FlashGen: The Flashcard Generator

Brendan Cicchi

Remington Maxwell

Sara Woods

# introduction

# comparative programming aspect

# parser

# gui design

When we first started looking into Python GUIs, there seemed to be a wide variety of options to choose from. After considering the importance of the looks and asthetics of our project, we came down to two options: WxPython and Tkinter. Originally we thought that Tkinter was the better option because there was a lot of documentation and example found online , but upon working with it further, the looks of the windows and text created by Tkinter looked outdated. We wanted the appearance of our flash card generator to look fresh and new, not like the current flash card creators currently present on the web. This caused us to turn to WxPython, which uses the current window settings of the computer’s operating system to display dialog boxes and window. The difference between these two GUI frameworks was essentially the difference between Windows 95 and Windows 8. After starting to work with WxPython, our project was well underway.

When first designing what the program would look like, we thought about what widely-used programs were similar to ours. The closest that came to mind was Microsoft PowerPoint, so we incorporated aspects of this program into our design process. We liked the idea of making the program appear more like an app, with a cleaner appearance and less clutter on the screen. This would make the program seem more user friendly. On the left hand side, we also wanted to show “previews” of what flashcards were currently in the deck. We thought that this would be a great feature to add because if someone was looking for a particular card, they could easily find it whereas in other flash card creators, one must click “next” multiple times to find the card. With printed out flashcards, it can be very tedious looking for one specific flashcard. With this planned setup of having previewed flashcards on the left and the current flashcard largely displayed on the right, we determined it would be an intuitive and familiar user interface. The blue color scheme was inspired by Microsoft Word and a consciousness to color blind users. By using darker and lighter shades of blue to clearly distinguish between active cards, a color blind user can also take advantage of this feature.

For the flashcard, functions were added to display the text from the JSON object. These functions included a “previous” function for pressing the previous button, a “next” function for pressing the next button, a “shuffle” function for pressing the shuffle button, and a “flip” function for pressing the flip button. For the previous and next functions, the index into the JSON object was shifted left or right, respectively. For the shuffle function, the Python shuffle function was used to shuffle the items in the JSON object. For the flip function, the index was changed to access the “answer” string of the JSON object.

The variety of buttons present in WxPython, especially buttons with images, was an important aspect when we decided to move from Tkinter to WxPython. Not only did WxPython have more button types, but it also had the added ability of button tips, displaying information on what a button does when the user hovers over it. This was essential to our “clean” design by allowing us to not need to add labels to the buttons on the design. For the button images, images were created in PhotoShop Elements and then colored to match the theme of the program design.

One of the most difficult parts of the GUI was figuring out how to implement the preview feature. We originally thought that displaying a text box over a static image, the same image as the main flashcard, would suffice. After working with this implementation, however, we found that there was not any easy way to trigger an event after clicking on the previewed flashcard. We wanted to be able to click a flashcard preview and to have the main flashcard displayed update to show the new text. This led us to thinking about using buttons for the previews. Although WxPython has many buttons to choose from, it does not have buttons with both an image and a label. This was very disappointing because we had wanted to match the flashcard previews’ backgrounds with the main flashcard background. We tried to manipulate text to appear over a button with an image, but WxPython has problems with layering objects on the GUI design. Thus, even though we were able to get text to appear over the button, the button was not clickable. This was extremely frustrating so we looked into other possibilities and found the Aqua Button, which was a button import. We decided on this particular button because the gradient it provided on the button was customizable and flashy.

After implementing the previews using buttons, we added a for loop that would iterate through the JSON strings, trim the text of the question so that it would fit properly on the button, create a button for each element and add the trimmed text, and then added all of these buttons to the left side. This left side consisted of a WxPython Panel, which had the added ability of adding a scrollbar if there were more flashcards than could be displayed in the provided space.

Another major challenge in the GUI design was figuring out how to properly implement the text to display on the main flashcard. Since the size of the text of the questions and answers could greatly vary, we created a cutoff point for the amount of text that could be added to the notecard. Despite adding this cutoff point, we still needed to figure out how to center the displayed text on the notecard with a variety of sizes. This led use to WxPython boxsizers, which allow centering of items by putting them into boxsizers. Boxsizers can be horizontal or vertical, can be nested, and allow organization of objects on a page. At first these seemed to be very useful and simple to understand, but they were very difficult to work with. Because WxPython doesn’t like layering objects, it doesn’t like to layer anything with boxsizers either. This meant that we couldn’t add our notecard image and notecard text to the same boxsizer otherwise the notecard text would not appear over the notecard image. After trying different combinations of vertical and horizontal boxsizers to center the text, we eventually settle on the text being added to its own boxsizer, which was a vertical boxsizer nested in a horizontal boxsizer. This boxsizer dynamically adjusted the location of the text based on the amount of text, centering it over the notecard image.

For reading in the text file, upon running our Python program, a dialog box is opened where the user can select a .txt file to read in. We decided to just have the type of file that FlashGen recognizes by a .txt because it is recognizable by any system, allowing users to easily share flashcard files with one another. Also, the flashcard files can be easily edited in any text editor. Once the file has been selected, Haskell handles the parsing and creates a JSON object, which is then used by the Python GUI to create the flashcards.

Once we finished all of the aforementioned aspects of our design, we decided to add a feature for organizing the cards in the stack. We added the up and down buttons to move the currently view flashcard up or down in the stack. In this way, the order of the flashcards could be determined. We also added a File Menu with the options Open, Help, About, and Quit. The Open option opens a dialog box where a different text file can be selected, changing the flashcard display to a different set of flashcards. The Help option displays information on how the buttons work in the program and how to properly write a text file for input into the program. The About option displays information about our team. The Quit option exits the program.