Jenna MacCarley

Recitation F

Code Layout:

My app will utilize the external module kivy, which is largely centered around the use of “widgets” or objects that are assigned specific tasks. As such, I will have many classes, and also a “builder string” that instantiates many unnamed objects in my code. Many of these widgets will inherit from other widgets already specified in kivy API, and will be customized to meet specific tasks in my code. As far as input and data storage goes, current, short-term program data will be stored in a handful of global objects. This is done specifically because of the way the kivy widget event handlers work, and I have found no reasonable alternative to globals as a solution. Basically, these class objects are instantiated once, often unnamed (due to the builder string) and must communicate with one another while the program is running. The only way I have found to do that is using the widget.dispatch() method which calls an event handler method directly in another widget. For long term data storage, I have a class, currentData, that will store all the data storage globals into a single object which is used for pickling the data and unpickling the data upon startup. This allows the program to retain all the saved settings.

Some notable/central classes include MyBudgetGraph, MyStatusLabel, MySubmitButton, RevenuesAndExpedentures, GetAllPercentages and LineGraph. RevenuesAndExpedentures (I know, spelled wrong but it stuck) is the central object type that my program depends on, as each revenue or expense input and it information is stored in these objects, then placed in a global list for access by the rest of the program. MySubmitButton is responsible as a hub for dispatching changes to every other part of the program, as every part depends on the revenue and expense data.

SOMETHING IMPORTANT to note about my code, and the thing I am arguably most proud of, are the many little kivy workarounds and object oriented uses imbedded into my code. Some examples include the use of the partial function in multiple areas, to bind events to functions and useful data. Also, kivy has issues deleting items from a widget (e.g. popups and canvas objects) completely once it uses them, so I had to do some experimenting as to how to best avoid these issues. One issue that I found unavoidable was the accordion item canvas problem, which will be evident once a line graph is drawn, where “chunks” of the graph get left behind.

At the end of all these classes there will be an app class that instantiates the app, loads the builder string, and begins the program. This will set up the screen manager which controls the changing between different views in the app. It will also instantiate the main globals which the program will use. Finally, at the end of my program I must register my classes with the factory.register() function so that kivy recognizes the use of all my classes.