P01: Final Project

- Your final project may be completed alone or in pairs.
- Your task in this final project is to expand code created earlier in the term in a direction of your own choosing and using an appropriate object-oriented programming structure.

Learning Objectives

- Synthesize elements of learning from the entire course into a single, final project.
- Create a GUI to wrap your project.
- Work in an Agile fashion.

How to Begin

- To begin, make a copy of this document by going to File >> Make a Copy...
- Share the copied document with your partner. You can share this document by hitting the blue button in the top right of the document, then entering their email address in the bottom input field.
- Change the file name of this document to **username1, username2 - P01: Final Project** (for example, **heggens, wilborned - P01: Final Project**). To do this, click the label in the top left corner of your browser.
    - You will be asked to create a team. *If you are working alone, create a team anyways, with just you in it:*
        - One of you will create the team.
        - Then, the other will join the team created by the first partner.

Background

In this course, you've learned about the following programming principles:

- *Chapter 1 General Introduction*
- *Chapter 2 Simple Python Data*
- *Chapter 3 Debugging Interlude I*
- *Chapter 4 Python Turtle Graphics*
- *Chapter 5 Python Modules*
- *Chapter 6 Functions*
- *Chapter 7 Selection*
- *Chapter 8 More About Iteration*
- *Chapter 9 Strings*
- *Chapter 10 Lists*
- *Chapter 11 Files*
- *Chapter 12 Dictionaries*
- *Chapter 13 Exceptions*
- *Chapter 15 GUI and Event Driven Programming*
- *Chapter 17 Classes and Objects - The Basics*

- *Chapter 18 Classes and Objects - Digging a Little Deeper*
- *Chapter 19 Inheritance*

It's time to put all this great knowledge to use!

The Requirements

- **Code reusability is good**! This project gives you the opportunity to choose something from the course you really enjoyed and expand upon it. In particular, you must enhance a program you or your partner has made in the course, and/or build off of code which we have provided to you.
- To this end, you must **design one or more classes** that will effectively reimplement your old code.
- For this project, you need to be **interactive**! Your project *must* use either the Turtle library, Pygame, or the Tkinter module to make the interactivity with the user interesting and fun. Feel free to use more than one together!
- Feel empowered to be creative! Your program or game should be interactive as well as interesting both to create and to use.

Milestones

To ensure that you're making continued progress for the next few weeks, there are four milestones where we'll be checking up on you:

1. (11/13) **Milestone 1**: Initial planning, design, setup, delegation of tasks (for partners)
2. (11/20) **Milestone 2**: Code setup, first version of code, initial testing
3. (12/04) **Milestone 3**: Nearly finished second version of code, expanded test suite
4. (Final Exam Period) **Milestone 4**: Final version of product

Milestone 1 - due Wednesday, November 13th, 2024

First, join the **Github classroom**.
Your first task is to build a design document. You'll use the **README.md** file to do this. An example design document with instructions for each section is included in your repository in **README.md**. Be sure to look at the source code AND how it's rendered on Github (as they are not the same). You will be graded on the rendered version, NOT the source code version, so check your formatting! **This file includes lots of help text for how things should be done; read it thoroughly and carefully!** Github uses a styling language called markdown (hence, the .md extension). A nice cheat sheet for how to format markdown is included here. You'll need to refer back to it often to do certain formatting tricks, like adding images.
Next, look back at the previous teamworks, homeworks, and examples you've seen. You'll be building off of these assignment(s) as a final product. In some cases, you'll be combining or taking knowledge from multiple assignments.

CSC 226: Software Design & Implementation
P01: Final Project

List each of the files you'll be using as resources in the "**References**" section of the design document you're building in the README.md file.

Open your repository in PyCharm, and bring in all of the code you listed above into your new repository for P01. **Do not modify this original code!** More on this later...

NOTE: Students often forget about the write-up by the end of the project, and try to do it at the very last minute. The result is usually lots of grammatical errors, which do count against you. Instead, we encourage you to take breaks from coding by updating the **README.md** as you go. That will both help you take breaks from coding (which are useful in preventing exhaustion from trying to solve those annoying bugs!) as well as ensure your document is in good shape by the end of the project. It also gives you clarity about your program, as you try to explain it in the document.

IDEA: library catalog where user can add books, search by author and/or genre, and track read/unread status
-use classes to build for book entries (properties: genre, title, author)
-use tkinter to display books in GUI

Class name: LibraryCatalog	
Instance Attributes:	**Class Collaborations (other classes):**

• `books`:
  ○ a list to store all book instances, representing the catalog
• `self.file_path`:
  ○ stores the file path for loading and saving data
	• `books` : a list that holds instances of the `Book` class (this class depends on the `Book` objects to populate and mangage the catalog)

Class Methods:	**Class Collaborations (other classes):**

• `__init__(self, file_path)`:
  ○ a constructor for the LibraryCatalog class
• `add_book(self,book)`:
  ○ adds a new `book` object to the catalog
• `search_by_author(self, author_name)`:
  ○ searches the catalog for books by a specific author and returns a list of matching books
• `search_by_genre(self, genre_name)`:
  ○ searches the catalog for books of a specific genre and returns a list of matching books
• `recommend_book(self, genre=None)`:
  ○ recommends a book based on a given genre or author, returning relevant `book` object if available
• `load_catalog(self)`:
  ○ loads book data from the file at `file_path` and populates `books` with `Book` instances.
• `save_catalog(self)`:
  ○ saves the current list of books to the file at `file_path`
	• `add_books(self,book)`: accepts an instance of the `Book` class as an argument to add it to the `books` list
	• `search_by_author(self,author_name)` and `search_by_genre(self,genre_name)`: methods search through `books` which contains `Book` objects, and return matching `Book` instances; `LibraryApp` will call these methods to fetch books based on user search criteria
	• `recommend_book(self, genre=None, author=None)`: finds a `Book` object based on a specified genre or author and returning it to `LibraryApp`
	• `load_catalog(self)` and `save_catalog(self)`: load and save `Book` data to/from a file

Class name: Book	
Instance Attributes:	**Class Collaborations (other classes):**

• `self.title`:
  ○ stores the title of the book
• `self.author`:
  ○ stores the name of the author
• `self.genre`:
  ○ stores the genre or category of the book
• `self.status`:
  ○ tracks whether the book has been read or is unread
	• None

Class Methods:	**Class Collaborations (other classes):**

• `__init__(self, title, author, genre, status)`:
  ○ initializes a new book with provided title, author, genre, and unread/read status
• `update_status(self, new_status)`:
  ○ updates the `status` attribute to a new value, allowing user to mark book as read/unread
• `get_info(self)`:
  ○ returns a formatted string with the book's title, author, genre, and status, which can be used in the catalog's display
	• None

Class name: LibraryApp	
Instance Attributes:	**Class Collaborations (other classes):**

• `self.root`:
  ○ tkinter (root) window
• `self.catalog`:
  ○ an instance of `LibraryCatalog` for managing book data
• `self.display_frame`:
  ○ frame widget in the GUI to display list of books and search results
• `self.search_field`:
  ○ field for user input to search by author or genre
• `self.status_filter`:
  ○ dropdown (radio button) to select read/unread filter options
	• `catalog` : part of `LibraryCatalog`

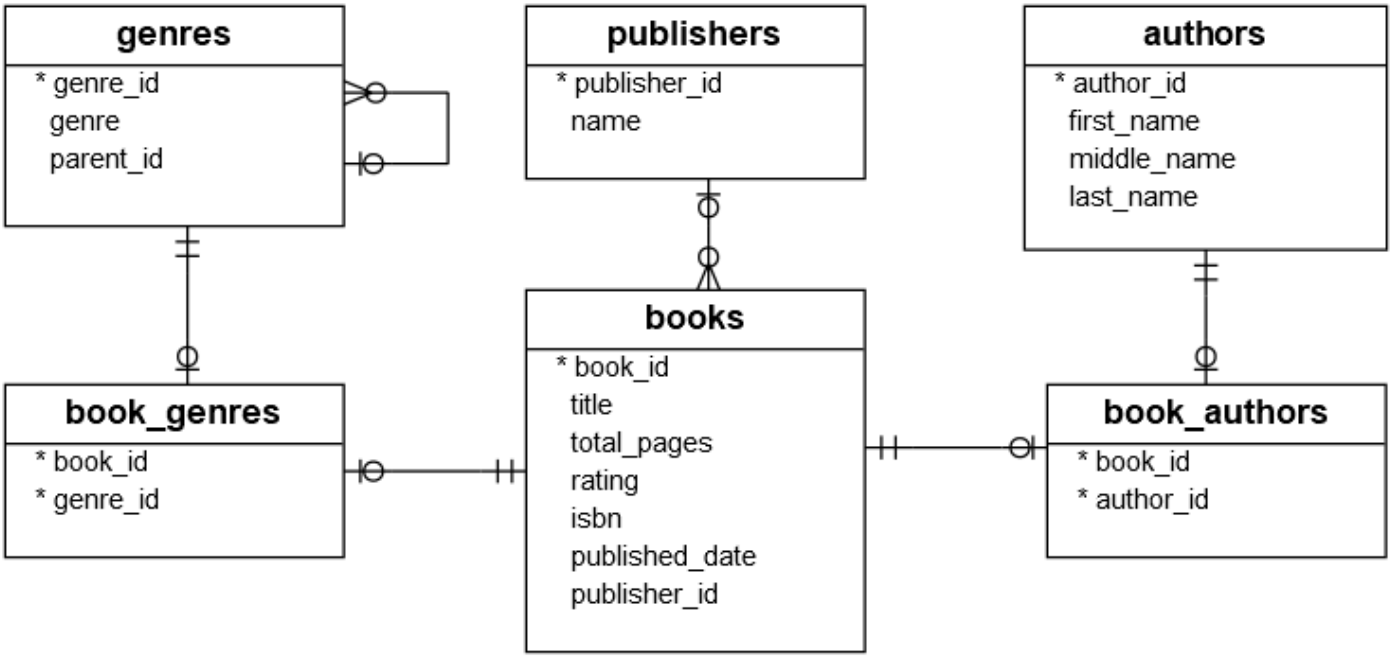
Class Methods:	**Class Collaborations (other classes):**

• `__init__(self,catalog)`:
  ○ initializes app window and sets up Tkinter widgets and layout, connecting to the `LibraryCatalog`
• `display_books(self, books)`:
  ○ takes a list of `Book` objects and displays their details the the `display_frame`
• `search_books(self)`:
  ○ gets user's search input and filter selection, performs a search using `LibraryCatalog`, and updates the display with results
• `add_book_gui(self)`:
  ○ collects input from user for a new book, creates a `Book` instance, and adds it to the `LibraryCatalog`
• `recommend_book_gui(self)`:
  ○ displays a book rec based on genre or author preferences using `LibraryCatalog.recommend_book()`
• `save_and_exit(self)`:
  ○ saves the catalog data to a file before closing the GUI
	• `display_books(self, books)`: collaborates with the `LibraryCatalog` class by accepting a list of `Book` objects from it
	• `search_books(self)`: this method calls `LibraryCatalog.search_by_author()` and `LibraryCatalog.search_by_genre()` to fetch lists of `Book` objects; collaboration with `LibraryCatalog`
	• `add_book_gui(self)`: collects user input to create a `Book` instance, then calls `LibraryCatalog.add_book()` to add it to the catalog; collaboration with `LibraryCatalog` and `Book`.
	• `recommend_book_gui(self)`: calls `LibraryCatalog.recommend_book()`
	• `save_and_exit(self)`: calls `LibraryCatalog.save_catalog()`, collaborating with `LibraryCatalog`

```
Subtask I.
   Subtask I.A: import data into system from a pre-existing file
       Subtask I.A.1: read file contents
       Subtask I.A.2: parse file data into structured format
       Subtask I.A.3: load data into catalog database
   Subtask I.B: store book details (title/author/genre/availibiltiy/date)
       Subtask I.B.1: define data structure to represent book details
       Subtask I.B.2: validate inpute data so its accurate and complete
       Subtask I.B.3: validate book details in catalog
   Subtask I.C: update catalog when the books are returned, checked out, or added to library
       Subtask I.C.1: track changes in book availibility
       Subtask I.C.2: modify catalog entries based on user actions
       Subtask I.C.3: log updates for review purposes

Subtask II.
   Subtask II.A: create function to search books by title/author/genre
       Subtask II.A.1: accept user input for search criteria
       Subtask II.A.2: filter catalog based on input criteria
       Subtask II.A.3: return matching search result
   Subtask II.B: implement sorting method (title alphabetically or publication date)
       Subtask II.B.1: define sorting options
       Subtask II.B.2: apply sorting algorithm to search results
       Subtask II.B.3: display sorted results in chosen order
   Subtask II.C: handle search results that are invalid (from user input)
       Subtask II.C.1: detect invalid/empty input from user
       Subtask II.C.2: provide error message
       Subtask II.C.3: prompt user to re-enter valid input

Subtask III.
   Subtask III.A: design graphical interface for users
       Subtask III.A.1: create layout for main interface (search bar, key, catalog view)
       Subtask III.A.2: design buttons and input fields for interaction
       Subtask III.A.3: integrate graphical components
   Subtask III.B: provide instructions for the user to navigate the search and sorting
       Subtask III.B.1: display instructions that are clear
       Subtask III.B.2: include help section for guidance
       Subtask III.B.3: ensure instructions are accessible
   Subtask III.C: include error messages to improve usability
       Subtask III.C.1: define specific error message for common issue
       Subtask III.C.2: display error message in clear manner
       Subtask III.C.3: guide user to resolve error (re-entering input properly)
```



https://www.db2tutorial.com/getting-started/db2-sample-database/
https://www.youtube.com/watch?v=girsuXz0yA8&ab_channel=Kite

Milestone 2 - due Monday, November 20th, 2024

For Milestone 2, we will be looking at your commit history and issue queue in your repository. Make sure you do a push before the deadline. It is definitely better to commit sooner rather than later to avoid any last-minute complications.

We will be looking at your commit history for the following:

1. No changes (i.e., commits) means no credit for Milestone 2
2. Meaningless commits don't count; make good commit messages!
3. Meaningless code doesn't count; make real progress by Milestone 2!
4. I do not expect perfection! Show me you're making smart choices. Remember the suggested incremental development from Chapter 15.31

We expect to see all of your files set up, including the test suite. Ideally, we would like to see some initial tests in the test suite. We also expect to see many of your functions stubbed out (the function defined, with `pass` inside the function body). More importantly, a healthy issue queue should be seen; a glimpse at titles should tell the story of where your code is going.

❗ IMPORTANT: If you are working with a partner, I EXPECT commits from both members. By the end of the project, if more than 70% of your *meaningful* commits are from a single partner, you will be penalized! Use git to effectively collaborate!

A word of warning. In Milestone 1, you copied in code from previous assignments into your repository. **We highly recommend you do NOT modify those files as a starting point for your project.**

Instead, create a ***new*** file for your project, and as you need snippets of code, you can copy in functions and other code from those files. There are many reasons to do this, but the most obvious reason and the only one we will mention here: **Your old code is bad.** It's bad, because you were just learning that concept. It's not because you're bad. You made mistakes or used coding practices that were incorrect, or there's better ways that you know now.

So, use your old code as a reference, not a starting point.

Milestone 3 - due Wednesday, December 4th, 2024

At this point, you should have made significant progress on your project. We will be looking again at your commit history and issue queue in your repository. Make sure you do a push before the deadline, and update your issue queue with progress you've made (as well as new issues!). For those working with a partner, I should see both of you making progress towards the final goal!

Be sure you're taking advantage of git by this point in the project. For example, say you have something working well, but you got this new, ingenious idea. You don't want to derail all of the code you've got working already, but you'd like to spend a little time exploring this new idea, because, IT'S AWESOME! Create a new branch! If the new idea doesn't pan out… no harm. Switch back to your original branch and keep coding away. New idea is awesome and you want it in the main code? Sweet! Merge the branches together. Remember, git is your friend in helping you write code without losing previous versions, as long as you remember to write meaningful commit messages, commit often, and take advantage of branching.

We expect to see most of your project completed by now. No functions should have stubs in them anymore. Those that are incomplete should contain FIXME or TODO notes that indicate what needs to be done next to finish that method. We also expect to see some calls in `main` that initialize your objects and tests that are passing. Your `main()` may not solve the final problem of your program, but it should demonstrate progress towards a finished product.

Your test suite should have lots of passing tests.

Milestone 4 - due December 13th at 5:00PM

This milestone is your final product. You should have all of the coding complete and cleaned up. You should also have your **README.md** complete with all ❗ removed, and all of your code should be cleaned up so that it is well-commented, docstrings are appropriate, there is not any unused code still in your program, etc. Your issue queue should NOT be empty; instead, it should acknowledge work that was incomplete, bugs that are known but not fixed, and features that you wanted to implement, but did not have time to incorporate. No code is ever truly complete!

In the Milestone 4 section of your README, there is a reflection prompt. Be sure to fill it in after reflecting on this final CSC226 experience as part of your submission. And finally, you'll be presenting your final project in a demo-style session:

YOU MUST ATTEND THE ENTIRE CLASS ON THIS DAY!!

Section A (MWF 9:20AM): Friday, December 13th, 8:30AM - 10:45AM
Section B (MWF 10:40AM): Thursday, December 12th, 11:45AM - 2:00PM

We will discuss the details of the demo in class.

Grading

Milestone 1:	10 points
Milestone 2:	10 points
Milestone 3:	10 points
Demo:	30 points
Code & Issue Queue:	25 points
README:	15 points
TOTAL:	**100 points**

Submission Instructions

Follow the submission instructions to submit your work by December 13th at 5:00PM.