COMP 3421

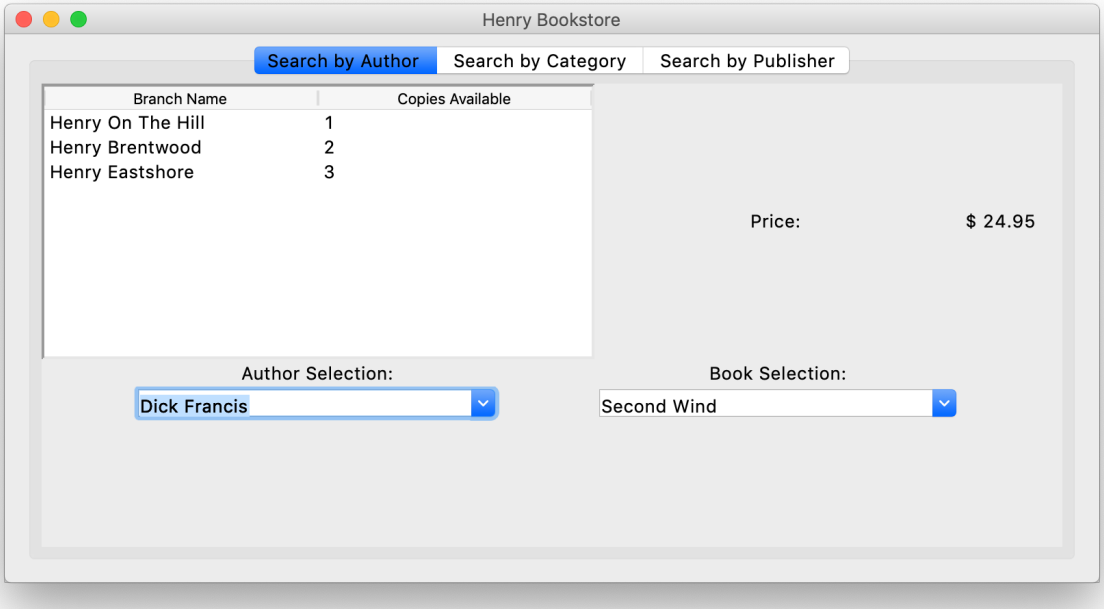
Assignment 1 – Python Version

20 points

This assignment will focus on connecting Python to a database via the Python Connector interface. There are two main pieces to the assignment: The Graphical User Interface (GUI) and the Data Access Object (DAO). In addition, there are some interface classes to glue the two main pieces together.

The Python GUI

You will build a Python GUI (tkinter) that looks more or less like the following. It is a front-end search to a bookstore database. It should provide searching by Author, Category, and Publisher. Note that I want you to do each of these in their own tab. You should have a file called Henry.py that starts the GUI running. Basically, this will set up the tabs using tkinter's Notebook. The content on each tab should be handled in separate classes (see below).



Each tab will have its own class. Call these classes: HenrySBA, HenrySBC, and HenrySBP (SBA – Search by Author, etc). Each of these classes should take into its constructor the Frame attached to the tab control. They will then each place content onto this Frame. This is best to do in the constructor.

I'll describe the SearchByAuthor here, but the other two tabs are similar. You will need a Combobox to select the author. Note that when you create such a box, you can provide a list of strings you want displayed in the box. Since you want this filled from the start with all the authors, you will get this list of author data from the DAO (see below). If you want to work on the GUI first, you can always make a list of fake info and see what it looks like visually before attaching to the DAO. Note that the DAO will return a list of Author objects. If you setup your Author object's __str__ method correctly, you can simply put the list of Author objects into the values of the Combobox, and it will automatically call the string producing method to convert them to strings.

You will also need to setup an event handler for the box – specifically by binding a function. I would suggest this function be in your class as well. That way, it can have access to any instance (self.) variables it needs, like the book combobox. This callback function will get executed when the user selects a new author from the box. Thus, it is its job to find which author was selected and then go to the DAO and get the book information associated with that author. Note that only Strings are stored in the Combobox so you cannot get the full Author object from the selection – only the author's string name. This poses a problem for us, since the author name is not enough to query with since it is potentially not unique (two authors may have the same name, but they will have different numbers). Thus, we need to obtain the author number in order to do a query – either for books or for author number due to potential duplicate names. The best way to handle this issue is to keep the Author list that was originally used to populate the Combobox as an instance variable. Then you can simply find the correct Author object from this list by list index (you can obtain which index was selected in the Combobox). Your book Combobox should already be visible on the screen, so you will need to remove all the old books from this box and add all the new ones to it.

The book Combobox will act in a similar manner to the author one. That is, it will need a bound callback function as well that goes and gets all the branch data from the DAO associated with the selected book. It should also know about the price of the book. The price information should be placed in a Label and the branch data in a TreeView. As seen in the GUI, I would like the branch info to contain the name of the branch location as well as how many copies of the book that branch has on hand.

Note that your GUI should always have information in all its boxes/lists. That means, when it first starts, it needs to populate the books from the first author on the list and the branch info from the first book on that list. I wouldn't try to do this right away when building your GUI, but it is something that needs to be done before it is submitted. Note that there might be an easy way to code this by breaking up your callback functions. This also means that you should never be able to select an author that doesn't currently have any books in the bookstore. There are two such authors in the database. Those authors should not even show up as potential items to select.

Please also note that the layout of the GUI shown above is the minimal needed for the assignment. Feel free to make your GUI look better than the prototype shown. Although the visual design layout of the GUI is the least important part in the grading – I'm much more

interested in it functioning than looking good. If you would like to change the specific GUI components described above, please come talk to me first.

The DAO

A Data Access Object is a class that sits between the GUI and the database. Its job is to separate the two, allowing the GUI to be changed out without having to rip out a lot of database code – or to change out the database without having to redo the GUI. You should create a class called **HenryDAO** which will serve as the DAO. HenryDAO should create the connection when it is constructed. That is, you will be making a single DAO object for your entire GUI. The connection will remain open the entire time the GUI is active. Then your DAO should provide a number of methods that will execute SQL queries on the database and return list of data which the GUI can use. Note that these methods will not need to reopen the connection since it should already be open.

Note that it is best to create the class in its own file called henryDAO.py. Then you can simply import it when needed via "from henryDAO import HenryDAO".

You will be in charge of deciding how many methods you will need to select the various types of data. For example, you will probably want one called getAuthorData. This would execute a query on the database which brings back the author information. This information probably includes author number as well as first and last names. Thus, you won't be able to simply return a list of a built-in type like String or Integer, since you have at least 3 pieces of information to return. To properly return this type of information, you should create another class called **Author**. Author's job is to simply hold onto the 3 pieces of information. The Author class might need __str__ or getters to provide the pieces of information to the GUI. That is, you will be returning a list of Authors and all the relevant author information will be inside that class. You are not to let the database connector cursor out of the DAO – just like you are not to pass SQL queries into the DAO. The job of the DAO is to separate database information from the GUI. The GUI should not even be aware that a database is being used – just that it asks the DAO for information and receives it in list form.

You will be writing several such methods like getAuthorData to get each set of data needed for the GUI. And along with this, you may need to create other classes like Author in order to hold onto the pieces of information returned from the database. Note that it is best to create all these "interface" classes in their own file called henryInterfaceClasses.py. Then you can simply import it when needed via "from henryInterfaceClasses import Author", etc.

Database Connector

I am providing sql scripts to load the henry_books database into MySQL. You will also need to install the python connector to connect to MySQL from Python. You can use PIP to do this via "pip install mysql-connector-python".

Testing

It would be a good idea to test part of the GUI with fake, hard-coded data. That way you can test out some GUI development even if you don't have the full DAO working. It would also be a good idea to test the DAO by having it do lots of printouts to the screen. That way you can test that it is returning the correct information without worrying about your GUI working.

For testing your DAO queries, it is quite useful to use first try the queries in MySQL workbench. It is also useful to printout the query you are about to execute. You will be often putting queries together from some fixed pieces as well as some variable pieces (i.e. author_num selected). And so, it can be useful to grab the constructed query that is printed to the screen and to run it directly in MySQL Workbench to see exactly what it should be returning to you.

Additional Notes

The most important thing in terms of grading is you DAO and connection to the database. The second most important piece is the overall GUI setup. The third is the handing of special cases to make sure the GUI doesn't do funny things in certain situations.

DAO: 8 points
Interface classes (e.g. Author)): 3 points
GUI display of data: 7 points
Initial population of GUI (and other special cases): 2 points

You may work with a partner on this assignment.