CP1404/CP1804/CP5632 – Assignment 2

Travel Tracker 2.0 (Complete)

Task:

Create both a console program and a Graphical User Interface (GUI) program similar to your first assignment, using Python 3 and the Kivy toolkit, as described in the following information and accompanying screencast video. This assignment will help you build skills using **classes** and **GUIs** as well as giving you more practice using techniques like selection, repetition, exceptions, lists, dictionaries, file I/O and functions. **Some requirements have in-text help references, like [0]**, that refer to the resources list near the bottom of this document. Please check these references to find help on that topic. ***Everything*** you need to complete this assignment can be found in the subject materials.

Start your work by clicking this link to create a new repository in GitHub classroom:

https://classroom.github.com/a/rQeX3xi3

Do not use any other repo or a copy of this one... just use this actual repository!
This will give you a new repo containing starter code files and a README for your project reflection, all of which you must use. Do not add any other files in this project, and do not rename anything - just use this as your assignment repo. Do not "download" this repo, but rather *checkout* this repo using PyCharm ("Get from Version Control").

Classes:

The most important learning outcome of this assignment is to be able to use **classes** to create reusable data types that simplify and modularise your programs. The fact that you can use these classes in both console and GUI programs highlights this modularity.

It is important that you ***create these classes first* – before any code that requires them**. This is good coding practice. You should write and then test each method of each class – **one at a time – committing as you go** (e.g. you might commit each time you complete a method and its tests).

The starter code includes two files (test_place.py and test_placecollection.py) with incomplete code for testing your classes. **Complete these files with simple tests**, that you write as you develop your Place and PlaceCollection classes.

Do not change the existing tests… write code that makes these tests pass.

You may use `assert` as shown in lectures [1], or just very simple tests that print the results of calling the methods you are testing with expected and actual results [2].

Once you have written and tested your classes, you can then use the Place class in your console program.

We will assess your Git commit history to see (and mark) that you do these in an appropriate order, so make sure you write your classes, with tests, then the console program, *before* attempting any functionality for the GUI.

- Complete the **Place** class in place.py.
  This should be a simple class with the required attributes for a place (name, country, priority and visited status) and the methods:
    - __init__
    - __str__
    - two methods to mark the place as unvisited/visited
    - a method to determine if a place is considered "important", which is defined as having a priority <= 2.

- Complete the **PlaceCollection** class in placecollection.py.
  This class should contain a *single* attribute: a list of Place objects, and at least the following methods:
    - load places (from csv file into Place objects in the list)
    - save places (from place list into csv file)
    - add place – add a single Place object to the places attribute
    - get number of unvisited places
    - sort (by the key passed in, then by priority) (attrgetter from [3])

  Do not store any additional attributes in this class. (E.g. you don't need to store the number of places because this information is easily derived from what you do store.)

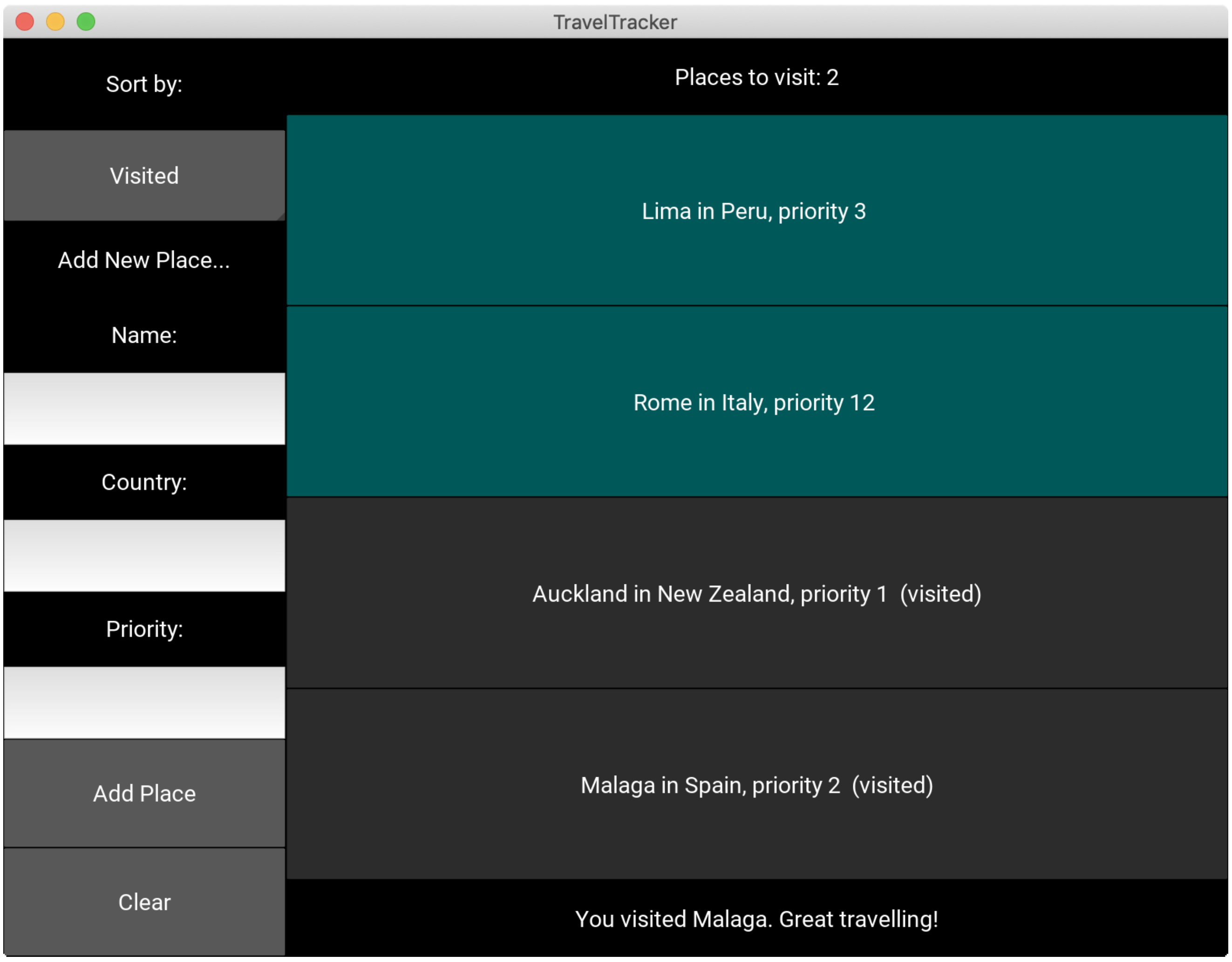
Remember to test your classes before moving on to using them in the following programs.

Console Program:

After you have written and tested your classes, rewrite your first assignment to make use of your new Place class. Optionally, you may also use the PlaceCollection class in your console program. Start by copying the code from your first assignment into the existing a1_classes.py file. In the first assignment, each place was stored as a list. Modify your code so that each place is stored as an object of your new Place class.

You do *not* need to rewrite your first assignment in any other way, even if it had problems. We will only evaluate how you use the Place class in the console program.

GUI Program:



Places to visit: 2

Lima in Peru, priority 3

Rome in Italy, priority 12

Auckland in New Zealand, priority 1 (visited)

Malaga in Spain, priority 2 (visited)

You visited Malaga. Great travelling!

(The display and functionality explained here is also shown in the screencast video demo.)

The GUI program should have the following features, as demonstrated in the accompanying screencast video. Complete the main program in a Kivy App subclass in main.py. [4, 5]

- The data (model) for this program should be a single PlaceCollection object.
- The program should load the CSV file of places using the method from PlaceCollection.
- The places file must be saved (there's a method for that!) when the program ends, updating any changes made with the app by the user (adding new places or marking them as un/visited) (`on_stop` method from [5]).
- The left side of the screen contains a drop-down "spinner" for the user to choose the sorting (see spinner_demo from [4]), and text entry fields for inputting information for a new place. You might like to consider using a **dictionary** to help with mapping the GUI text to the attributes of the class to be sorted by.
- The right side contains a button for each place, colour-coded based on whether the place is visited or not (the actual colour scheme is up to you).
- The status bar at the top shows the number of places still to visit.
- The status bar at the bottom shows program messages.
- When the user clicks on a place button, the place changes between visited and unvisited (see guitars_app.py from [4] for an example of using Kivy with custom objects associated with buttons).
- The text to display when a place button is clicked is like (where "name" is the actual place name):
    - You need to visit name.
    - You visited name.

  For important places, it should be like:
    - You need to visit name. Get going!
    - You visited name. Great travelling!

Adding Places (Error Checking):

- The user can add a new place by entering text in the input fields and clicking “Add Place”.
- All place fields are required. If a field is left blank, the bottom status label should display “**All fields must be completed**” when “Add Place” is clicked.
- The priority field must be a valid integer. If this is invalid (and no other fields are empty), the status bar should display “**Please enter a valid number**”.
- If priority is less than 1, the status label should display “**Priority must be > 0**”.
- Pressing the Tab key should move between the text fields. (popup_demo from [4])
- When the user successfully adds a place, the entry fields should be cleared and the new place button should appear on the right. (dynamic_widgets from [4])
- When the user clicks the “Clear” button, all text in the input fields and the status label should be cleared.

General Coding Requirements:

- At the very top of your main.py file, complete the comment containing your details.
- Document all of your classes and methods clearly with docstrings. Include inline/block comments as appropriate. You do not need comments in the kv file.
- Make use of named constants where appropriate. E.g. colours could be constants.
- Use functions/methods appropriately for each significant part of the program. Remember that functions should follow the Single Responsibility Principle.
- Use exception handling where appropriate to deal with input errors. When error checking inside functions (e.g. a handler for clicking the Add Place button), you might like to consider the "Function with error checking" pattern from [6].
- Complete your GUI design using the kv language in the app.kv file. Creating the place buttons should be done in main.py, not in the kv file, since this will be dynamic (dynamic_widgets from [4]).

Project Reflection:

It is important and beneficial for you to start developing good coding and working practices, so you will complete a short but thoughtful reflection on this project. Complete the template provided in the README and reflect on what you learned regarding both coding and your development process. **This is worth significant marks, so allocate significant time to it.** We expect answers that show some **detail** and **thought**, not just trivial statements.

Git/GitHub:

You must use Git version control with your project stored in the private repository on GitHub that will be created when you accept the GitHub Classroom invitation above.
You are assessed on your use of version control including commits and commit messages, using the **imperative voice** (like "Add X" not "Added X"). [7]

Submission:

Submit a single zip file by uploading it on LearnJCU under Assessment (click on the title of the assignment). Your zip file should contain the entire project directory, including the .git directory (just zip up your project directory). Make sure your GitHub URL is included in main.py. Please name the zip file like: **FirstnameLastnameA2.zip.**

Due:

Submit your assignment by the date and time specified on LearnJCU. Submissions received after this date will incur late penalties as described in the subject outline.

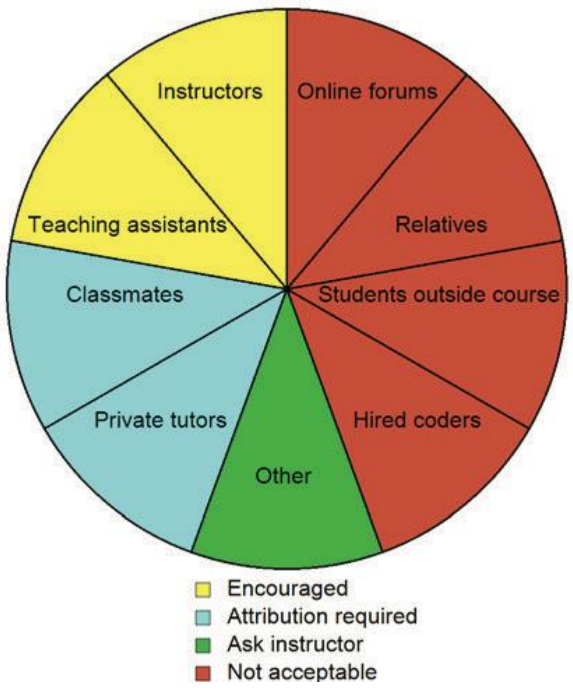
Integrity:

The work you submit for this assignment **must be your own**. Submissions that are detected to be too similar to that of another student will be dealt with according to the College procedures for handling plagiarism and *may result in serious penalties*.

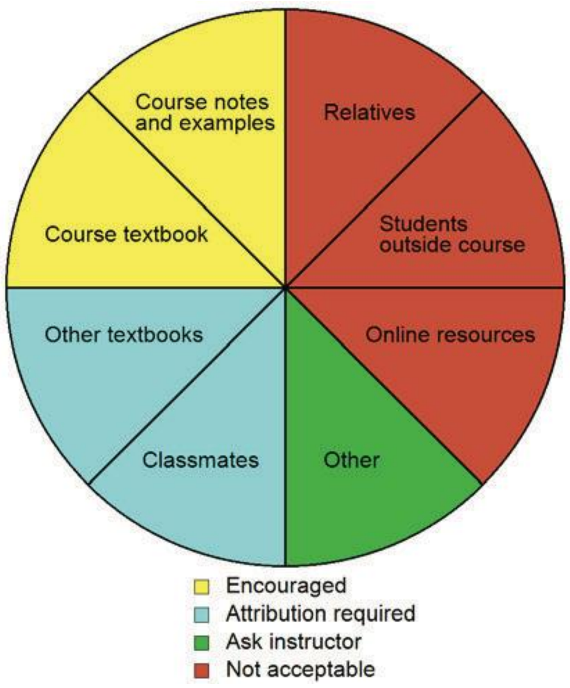
The goals of this assignment include helping you gain understanding of fundamental programming concepts and skills, and future subjects will build on this learning. Therefore, it is important that you develop these skills to a high level by completing the work and gaining the understanding yourself. You may discuss the assignment with other students and get assistance from your peers, but you may not do any part of anyone else’s work for them and you may not get anyone else to do any part of your work. Note that this means you should **never give a copy of your work to anyone or accept a copy of anyone else’s work, including looking at another student's work or having a classmate look at your work.** If you require assistance with the assignment, please ask **general** questions in #cp1404 on Slack, or get **specific** assistance with your own work by talking with your lecturer or tutor.

The subject materials (lecture notes, practicals, textbook and other guides provided in the subject) contain ***all*** of the information you need for this assignment. You should not use online resources (e.g. Stack Overflow or other forums) to find resources or assistance as this would limit your learning and would mean that you would not achieve the goals of the assignment - mastering fundamental programming concepts and skills.

Assistance: Who can you get help from? Use this diagram to determine from whom you may seek help with your programs. [8]



Resources: Where can you get code from? Use this diagram to determine where you may find code to use in your programs.



Sample Output:

Study the screencast provided with this assignment to see how the GUI program should work, including what the messages should be and when they occur.

References – Resources from Subject Materials:

1. Lecture Notes for Chapter 15 - Testing.
2. Practical 06 - Classes. https://github.com/CP1404/Practicals/tree/master/prac_06
3. Lecture Notes for Chapter 11 - Classes
4. KivyDemos. https://github.com/CP1404/KivyDemos
5. Lecture Notes for Kivy.
6. Programming Patterns. https://github.com/CP1404/Starter/wiki/Programming-Patterns
7. Lecture Notes for Version Control.
8. Negotiating the Maze of Academic Integrity in Computing Education. https://dl.acm.org/citation.cfm?doid=3024906.3024910

Marking Scheme:

Ensure that you follow the processes and guidelines taught in class in order to produce high quality work. Do not just focus on getting the program working. This assessment rubric provides you with the characteristics of exemplary down to very limited work in relation to task criteria.

Criteria	**Exemplary (9, 10)**	**Good (7, 8)**	**Satisfactory (5, 6)**	**Limited (2, 3, 4)**	**Very Limited (0)**
Project reflection ***15%***	The project reflection is complete and describes development and learning well, shows careful thought, highlights insights made during code development.	Exhibits aspects of exemplary (left) and satisfactory (right)	Project reflection contains some good content but is insufficient in coverage, depth or insight.	Exhibits aspects of satisfactory (left) and very limited (right)	Many aspects of the project reflection are missing or could be improved.
Use of version control **10%**	Git/GitHub has been used effectively and the repository contains a good number of commits with good messages that demonstrate incremental code development **starting with classes and testing then console before GUI**.		Git/GitHub used but several aspects of the use of version control are poor, e.g. not enough commits, or meaningless messages that don't represent valuable incremental development in an appropriate order.		Git/GitHub not used.
Console program **8%**	Class is used correctly in console program.		Class is used in console program but not correctly.		Class is not used in console program.
Error handling **8%**	Errors are handled correctly and robustly as required.		Some errors are handled but not all, or errors are not handled properly.		No reasonable error handling.
Correctness ***16%***	GUI layout is correct and program works correctly for all functionality required.		Aspects of the GUI layout are incomplete or poorly done or there are significant problems with functionality required.		GUI layout is very poor or not done. Program works incorrectly for all functionality required.
Identifier naming **10%**	All function, variable and constant names are appropriate, meaningful and consistent.		Several function, variable or constant names are not appropriate, meaningful or consistent.		Many function, variable or constant names are not appropriate, meaningful or consistent.
Use of code constructs **12%**	Appropriate and efficient code use, including no unnecessary duplication, good logical choices for control and storage, good use of constants, no global variables, good use of functions in main app, etc.		Several problems, e.g. unnecessary duplication, poor control, no use of constants, improper use of global variables, poor use of functions in main app.		Many problems with code use.
Use of classes and methods **10%**	Classes and methods are used correctly as required. Method inputs and outputs are well designed.		Some aspects of classes and methods are not well used, e.g. methods not used where they should be, problems with method/parameter design, incorrect use of objects.		Classes and methods used very poorly or not used at all.
Commenting **6%**	Code contains helpful # block comments, all classes and methods have meaningful docstrings and main module docstring contains all details (name, date, basic description, GitHub URL).		Comments are reasonable, but some classes and methods have no docstrings, and/or there is some noise (too many comments), and/or missing details in main module docstrings.		Commenting is very poor or not done.
Formatting **5%**	All formatting is appropriate, including indentation, horizontal spacing and vertical line spacing. PyCharm shows no formatting warnings.		Problems with formatting reduces readability of code. PyCharm shows multiple formatting warnings.		Readability is poor due to formatting problems. PyCharm shows many formatting warnings.