MILESTONE 1

This document is meant to be read in conjunction with the overall Project Description. It describes only those tasks associated with Milestone 1.

The first milestone 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. That way, 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 organize your time better.

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

Getting Started Problems

In the first milestone, it may happen that one of your teammates has not been communicating with your team. In this case:

- The team leader should provide the instructor with the name of the missing student. (If the team leader is absent, one of the other team members should do this).
- The team should complete the tasks for lab 1 but do not add the absent students' names in the contract and the team submission (as they did not participate in the drafting of that document or the team code).
- We will attempt to locate the student and assign an appropriate penalty after discussing the absence.

Contents

Milestone 1 Lab 1 (P1)

The main tasks to complete in your first lab period are as follows:

(1) team formation

(2) problem exploration

(3) reading a data set.

P1 Task 1: Team Formation

You have just met your new colleagues with whom you will work during this term. Get to know each other *informally* – Exchange phone numbers and/or emails. Tell each other about your backgrounds and your interests. There are good reasons to learn a bit about each other, such as:

- How do you like to communicate? (Find a communication method that works for everyone.)
- Do you have a job? (Some people may have time constraints.)

You must also get to know each other *formally* by collaborating to complete your Team Contract.

1. Download the ***Team Contract Template*** posted on Brightspace.
    - Required Filename: *Txxx_team_contract.pdf* where **xxx** is your team identifier (e.g., 021)
2. **Follow the instructions** within the contract, participate fully, spend time thinking about your answers, plan out all your meeting times and conflict resolution strategies. Should conflict occur, the instructors will be using this contract to assess the situation and decide how to intervene.
3. (**Team Submission**): Submit the electronic version of your team contract in Brightspace.
    - Every individual whose signature appears on the printed final page will earn the same mark.

P1 Task 2: Project Understanding and Project Exploration: Understanding Google Books Dataset

As a team, make sure that everyone watched the videos under "Course Project Overview" and (re-) read the "Project Overall Description." This document provides an overview of the project. You will not need all the information now, but the intention is to let you see the "big picture" so you understand what program you will ultimately develop.

Do you have any questions about the project? Are there ambiguities? Do you see any upcoming challenges for teamwork? You do not have to have all the answers now because some will come about as you begin to work but make sure that everyone in the team has a common understanding.

The data was acquired from the Google Books store. Google API was used to obtain the data. You do not need to do any steps to get the data; you just need to download the data file from Brightspace.

Nine features were included for each book in the data set. The column names are self-explanatory. Nevertheless, they are explained below.

- **title**: the title of the book.
- **author**: name of the authors of the books (might include more than one author).
- **language**: the language of the book
- **category**: the categories associated with the book (by Google store)
- **rating**: the rating of each book out of 5.
- **publisher**: the name of the publisher.
- **pages**: number of pages of the books.

There is **no submission** required. Please do the work for your own understanding. You will notice that the dataset contains the same book in different categories. You will see that there are duplicated entries. That is on purpose.

P1 Task 3: Reading the data set

In this task, your team will use the function design recipe to develop functions that can read the data set and store it in different formats using combinations of lists and dictionaries. Your team will write four functions. Each function will read the ***google_books_dataset.csv*** file (the file name will be the only input parameter of the function) and will return a dictionary or a list with the stored data based on the case you are implementing.

CASE 1: book_category_dictionary

In this case, the dictionary keys are the ***categories*** such as ***Fiction, Fantasy, Biography,*** etc. The function will store the dictionary using the following format. Note that the rating is a float number, and the pages attribute an integer number.

```
book_dictionary = {
   "Fiction":[ {"title": "Antiques Roadkill: A Trash 'n' Treasures Mystery",
                "author": " Barbara Allan",
                "language ": "English",
                "rating": 3.3,
                "publisher": " Kensington Publishing Corp.",
                "pages": 288
                },
                {another element},
                …
              ],
   "Biography":[ {"title": "The Nightshift Before Christmas: Festive hospital
                          diaries from the author of million-copy hit",
                 "author": " Adam Kay",
                 "language": "English",
                 "rating": 4.7,
                 "publisher": "Pan Macmillan",
                   "page": 112
                  },
                  {another element},
                  …],
   ….
}
```

CASE 2: book_publisher_dictionary

In this case, the dictionary keys are the ***publisher,*** such as ***Kensington Publishing Corp., Pan Macmillan, HarperCollins UK.,*** etc. The function will store the dictionary using the following format. Note that the rating is a float number, and the pages attribute an integer number.

```
book_dictionary = {
   "Kensington Publishing Corp.":[ {
                  "title": "Antiques Roadkill: A Trash 'n' Treasures Mystery",
                  "author": " Barbara Allan",
                  "language ": "English",
                  "rating": 3.3,
                  "category": "Fiction",
                  "pages": 288
                  },
                  {another element},
                  …
                ],
   " Pan Macmillan ":[ { "title": "The Nightshift Before Christmas: Festive
                          hospital diaries from the author of million-copy hit",
                   "author": " Adam Kay",
                   "language": "English",
                   "rating": 4.7,
                   "category": "Biography",
                   "pages": 112
                  },
                  {another element},
                  …],
   ….
}
```

CASE 3: book_author_dictionary

In this case, the dictionary keys are the ***authors*** such as ***Barbara Allan,*** etc. The function will store the dictionary using the following format. Note that the rating is a float number, and the pages attribute an integer number.

```
book_dictionary = {
  "Barbara Allan":[{"title":"Antiques Roadkill: A Trash 'n' Treasures Mystery",
                  "language ": "English",
                  "rating": 3.3,
                  "publisher": " Kensington Publishing Corp.",
                  "category": "Fiction",
                  "pages": 288
                  },
                  {another element},
                  …
                ],
   "Adam Kay":[ {"title": "The Nightshift Before Christmas: Festive hospital
                            diaries from the author of million-copy hit",
                 "author": " ",
                 "language": "English",
                 "rating": 4.7,
                 "publisher": "Pan Macmillan",
                 "category": " Biography ",
                 "page": 112
                 },
                {another element},
                …],
   ….
}
```

CASE 4: book_list

In this case, you will have a list with the book data stored as a dictionary. Note that the rating is a float number, and the pages attribute an integer number.

```
book_list = [{"title": "Antiques Roadkill: A Trash 'n' Treasures Mystery",
             "author": " Barbara Allan",
             "language ": "English",
             "rating": 3.3,
             "publisher": " Kensington Publishing Corp.",
             "category": " Fiction",
             "pages": 288},
             {"title": "The Nightshift Before Christmas: Festive hospital
                        diaries from the author of million-copy hit",
              "author": " Adam Kay",
              "language": "English",
              "rating": 4.7,
              "publisher": "Pan Macmillan",
              "category": " Biography ",
              "pages": 112,
             },
             {another element},
             …
           ]
```

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

<u>The leader of the team shall:</u>

1. Ensure that everyone understands the tasks you must complet. 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. Determine who is writing which function<u>. In teams of three (or, if there are absent teammates), you can ignore case 4. If there are two missing teammates, you can also ignore case 3.</u>
3. You need to submit the division of tasks (i.e., who is doing what). You can assign work to the missing team members in case they join late.

Each individual shall:

1. **Follow the FDR** to construct their assigned function.
2. You cannot use any external libraries other than: "`import string`" and "`from typing import List`". You must use what you learned in lecture 3. You can treat a csv file as a text file; just open it with Notepad instead of Excel to check it out.
3. For now, do not worry about duplicated entrances.
4. Required Filename: **Txxx_P1_yyy.py** where **xxx** is your team identifier and **yyy** = { book_category_dictionary, book_publisher_dictionary, book_author_dictionary, or book_list}
5. You must test your function. You can just call your function for this deliverable and ensure that the data is loaded correctly. You must submit the main script with the test as you did in lab0 of this course. That file should be named **Txxx_P1_main_yyy.py** where **xxx** is your team identifier and **yyy** = { book_category_dictionary, book_publisher_dictionary, book_author_dictionary, or book_list}
6. Your final version of your Python program file should contain (1) the function with type annotations and docstring (2) the main script in which the function is called. For your docstring, one example is good enough. Do not forget to include the preconditions.

Each ***INDIVIDUAL*** must submit their own work to be marked individually. Please refer to the Marking Rubric.

P1 Task 4: Code review

For the rest of the project, you will be working with the function `book_category_dictionary` (i.e., case 1).

As a team, ensure that the function works properly, it is written according to the coding conventions posted, and you are not using any external libraries other than: "`import string`" and "`from typing import List`"

Create a copy of that file and store it as **T**xxx_P1_load_data.py, where **xxx** shows your team identifier. At the top of the file, add a comment to include all the active team members' names. If a name is missing, that person will receive a zero. Make sure to state who wrote the function initially (for the individual submission) and who reviewed it (the other three team members).

Ensure that it still works as expected. You do not need to submit the test case for this function as a team.

Now, you have developed the first module of your project! You can load the data set.

Submit the team's code (i.e., **T**xxx_P1_load_data.py). **One submission for the whole team.**

Summary of the deliverables for Milestone 1 – P1

- **P1 – Task 1:** Team Contract – Team submission – One file per team
- **P1 – Task 3.1**: Division of Tasks for "reading the data set."
- **P1 – Task 3**: Reading the data set - Individual submission – Two files per student.
- **P1 – Task 4**: Code review – Team submission – One file per team

Milestone 1 Lab 2 (P2)

This week, you will start adding functionalities to your software to retrieve different data, add values to the dataset, remove values from the dataset, etc.

P2 Task 1: Add, Remove and Search Coding Exercise

The following is a list of 8 functionalities (explained as functions) that your software will have. You will divide the work as you did for Milestone 1 lab 1, but first, let's check the functions you will be developing.

Function 1: add_book

Use the function design recipe to develop a function named `add_book`**.** The function has two input parameters, (1) the dictionary where the book must be added and (2) a tuple argument that has **seven** values which are title, author, language, publisher, category, rating, and pages. The function adds the book to the dictionary and verifies that the book has been added. The function returns the updated dictionary and prints a message stating, "*`The book has been added correctly`*" or "*`There was an error adding the book`*".

Function 2: remove_book

Use the function design recipe to develop a function named `remove_book`**.** The function has three input parameters, (1) the dictionary from where the book must be removed, (2) the title of the book and (3) the category. The function returns the updated dictionary and prints a message stating, "*`The book has been removed correctly`*" or "*`There was an error removing the book. Book not found.`*".

Function 3: get_books_by_category

Use the function design recipe to develop a function named `get_books_by_category.` The function has two input parameters, (1) the dictionary where the data is stored and (2) the category to print. The function will return the number of books in that category. Additionally, it will print:

```
The category zzz has xxx books. This is the list of books:
Book 1: "Title" by "Author"
Book 2: "Title" by "Author"
…..
```

Function 4: get_books_by_rate

Use the function design recipe to develop a function named `get_books_by_rate`**.** The function has two input parameters, (1) the dictionary where the data is stored and (2) a positive integer argument which is the rate. The function returns the number of books (note that the same book in a different category counts as a single book) for the given rate. For example, if the input parameter for the rate is 3. The function will return the number of books whose rate is greater or equal to 3 and smaller than 4. Additionally, it will print the information of all the books for the given rate as follows:

```
There are xxx books whose rate is between zzz and yyy. This is the list
of books:
```

```
Book 1: "Title" by "Author"
Book 2: "Title" by "Author"
…..
```

Function 5: get_books_by_title

Use the function design recipe to develop a function named `get_books_by_title`**.** The function has two input parameters, (1) the dictionary where the data is stored and (2) the book's title. The function returns a Boolean variable which is True if the title exists in the dictionary; otherwise, it returns False. Additionally, the function prints a message stating, "*The book has been found*" or "*The book has NOT been found*".

Function 6: get_books_by_author

Use the function design recipe to develop a function named `get_books_by_author`**.** The function has two input parameters, (1) the dictionary where the data is stored and (2) the author's full name. The function returns the number of books written by the author (note that the same book in a different category counts as a single book). Additionally, the function prints all the titles by the given author as follows:

```
The author zzz has published the following books:
Book 1: "Title", rate: "rating"
Book 2: "Title", rate: "rating"
….
```

Function 7: get_books_by_publisher

Use the function design recipe to develop a function named `get_books_by_publisher`**.** The function has two input parameters, (1) the dictionary where the data is stored and (2) the publisher's name. The function returns the number of books published by the given publisher's (note that the same book in a different category counts as a single book). Additionally, the function prints the books information for that publisher as follows:

```
The publisher zzz has published the following books:
Book 1: "Title" by "Author"
Book 2: "Title" by "Author"
```

Function 8: get_all_categories_for_book_title

As will have already noticed, the same book can be listed under different categories. Use the function design recipe to develop a function named *`get_all_categories_for_book_title`***.** The function has two input parameters, (1) the dictionary where the data is stored and (2) a book title. The function returns the number of categories associated with the given title. Additionally, the function prints the following information:

```
The book title zzz belongs to the following categories:
Category 1: "Category"
Category 2: "Category"
…
```

As you can see, all the functions take a dictionary as an input parameter that is a dictionary which your function in P1-Task 4 returns!

The leader of the team shall:

1. Ensure that everyone understands the task to 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 function.
3. You need to submit the delegation of tasks. As mentioned, there are 8 functions and, in most teams, there are four members, so each student will write 2 functions. If you are a team of three (or, if there are absent teammates), two students will need to write 3 functions.

Each individual shall:

1. **Follow the FDR** to construct their assigned function.
2. You cannot use any external libraries other than: "`import string`" and "`from typing import List`".
3. Remember **not to include** duplicated entrances in your counts (return output values of the function) and your print statements.
4. Required Filename: **Txxx_P2_fun.py** where **xxx** is your team identifier.

Each ***INDIVIDUAL*** must submit their own work to be marked individually. Please refer to the Marking Rubric.

P2 Task 2: Unit Testing Using Modules

In the previous task, each team member has implemented two functionalities add/remove data from the datasets or search for data. In this task, you will use modules to develop the automated unit tests for your functions.

You need to test each function individually as explained in the lecture "Unit testing". The test harness will be developed before the function is implemented. Remember to use automated testing as you learn in ECOR1041. Do not worry about the print statements in your tests, just focus on the returned value of the function.

For your tests, it would be a good idea to use a smaller data set. You can manually create dictionaries for your "test data set" or you can create *test_xxx.csv* files that you load using the function developed in P1. In the latter case, remember to import the module developed in P1, so your lab runs without errors.

The leader of the team shall:

1. Ensure that everyone understands the task to 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 each test. To ensure the test harness is developed before the functions are implemented, none of the students can write the tests for the functions they developed in the previous task. Each student should write the tests for 2 functions. If you are a team of three (or, if there are absent teammates), two students will need to write 3 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. Develop the test for their assigned functions. You must test your functions programmatically following the practices taught in the unit testing lecture and ECOR1041.
2. Required Filename: **Txxx_P2_test.py** where **xxx** is your team identifier

Each ***INDIVIDUAL*** must submit their own work to be marked individually. Please refer to the Marking Rubric.

P2 Task 3: Milestone 1's Final Team Code

As a team, package your code together.

1. Create a new Python file with all the functionalities. Required Filename: **Txxx_P2_add_remove_search_dataset.py,** where **xxx** is your team identifier**.**
2. Copy-paste everyone's function into this one file.
3. After all the functions, copy-paste everyone's tests and make sure you have the correct import modules at the top of the file. You will notice that you may have duplicated testing function (e.g., more than one "test_int" function). Simplify your code to avoid duplications.
4. Make sure that your code still runs without errors and all test passes.
5. 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, etc.
6. Ensure that the main script only runs when it is executed from the module. Review Chapter 6 section "Selecting Which Code Gets Run on Import: __main__" from the coursebook.
7. Make sure it still compiles and runs.

Your file should be organized as follows:

```
#imports

#The eight functions from P2 - Task 1

#The test functions you developed for automated testing

#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.
```

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

- Txxx_P2_add_remove_search_dataset.py

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

Summary of the deliverables for Milestone 1 – P1

- **P2 – Task 1.1:** Division of tasks
- **P2 – Task 1**: Implementation of functionalities - Individual submission – One files per student
- **P2 – Task 2.1:** Division of tasks
- **P2 – Task 2:** Implementation of testing - Individual submission – One files per student
- **P2 – Task 3:** Code review – Team submission – One file per team

Milestone 1 Marking Rubrics

P1 Task 1: Team Contract

Satisfactory (2/2)	Marginal (1/2)	Unsatisfactory (0/2)
The contract is complete. The answers to the questions are thoughtful. The document is tidy and well-written	The contract is incomplete, or the answers are mostly superficial and short. The document is untidy and shows a lack of care, making it difficult to read	It is late or the contract is missing

Coding Tasks (P1 Task 3, P1 Task 4, P2 Task 1, P2 Task 2, P2 Task 3)

Preconditions

A student's work will not be marked (i.e., will earn UNSATISFACTORY=0) if **any** of the following are true:

- The files are **not in the proper .py** format (no .pdf, zip or py.py file extensions)
- The requested files are either not present or do not run without errors
- The file's **author** is **not identified** in the heading of the file. (50% penalty for individual submissions)
- If applicable, they use programming techniques that have been explicitly discouraged for the course for teaching reasons.

Documentation Expectations

- All functions headers must have type annotations on all arguments, and on the return type.
- All functions must have a good docstring describing WHAT the functions do, not HOW.
  - Good Example: Returns the average of the three integer parameters.
  - Poor Example: The function returns the average of three values.
  - Poorest Example: The function adds the three numbers and returns their average.
- The docstring must include at least **two** examples and the preconditions.
- When applicable, the functions must have a thoughtful set of tests in the proper shell format.

Coding Expectations

- Variables and functions use meaningful names, in the proper style.
- The files should be organized following the guidelines provided.
- The code is properly indented and has comments identifying the different sections and the authors.
- Constants must be used instead of hard-coded values, in the proper style.

Appendix A: Coding "Do's and Don'ts"

Keywords and Statements

- Your code can have statements that use these Python keywords (reserved words): False, None, True, and, def, del, elif, else, for, from, if, import, in, not, or, pass, return, while.
- Your code cannot have statements that use these keywords: as, assert, asynch, await, break, class, continue, except, finally, global, is, lambda, nonlocal, raise, try, with, yield.
- Your code cannot use list comprehensions, generator definitions and generator expressions. (None of these constructs are taught in ECOR 1042).

Container/Collection Types

- Your code can use the built-in container types list, tuple dictionary and set.

Modules

- Your code can use the following modules that are provided with a Python installation (the "standard" library): math, string, typing
- Your code cannot use any module from the standard library or if it is provided by a third-party, other than the ones listed above. If you think another module is required, this must be approved by one of the course instructors, who will update the list of permitted modules.

Style

- Your code must adhere to the "official" style guide for Python code (PEP 8), which can be found here: https://www.python.org/dev/peps/pep-0008/. Alternately, you can follow the style used in the course textbook "Practical Programming", which is very close to PEP 8.