Carleton UNIVERSITY
Canada's Capital University

MILESTONE 2

This document is supposed to be read in conjunction with the overall Project Description. It only describes the tasks associated with Milestone 2.

This second milestone also contains two sets of deliverables and spans two lab periods (i.e., two weeks). Accordingly, to help you manage your workload, the work is described as two labs and included in this document. Therefore, you can work at your own pace and see the progression of tasks that compose a project.

Time management is one of the skills you are learning during this project. We are helping you during this journey. That's why we have split the deadlines: ~~division of tasks are due on Fridays, individual work on Mondays, and teamwork on Wednesdays~~. We encourage you to be on top of the deadlines and not leave everything for the last minute. For example, you will notice that you should divide tasks to complete the individual work. Dividing the tasks as soon as possible helps you and your teammates better organize your time.

File Naming Convention: Throughout the project, you will be given specific formats for your filenames that include your team identifier, e.g., T021.

Milestone 2 Objectives

The first milestone established a good understanding of the project – including the problem domain of dataset analysis, the project goals of building an interactive dataset analyzer program, and the problem solution.

The objectives of the second milestone are as follows:

1. Expand the dataset analyzer program. The team will learn how to analyze the datasets by applying different types of analysis to the given dataset. This will also increase the amount of code that the team must manage.
2. Develop a text-based user interface whereby a user selects an operation to be applied to the dataset by entering text commands.
3. Work with industry-standard practices, including test-driven development and peer code reviews.

Contents

Milestone 2 Lab 4 (P3): Sorting

P3 Task 1: Sorting

In this task, your team will use the function design recipe to develop functions to sort the dataset based on different criteria and store it in other formats using combinations of lists and dictionaries. Each function will sort the dictionary books using specific criteria and will return a list with the book data stored as a dictionary.

Book data example:

```
{"title": "Antiques Roadkill: A Trash 'n' Treasures Mystery",
 "author": " Barbara Allan",
 "language ": "English",
 "rating": 3.3,
 "publisher": " Kensington Publishing Corp.",
 " category": [" Fiction", "Comedy"]
 "pages": 288}
```

Function 1: sort_books_title

Use the function design recipe to develop a function named `sort_books_title`.

The function takes a dictionary as an input parameter. This dictionary contains books stored as explained in Milestone 1 – lab 1 – Case 1.

First, the function creates a list with the book data. Each list element is a dictionary where **all** book data is stored. Note that the same book can be in different categories, so you will have a list on the "category" key of the dictionary (see example above). Then, the function uses a bubble sorting algorithm to sort the books by their titles based on the alphabetical order.

The function returns a list with the book data stored as a dictionary book where the books are sorted alphabetically by title. **The function will NOT have any print statements**.

Function 2: sort_books_publisher

Use the function design recipe to develop a function named `sort_books_publisher`.

The function takes a dictionary as an input parameter. This dictionary contains books stored as explained in Milestone 1 – lab 1 – Case 1.

First, the function creates a list with the book data. Each list element is a dictionary where **all** book data is stored. Note that the same book can be in different categories, so you will have a list on the "category" key of the dictionary (see example above). Then, the function uses a bubble sorting algorithm to sort the books alphabetically by publisher. <u>The books from the same publisher must be sorted alphabetically by title.</u>

The function returns a list with the book data stored as a dictionary book where the books are sorted alphabetically by publisher. **The function will NOT have any print statements**.

Function 3: sort_books_author
Use the function design recipe to develop a function named `sort_books_author`.

The function takes a dictionary as an input parameter. This dictionary contains books stored as explained in Milestone 1 – lab 1 – Case 1.

First, the function creates a list with the book data. Each list element is a dictionary where **all** book data is stored. Note that the same book can be in different categories, so you will have a list on the "category" key of the dictionary (see example above). Then the function uses a bubble sorting algorithm to sort the books based on their author in alphabetical order. The books with the same author must be sorted alphabetically by title.

The function returns a list with the book data stored as a dictionary book where the books are sorted alphabetically by author name. **The function will NOT have any print statements**.

Function 4: sort_books_ascending_rate
Use the function design recipe to develop a function named `sort_books_ascending_rate`.

The function takes a dictionary as an input parameter. This dictionary contains books stored as explained in Milestone 1 – lab 1 – Case 1.

First, the function creates a list with the book data. Each list element is a dictionary where **all** book data is stored. Note that the same book can be in different categories, so you will have a list on the "category" key of the dictionary (see example above). Then the function uses a bubble sorting algorithm to sort the books by the rate in ascending order. Books with the 'N/A' rate should appear first. Books with the same rate must be sorted based on their title.

The function returns a list with the book data stored as a dictionary book where the books are sorted by their rate in ascending order. **The function will NOT have any print statements**.

As mentioned before, there are four sorting functions and, in most teams, there are four members.

The leader of the team will:

1. Ensure that everyone understands the task that should be completed. The team leader does not need to be the expert and "have the solution"; instead, the leader must ensure that each member is empowered to do their portion of the work.
2. Delegate who is writing which functions.
3. You need to submit the delegation of tasks. Each member is supposed to write one sorting function and the test module for one function. If a team consists of three members (or, if

there are absent teammates), the team will have to divide the extra work (fairly) to complete all the functions.

4. To ensure the test module is developed independently from the functions, none of the students can write the test module for the function they are implementing. Each student should write the test for one function. If you are a team of three (or, if there are absent teammates), you will have to divide the extra work (fairly) to complete all the tests. Note that you do not depend on your teammate for this task. Your test harness should be ready before you have access to the function itself.

Each individual shall:

1. Follow the FDR to construct their assigned function. Required Filename: **Txxx_P3_sorting.py** where xxx is your team identifier.
2. Develop the test for their assigned function. You must test your functions programmatically, as we have been doing previously. Write a corresponding test function for the function/s you were given, following the practices taught in the unit testing lecture. Required Filename: **Txxx_P3_sorting_test.py** where xxx is your team identifier
3. Your submission should contain (1) the .py file for the sorting function, (2) the .py for the test, and (3) (if needed) the csv files used for testing and to load the data.

Note: Ensure your automated testing counts the number of tests passed and the number of tests that fail. If you use a modified version of check_equal (introduced in Lecture 5), you can define that function in its own module and re-use it throughout the whole project. Do not forget to include that file in your submission and include the import statements in your code.

Each *INDIVIDUAL* must submit their own work to be marked individually.

P3 Task 2: Team Code

At this point, you have implemented another module for your program.

Now, as a team, you must integrate all the functions and tests.

Your file should be named **Txxx_P3_sorting_fun.py,** and it should be organized as follows:

```
#imports

#The four functions from P3 – Task 1 (and extra function that arise from
refactoring)

#Main script with automated testing. The main script should only run if the
module is executed. It should not run when the module is imported. The main
script should print the total number of tests that pass and fail.
```

Before the code is delivered, it must be refactored. Refactoring is a term used to describe the process of cleaning up and reorganizing the code. The code works, but you refactor it to make it more readable, understandable, and maintainable (and perhaps more efficient).

You will notice that the four functions first convert the dictionary into a list. That code can be refactored. Write a function (e.g., dictionary_to_list()) that encapsulates that functionality. Your sorting functions should use this function instead of repeating the code.

You will also notice that your automated testing uses the same function (a version of check_equal - introduced in Lecture 5). Create a module for this function and name it **Txxx_check_equal.py**. All your tests should use the function implemented in this module.

Your team submission should include, at least, these two improvements.

Submit the team's code. **One submission for the whole team.** Your submission should include:

- **Txxx_P3_sorting_fun.py**
- **Txxx_check_equal.py**
- Any csv files used for testing
- **T**xxx_P1_load_data.py (if used to load csv files)

P3 Task 3: ITP Metrics

You will complete the ITP Metrics Peer Feedback and Team Dynamics assessment. You complete these assessments individually, but ITP Metrics combines the results into an overall team assessment.

On your own and without interference from any member of your team:

1. Login into your account on ITP Metrics. (You should receive an email by Wednesday, 23rd. If you didn't receive the email by that day, send an email to Dr. Ruiz-Martin no later than Thursday 24th at 11 am).
2. Please first verify that your team members are correctly listed. If this is not the case, please contact Dr. Ruiz-Martin immediately. Do not complete the task until the settings are corrected.
3. Individually, complete the Peer Feedback and Team Dynamics Assessment.

Optional: There are PDF reports generated by ITP metrics that give you more insight into your peers' feedback. These PDF reports are only generated if (1) your group size is three or more and (2) all members of your group complete the task. If these conditions are met, you are free to download these PDF reports and take a look, but it is not mandatory. Please do not email the instructors if a message is not generated – we cannot do anything about it. That is why this is an optional activity.

Summary of the deliverables for Milestone 2 – P3

- **P3 – Task 1.1:** Division of tasks
- **P3 – Task 1**: Implementation of functionalities and testing - Individual submission – Two files per student. You may submit as many csv files as you need for testing.
- **P3 – Task 2:** Code review – Team submission – At least two files per team
- **P3 – Task 3:** Complete the ITP metrics.

Milestone 2 Lab 5 (P4): The Text-Based Interactive User Interface

You will now build a text-based user interface, calling the data analysis functions that your team developed for Milestone 1 and Milestone 2 – lab 4

The exact required user interface for Milestone 2 is shown in *Figure 1*.

```
The available commands are:
    1-  L)oad data
    2-  A)dd book
    3-  R)emove book
    4-  G)et books
            T)itle    R)ate   A)uthor   P)ublisher   C)ategory
    5-  GCT) Get all Categories for book Title
    6-  S)ort books
            T)itle    R)ate    P)ublisher    A)uthor
    7-  Q)uit

Please type your command: <one space>
```

Figure 1: Required User Interface

Requirements:

1. The user interface is shown in *Figure 1*. It consists of the ten-line menu of commands, followed by a line with the statement "Please type your command" followed by the command prompt (a colon followed by one space). To execute a command, the user types the upper-case letter/s to the left of the '),' then presses the Enter key. After the command is executed and the results are displayed, the user should be prompt to enter another command. The entire menu, with the statement "Please type your command" and the command prompt is redisplayed.

2. Most of the commands are upper-case letters; for example, the L in L)oad file. For these commands, both upper-case and lower-case letters should be accepted; for example, both L and l will trigger the code associated to selecting and loading a file.

3. The L)oad file command allows the user to select and read the dataset file (i.e., enter the name of the file to be loaded).

4. The command lines *G and S* will ask the user to enter another letter to select the specific analysis the user wants to perform; for example, if the user enters the letter *S*, then the message "How do you want to sort?" will be displayed and user needs to enter another letter to sort the data such as *T for* the title.

6. After the command is executed, the execution results are displayed on the console.

7. The Q)uit command terminates the program.

8. The application will display the error message "File not loaded" if you attempt to use one of the commands before loading a file into the editor.

9. The application will display the error message "No such command" if an invalid command is typed. Note that if an invalid command is entered before a file has been loaded, **only** "No such command" should be displayed; it means, "File not loaded" **must not be** displayed simultaneously.

11. Do not provide additional commands in the dataset analyzer editor. If you want to add enhancements on your own time, you are certainly encouraged to do so, but for this milestone, you should not provide extra features, and the user interface will provide only those commands shown in *Figure 1*.

In addition, this feature functionally must implement the interaction with the user to get the additional parameters inside the function definition. For example, after typing G and R, the user should be prompted to enter a rate.

Iterative, Incremental Development of the Interactive User Interface

(This is a guide on how to approach this milestone)

Here is a suggested set of iterations when building the interactive user interface:

1. Write a while loop that repeats a fixed number of times (perhaps two times), prompting the user to enter a string (any string) and simply printing that string. For example,

```
Please enter a string: hello

hello

Please enter a string: Python

Python
```

2. Modify the body of the while loop so that it converts the string to the upper case, then prints it. Hint: Python's str type provides a very useful method. In the shell, type help(str), or have a look at Practical Programming, Chapter 7, Using Methods. The user interface should look like this:

```
Please enter a string: hello

HELLO

Please enter a string: Python

PYTHON
```

3. Modify the loop so that, instead of looping a fixed number of times, the enter-and-print-string code can be executed until the letter Q (or q) is typed:

```
Please enter a string (Q to quit): hello

HELLO

Please enter a string (Q to quit): Python

PYTHON
```

```
Please enter a string (Q to quit): programmer
PROGRAMMER
Please enter a string (Q to quit): Q
Q
```

4. You now have a primary command-interpreter loop, but only one command is currently recognized. It's time to modify the loop to provide the required editor user interface, but again, take it one step at a time.

   a. Modify the loop so that instead of prompting the user to enter a string, it displays the menu of commands and the command prompt, as shown in *Figure 1*.

   b. Write the code to recognize the L)oad file command. For all other commands – for now – print out the required error message ("No such command").

   c. Test the editor. It should be able to repeatedly load (select and display) files until the Q)uit command is entered.

5. Pick one command. Modify the loop so that it can call the command function when the corresponding command is entered. For example, modify the loop to find a book by title, the user then will be asked to enter the book title, then the program will call "*find_books_by_title*" to find this book, and the program will display the result if the book is found or not. Remember to do regression testing: ensure this most recent change didn't "break" any code that worked in the previous step.

6. Modify the loop to display the error message "No file loaded" if you attempt to use the command before you have loaded a file into the editor. (There are different ways to do this, but your instructors and the TAs are not going to tell you the best way. That is something the team must collectively design. Think carefully about what variables are needed and the best choice for the type of value assigned to each one.) Test again.

7. Incrementally extend the editor to provide the other commands, one at a time.

The user interface must prompt the user to enter the book information for the A)dd book command, R)emove book command, etc.

P4 Task 1: Milestone Planning and Individual Code

In a standard group of four, the work for the user interfaces should be divided between the team members: Each team member will work on implementing a set of commands as follows.

Each team member must have the primary interface explained in the previous section steps 1- 4.

Each team member will work on different commands for steps 5, 6, and 7.

1- Case 1: A team member will work on commands add and remove book.

2- Case 2: A team member will work on commands get books by title, rate, and author.
3- Case 3: A team member will work on commands get books by publisher and category and get all categories for book title.
4- Case 4: A team member will work on command Sort books and its subcommands.

For groups of less than four students, please develop an agreeable workload distribution.

In recent lectures, we demonstrated how to code using iterative, incremental development. We recommend that the team follow this approach while developing the user interfaces instead of following the "big-bang" technique (code everything at once, cross your fingers, and hope it works).

The leader of the team shall:

1. Ensure that everyone understands the task that should be completed. The team leader does not need to be the expert and "have the solution"; instead, the leader must ensure that each member is empowered to do their portion of the work.
2. Delegate who is writing which functions and <u>submit the delegation of tasks</u>.
3. Each member will write their share of the user interface.

Each individual will:

1. Follow the FDR to construct their assigned functionalities. Do not forget to import the modules you have previously developed.
2. <u>Required Filename</u>: **Txxx_P4_UI_yyy.py** where xxx is your team identifier and yyy = {case1, case2, case3, or case4}
3. You must test your function. For this deliverable, you can just call your function and ensure that the user interface works correctly. You do not need to submit any tests.
4. Your submission must include a functional version of your Python program. That means the TA should be able to execute your program (Txxx_P4_UI_yyy.py) and use the interface provided (only the commands you were assigned) without any additional files. This means that your submission must include the modules you previously developed as a team.

Each <u>*INDIVIDUAL*</u> must submit their own work to be marked individually.

P4 Task 2: Final Team Code

At this point, you have implemented the user interface commands lines as separate functionalities. Now, as a team, you must integrate all the commands lines together to have a Text-Based Interactive User Interface, as shown in Figure 1. Your team must review the user interface code and ensure that the user interface works correctly.

Before the code is delivered, it must be refactored. As a team, review the user interface code. Here are some questions to ask yourself:

- The user interface should not be one long script. Break it up into functions. For example, you could define a function that just displays the menu of commands. You could specify another function that calls the display-menu function, prompts the user to enter a command, and returns the command. These are just suggestions - we are leaving it up to the team to decide how many functions this module should have and what each of them should do.
- All the function headers must have type annotations. All the functions must have complete docstrings.
- You can use a long if-elif-else statement to check if a command entered by the user is valid, but there are better ways. For example, the program can initialize a list with all the single-character commands. Given this list, checking whether a command entered by the user is valid takes one line of code.
- Does all your new code follow the coding conventions? If not, edit the code so that it does. Polish, polish, polish (and while doing this, remember to retest, retest, retest).

Required Filename: **Txxx_P4_booksUI.py**, where xxx is your Brightspace team identifier

Before submitting:

1. Review the project-wide expectations for code submissions. For instance, the file must have a header with the team identifier, and each function must have the author's name.
2. Make sure you have refracted your code as explained above, and it still compiles and runs.

Submit the team's code

One submission for the whole team. Your submission must include a functional version of your Python program. That means the TA should be able to execute your program and use the interface provided without any additional files. This means your submission must include the three modules you previously developed as a team.

Summary of the deliverables for Milestone 2 – P4

- **P4 – Task 1.1:** Division of tasks
- **P4 – Task 1**: Implementation of your share of the user interface. You must submit a function version of the program.
- **P4 – Task 2:** Code review – Team submission – You must submit a function version of the program.