# CP1404 Assignment 2: Movies to Watch 2.0 Requirements

## Note:

This document is only meant to be a helpful reference for the assignment requirements stored right here with your code.
The PDF provided contains the complete details.
Any discrepancy is accidental - and the PDF is the correct and complete version of the assignment requirements.

## 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.
***Everything*** you need to complete this assignment can be found in the subject materials.

## 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
- 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 (testing, chapter 15), or just very simple tests that print the results of calling the methods you are testing with expected and actual results (as in practical 06).

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

## Movie

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.

## MovieCollection

Complete the **MovieCollection** class in moviecollection.py. It should contain a *single* attribute: a list of Movie objects, and at least the following methods:

- the standard constructor and str 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) (see `attrgetter` from the lecture on Classes)

  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) (see Data Storage pattern/principle)

# Console Program:

***After*** you have written and tested your classes, rewrite your first assignment to make use of your new Movie class. Optionally, you may also use the MovieCollection class in your console program. 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.

# 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.
- 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. See dynamic_widgets (py and kv)

**GUI Program details, marking guide, etc. are all in the assignment PDF.**
**Save a copy and keep it with you when you're working.**

# 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.md 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.

# References - Resources from Subject Materials:

Don't forget to learn all you can from:

- Lectures and lecture notes (LearnJCU)
- Practicals
- Program Patterns "Cheat Sheet" and other helpful guides at that wiki
- Kivy Demos repository
- Comments on potential problems (you can also potentially benefit from reading what *not* to do)