a valid file name (ex. "file_name.txt") with its file extension (if applicable) |or|
Enter ">q" or ">Q" to quit: missing_file.txt
ERROR: TARGET FILE NAME "missing_file.txt" IS NOT VALID!

Enter a valid file name (ex. "file_name.txt") with its file extension (if applicable) |or|
Enter ">q" or ">Q" to quit:
```

Another example of this user input error catching behavior is shown below for the *upper limit filter parameter entry*. The filter parameter prompts will only accept valid integers for filtering.:

```

Enter the LOWER limit (inclusive) for the meteor's MASS (g) ('Q' to QUIT): 29dwed000
ERROR: Invalid range limit: "29dwed000"
Enter the LOWER limit (inclusive) for the meteor's MASS (g) ('Q' to QUIT): 29000000
Enter the UPPER limit (inclusive) for the meteor's MASS (g) ('Q' to QUIT): joe
ERROR: Invalid range limit: "joe"
Enter the UPPER limit (inclusive) for the meteor's MASS (g) ('Q' to QUIT):
ERROR: Invalid range limit: ""
Enter the UPPER limit (inclusive) for the meteor's MASS (g) ('Q' to QUIT): )))))
ERROR: Invalid range limit: ")))))"
Enter the UPPER limit (inclusive) for the meteor's MASS (g) ('Q' to QUIT): 1000000000000000000

```

The user input error checking behavior is again demonstrated below at the *attribute selection menu*. Only the values “1”, “2”, or “3” will be accepted at this prompt; all other values will be caught, and the user will be re-prompted until a valid value is entered.

```

What attribute would you like to filter the data on?
1. meteor MASS (g)
2. The YEAR the meteor fell to Earth
3. QUIT
>> 3r23rc
ERROR: Invalid menu choice: "3r23rc"

What attribute would you like to filter the data on?
1. meteor MASS (g)
2. The YEAR the meteor fell to Earth
3. QUIT
>> 4
ERROR: Invalid menu choice: "4"

What attribute would you like to filter the data on?
1. meteor MASS (g)
2. The YEAR the meteor fell to Earth
3. QUIT
>>

```

All user prompts in the program should exhibit similar user error checking; only valid values should be accepted at each prompt.

Limited dependencies:

Only the following libraries can be used for this project:

os
pathlib
xlwt (for Microsoft Excel tools)
datetime
pytest

Submission and Additional Requirements

- 1) Create a new GitHub repository named "COMP390_Individual_Project1_2" (** Make sure your repository is set to PRIVATE **)
- 2) Push your project to your "COMP390_Individual_Project1_2" repository.
- 3) Make me (Prof. Matta) a collaborator on your repository. Use my email (j1matta@bridgew.edu) to search for my GitHub account to make me a collaborator. My GitHub username is 'j1matta'. My GitHub profile picture is shown below. Look for this picture when searching for my profile to further confirm that you are indeed making *me* a collaborator and not someone else.



Note: you should push changes made to your project to your GitHub repository often while you are developing your code, not just when you have completed the project. GitHub repositories are a great way to back up your work.

***** This project is worth 250 points. It is due NO LATER than the last day of finals - DECEMBER 21st at 11:59 PM. LATE SUBMISSIONS WILL NOT BE ACCEPTED. *****

~~~~ ~~