CP1404 – Assignment 2 – Movies to Watch 2.0 (Complete)

JAMES COOK UNIVERSITY AUSTRALIA

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, file I/O and functions. **Some requirements have in-text help references, like [0]**, that refer to the resources list near the bottom. 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/EbcWiXFO

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.

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).

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.

The starter code includes two files (test_movie.py and test_moviecollection.py) with incomplete code for testing your classes. **Complete** these files with simple tests, that you write as you develop your Movie and MovieCollection 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 Movie class in your console program.

- Complete the **Movie** class in movie.py. This should be a simple class with the required attributes for a movie and the standard methods: __init__ (constructor), __str__ (used when displaying movie details in the status message), and:
  - two (not one) methods, to mark the movie as watched or unwatched.

- Complete the **MovieCollection** class in moviecollection.py. It should contain a *single* attribute: a list of Movie objects, and at least the following methods:
  - add movie – add a single Movie object to the movies attribute
  - get number of unwatched movies
  - get number of watched movies
  - load movies (from csv file into Movie objects in the list)

- save movies (from movie list into csv file)
- sort (by the key passed in, then by title) [3]

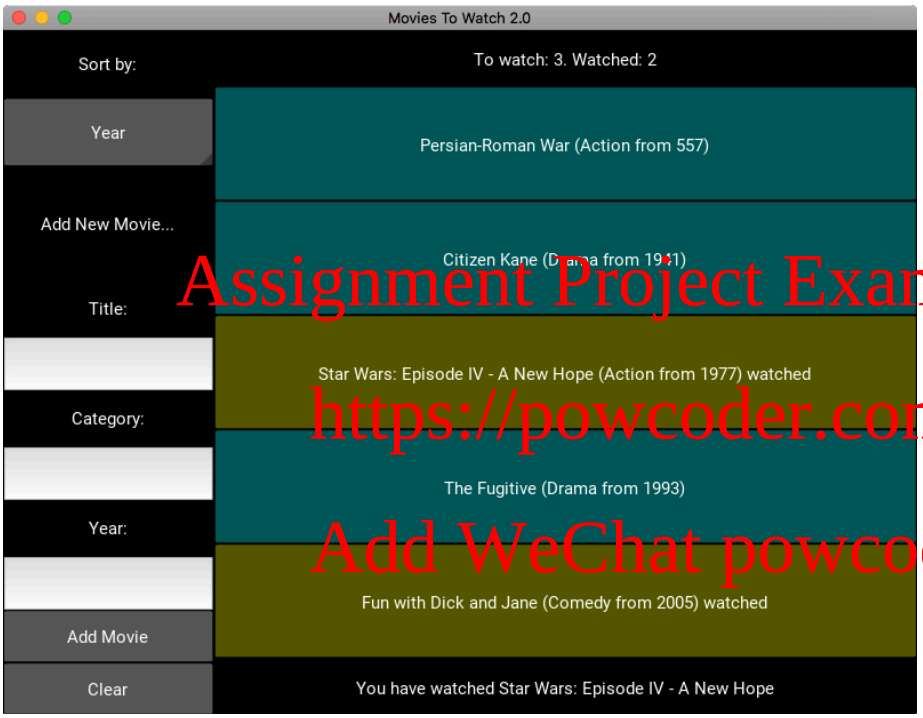
Notice that you do not store additional attributes like the number of movies, because this information is easily derived from what you do store (just the movies) [4].

Console Program:

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

GUI Program:



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

- Complete the main program in a Kivy App subclass in main.py. There will be no main() function, but rather your program will run() the Kivy app in the same way as you've seen in our example programs. [5, 6]
- The program should start by loading the same CSV file of movies as with your first assignment. This must be done within a method of your main app class using the appropriate method from your MovieCollection class.
- The movies 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 movies or marking them as watched) (on_stop method from [6]).

Ensure that your program GUI has the following features/functionality, as demonstrated in the accompanying screencast video.

- The left side of the screen contains a drop-down "spinner" for the user to choose the movie sorting (see spinner_demo from [5]), and text entry fields for inputting information for a new movie.
- The right side contains buttons for the movies, colour-coded based on whether they are watched or not (the colour scheme is up to you, but the colours must be different).

- The status label at the top of the right side shows the number of movies watched and still to watch.
- The status label at the bottom of the right side shows messages about the state of the program, including updating when a movie or other button is clicked on.
- When the user clicks on a movie button, the state of the movie changes between watched and unwatched (see guitars_app.py from [5] for an example of using Kivy with custom objects associated with buttons).
- The user can add a new movie by typing in the input fields and clicking “Add Movie”.

Adding Movies (Error Checking):

- All movie fields are required. If any field is left blank, the bottom status label should display “**All fields must be completed**” when “Add Movie” is clicked.
- The year field must be a valid integer. If this is invalid, the status label should display “**Please enter a valid number**”.
- The category must be one of the following: Action, Comedy, Documentary, Drama, Fantasy, Thriller
- Pressing the Tab key should move between the text fields. (popup_demo from [5])
- When the user successfully adds a movie, the text fields should be cleared and the new movie button should appear in the movies list on the right. (dynamic_widgets from [5])
- When the user clicks the “Clear” button, all text in the input fields and the status label should be cleared.

See the screencast video for a demo of this in action.

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 button), you should consider the "Function with error checking" pattern from [4].
- Complete your GUI design using the kv language in the app.kv file. Creating the movie buttons should be done in main.py, not in the kv file, since this will be dynamic. (dynamic_widgets from [5])

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.

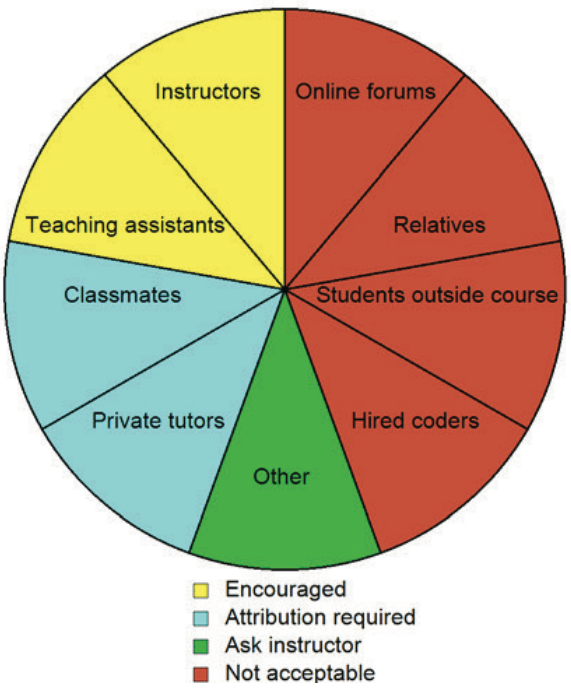
Integrity:

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*.

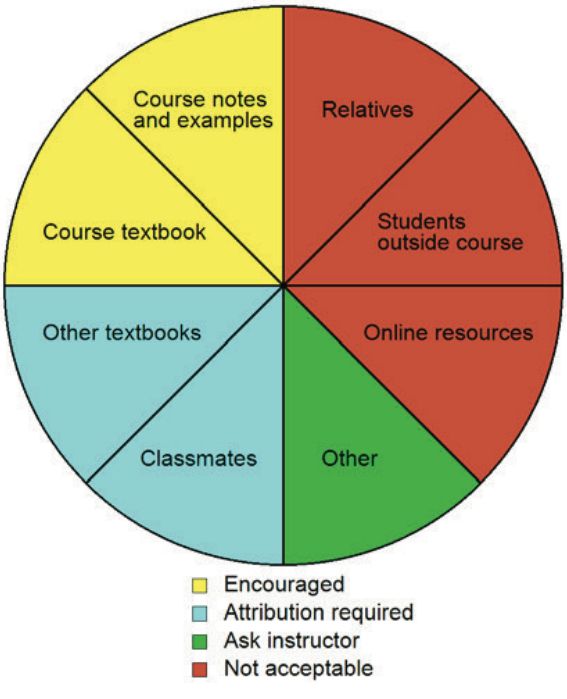
The goals of this assignment include helping you gain understanding of fundamental programming concepts and skills, and future subjects will build on this learning. Therefore, it is important that you develop these skills to a high level by completing the work and gaining the understanding yourself. 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 talk 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.
The console program should look identical to the requirements for Assignment 1; only the implementation has changed.

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. Programming Patterns. https://github.com/CP1404/Starter/wiki/Programming-Patterns
5. KivyDemos. https://github.com/CP1404/KivyDemos
6. Lecture Notes for Kivy.
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, 1)
Project reflection ***14%***	The project reflection is complete and describes development and learning well, shows careful thought, highlights insights made during code development.		Project reflection contains some good content but is insufficient in coverage, depth or insight.		Many aspects of the project reflection are missing or could be improved.
Use of version control **9%**	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 **9%**	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 **9%**	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 ***14%***	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 **9%**	All function, variable and constant names are appropriate, meaningful and consistent.	Exhibits aspects of exemplary (left) and satisfactory (right)	Several function, variable or constant names are not appropriate, meaningful or consistent.	Exhibits aspects of satisfactory (left) and very limited (right)	Many function, variable or constant names are not appropriate, meaningful or consistent.
Use of code constructs **9%**	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 **9%**	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 **9%**	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 **9%**	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.