



Assignment Cover Letter
(Individual Work)

Student Information:

Surname: Staniswinata **Given Name:** Stephanie **Student ID Number:** 2501997836

Course Code : COMP6047001 **Course Name :** Algorithm and Programming

Class : L1AC **Lecturer :** Jude Joseph Lamug Martinez, MCS

Type of Assignments: Term Final Project

Submission Pattern

Due Date : 17 January 2022 **Submission Date :** 17 January 2022

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: Stephanie Staniswinata

Project Specification

The objective of this project is to create a flashcard app that is easy to navigate and let the user upload their own file to create the cards. This was inspired from how hard it is to find an app that can automatically make a card from an uploaded file that doesn't use tab, comma, slash, or other common symbols that might be used in a writing.

I decided to create an app that can create decks of flashcards manually and also can ask users to upload a txt file that separates the front and back side of the card with back-slash, something that I supposed (at least in my experience) not many people use in a writing system.

The inputs needed for this program are:

- Mouse/Trackpad: To navigate through app and activate widgets.
- Keyboard: To input letters.
- File: To be read by the program and the program can process it to a deck.

The outputs from this program are:

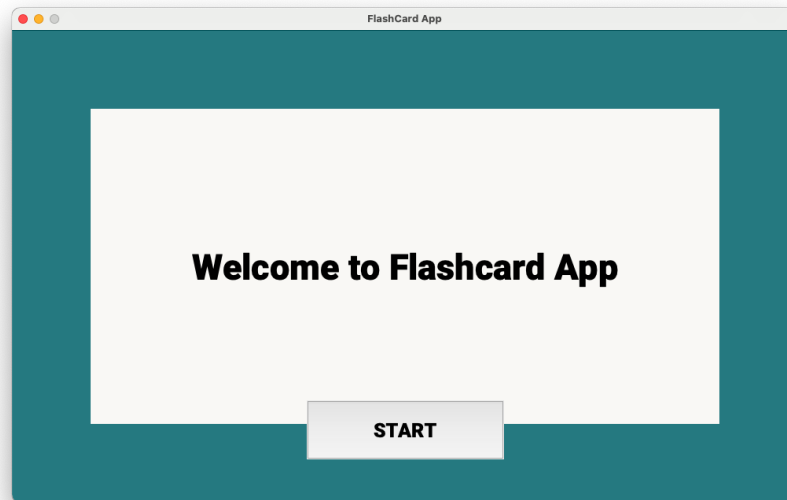
- Visual window with different context (navigate with widgets).

The modules used in this program are

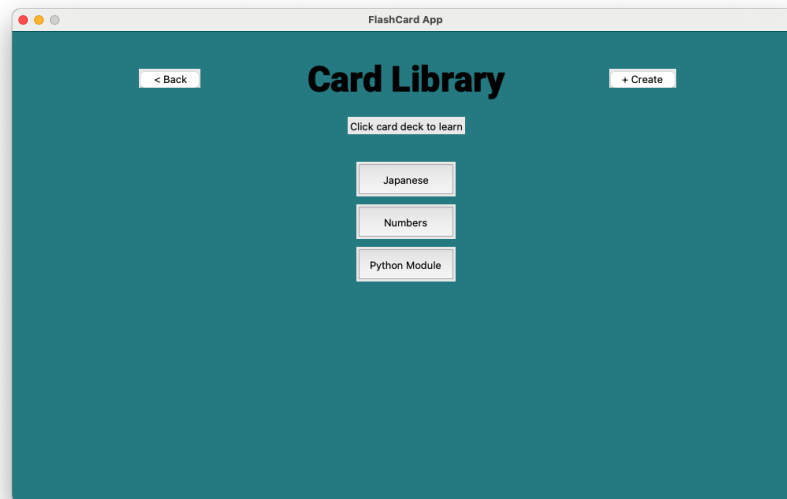
- Tkinter : for the GUI
- os : for getting the file path
- ast : for converting a string list to list

Solution Design

1. Users will see the welcome page



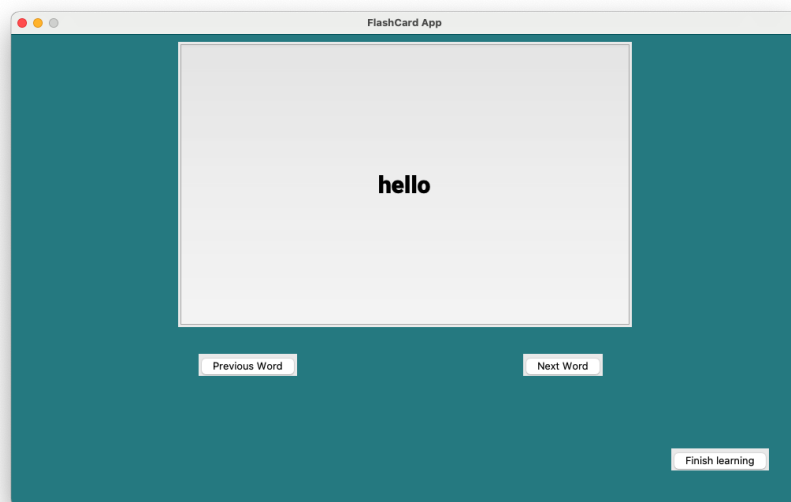
2. After clicking start, the user will go to the library page. Library list all the decks ever created by the user. There are also buttons such as back, create, and the buttons of the decks.



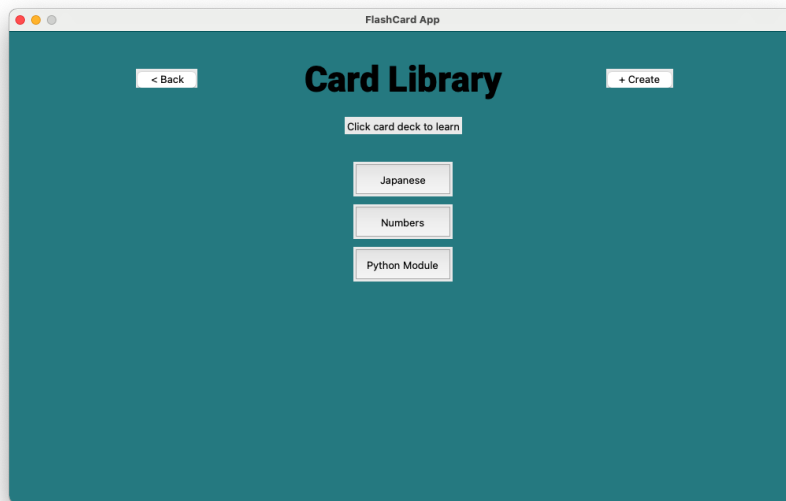
3. When clicking the deck button, the user will go to the learn page. Will see the front side of the first card.



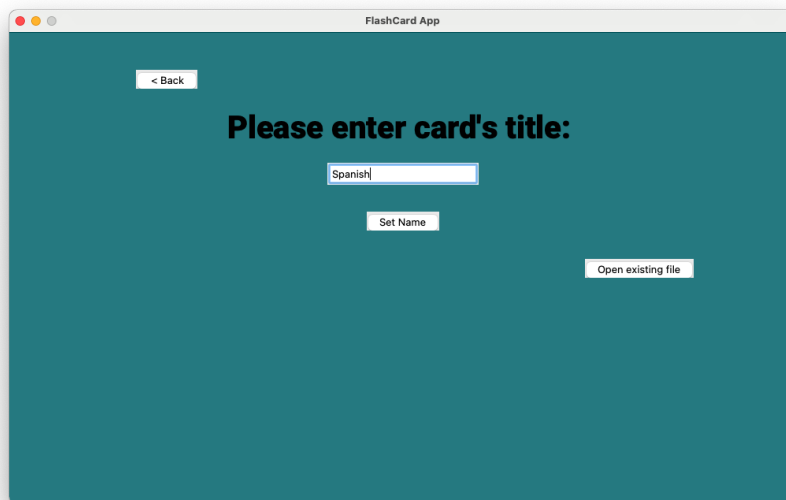
4. When the user clicks the big button, it will “flip the card”. Show the back side of the card.



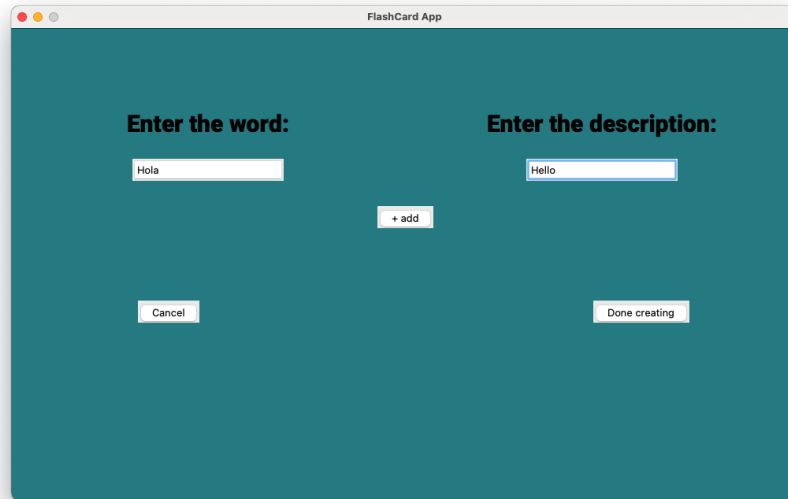
5. When the user clicks the 'Finish learning' button, it will return to the library page.



6. Users can click the '+ Create' button. Will go to the deck naming page, with two choices. To make a manual card, or read from a file. To make a manual deck, users can just fill the input bar with the name of the deck they wish to create.

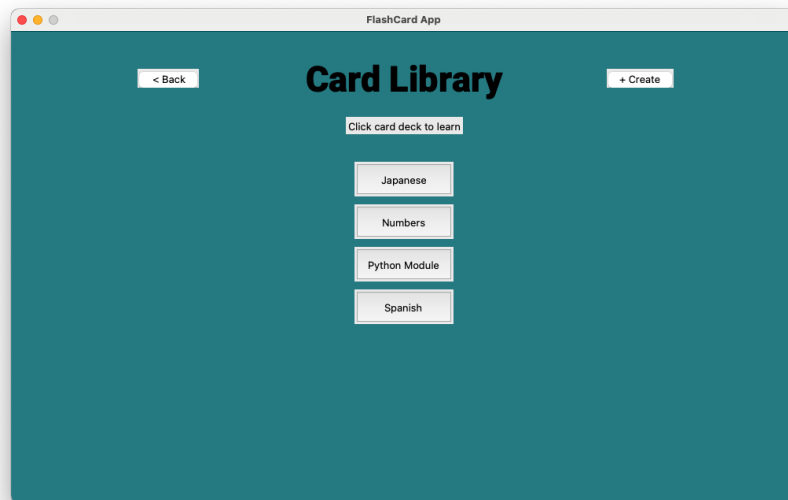


7. User then clicks the 'Set Name' button, and will be directed to create a cards page. Where users can manually input the front and back side of the card.



The screenshot shows a window titled "FlashCard App" with a teal background. It features two input fields: "Enter the word:" with the text "Hola" and "Enter the description:" with the text "Hello". Between these fields is a "+ add" button. At the bottom, there are two buttons: "Cancel" on the left and "Done creating" on the right.

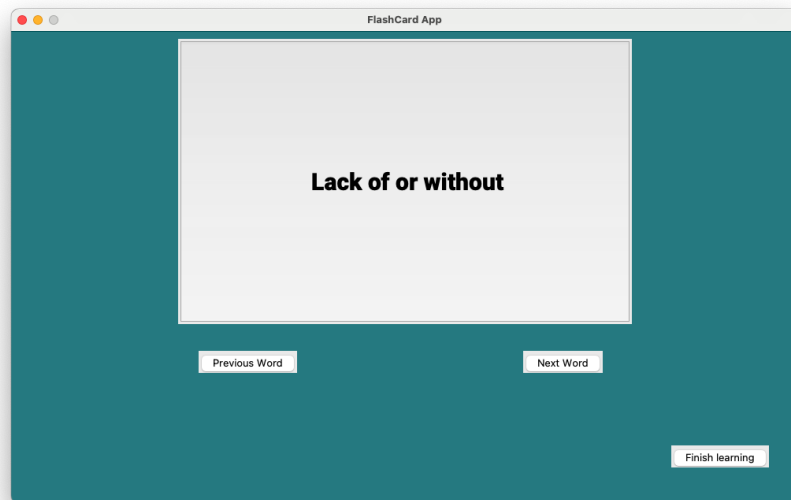
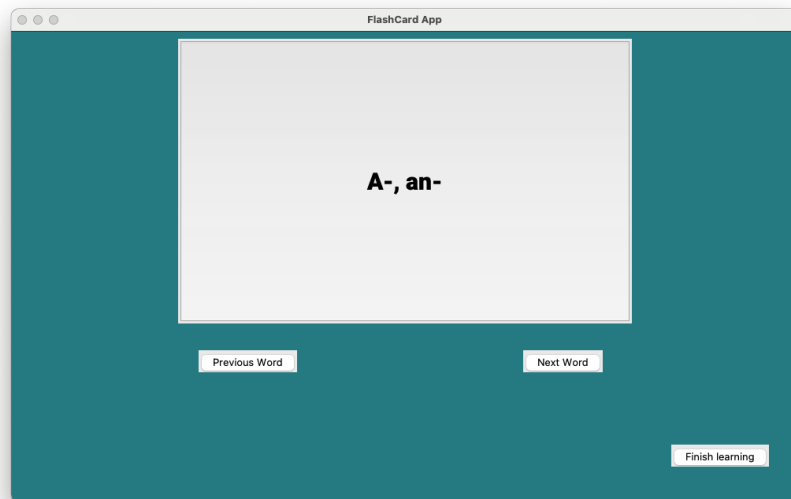
8. Users can keep adding words with the '+ add' button. When done, the user can use the 'Done creating' button. Users will be directed back to the card library page with a new button of the card they just made.



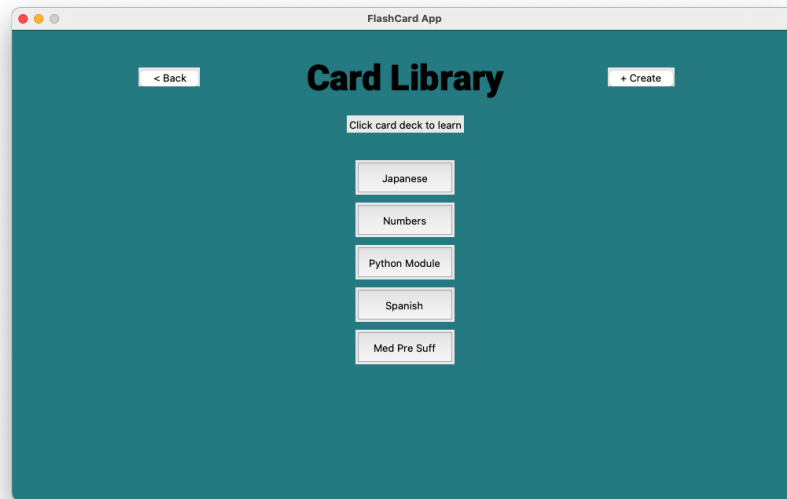
The screenshot shows a window titled "FlashCard App" with a teal background. At the top, there are two buttons: "< Back" on the left and "+ Create" on the right. The main heading is "Card Library". Below it is a button labeled "Click card deck to learn". Underneath this are four buttons stacked vertically: "Japanese", "Numbers", "Python Module", and "Spanish".

9. If the user clicks 'Open existing file', it will pop up a new window to choose a file that will be read by the program. After choosing the file, it will go to the learn page.





10. By clicking the 'Finish learning' button, the user will be directed back to the library page and user can see a new button from the file that they chose.



11. Users can exit the program by clicking the x button on the window, no saving required since everything is already stored inside the files.

Discussion

main.py

First, the initialization in the main.py file is where the root frame was created. This was made to determine the settings such as color, size and position of the window when running the app. The user will run the program from here.

```
global root

def main():
    # Window size variables
    width = 1000
    height = 600

    # Create tkinter root
    global root
    root = Tk()
    root.config(bg="#35838B")
    root.title("FlashCard App")

    # Assign value of device screen size
    setW = root.winfo_screenwidth()
    setH = root.winfo_screenheight()

    # Set the padding so it will position window center
    padW = (setW//2)-(width//2)
    padH = (setH//2)-(height//2)

    # Set the window size and position
    root.geometry(f"{width}x{height}+{padW}+{padH}")
    root.resizable(False, False)

    # To fill in the window with widgets from frames.py
    frames.welcome(root)

    # To run the window
    root.mainloop()
```

This is from the main.py file, where it's the initialization of the window using tkinter. I created a width and height variable so it's easier to input for sizing purposes. I also make the initial window's position appear in the middle of the screen by using the w.info_screenwidth and w.info_screenheight and calculate the middle part and assign it to the geometry method from tkinter, to set the size of the window. I also set the resizable to false so the user can't change the resolution, since the app is not responsive.

factory.py

Where the class is located, it will create a deck object with frame, name, and list as the parameters needed to create it. The function of this class is to make every deck as an object, therefore the user can access a specific list from a specific object name.

This class has the method buttonMake(), this is to create a specific button for that object. So it will return the right name and list to the learnCards frame.

The button created, when clicked, will take the user to the learning page. Where the user can start learning instantly.

```
8 class Deck():
9     # Initialization
10    def __init__(self, frame, name, list):
11        self.__name = name
12        self.__list = list
13        self.__frame = frame
14    # Getters
15    def getName(self):
16        return self.__name
17
18    def getList(self):
19        return self.__list
20    # Setters
21    def setName(self, name):
22        self.__name = name
23
24    def setList(self, list):
25        self.__list = list
26    # To create button
27    def buttonMake(self, frame, name, list):
28        # Create button name without \n
29        name = name.strip("\n")
30        # Create button with button widget tkinter
31        # Command to learnCards page when clicked
32        btn = Button(frame, text = name, command=lambda:frames.learnCards(frame, list), height=2, width=10).pack(side=TOP, pady=5)
```

frames.py

There are 5 frames I created with the tkinter module. The first frame will be the welcome frame.

```
12 def welcome(root):
13     # Welcome page with a start button
14     cleanPage(root)
15     box = Frame(root, bg = offWhite, width=800, height=400)
16     box.pack(padx=100, pady=100)
17
18     title = Label(root, text="Welcome to Flashcard App",font=("Roboto", 45), bg=offWhite)
19     title.place(relx=.5, rely=.5, anchor=CENTER)
20
21     startBtn = Button(root,text="START", padx=70, pady=20, font=("Roboto", 25), command=lambda:cardLib(root), border=0)
22     startBtn.place(relx=.5, rely=.845, anchor=CENTER)
```

Root is from the initial root form main.py, since it's global it will share the same rules such as position, window size, etc.

The first line of the method is the cleanPage method that will destroy all the widgets from the previous frame. Even though this is the first page of the app, still users can go back to this frame and the program has to destroy whatever widgets appeared before.

The welcome page has the frame, label, and button widget to start (a button to move to card library page).

Next, there's the card library frame.

```
24 def cardLib(root):
25     # Card library page where user can access decks and create
26     cleanPage(root)
27     box2 = Frame(root)
28     box2.pack(pady=80, side=TOP)
29
30     loadCards(box2, root)
31
32     title = Label(root, text = "Card Library", font=('Roboto', 45), bg= back)
33     title.place(relx=.5, rely=.1, anchor=CENTER)
34
35     hint = Label(root, text="Click card deck to learn")
36     hint.place(relx=.5, rely=0.2, anchor=CENTER)
37
38     createBtn = Button(root, text = "+ Create", command=lambda: createName(root), border=0)
39     createBtn.place(relx=.8, rely=.1, anchor=CENTER)
40
41     backBtn = Button(root, text = " < Back", command=lambda:welcome(root), border=0)
42     backBtn.place(relx=.2, rely=.1, anchor=CENTER)
```

It has multiple buttons that let the user navigate through the app.

Created another frame, as a place where the program will place new buttons created with the class. Also added a hint label to tell the user to click on a button to learn them. The *createBtn* will navigate to the createName frame. The *backBtn* will navigate back to the welcome page.

When the user clicks the createBtn, it will go to the createName frame.

```
44 def createName(root):
45     # Create deck page, user can make manually by set name or use an existing file from device
46     cleanPage(root)
47     namePrompt = Label(root, text="Please enter card's title: ", font=("Roboto", 40), bg=back)
48     namePrompt.place(relx=.5, rely=.2, anchor=CENTER)
49
50     global cardName
51     cardName = StringVar()
52     name = Entry(root, width=20, textvariable=cardName)
53     name.place(relx=.5, rely=.3, anchor=CENTER)
54
55     setBtn = Button(root, text = "Set Name", command=lambda: cardNaming(cardName, root), border=0)
56     setBtn.place(relx=.5, rely=.4, anchor=CENTER)
57
58     openBtn = Button(root, text = "Open existing file", command=lambda:createWithFile(root), border=0)
59     openBtn.place(relx=.8, rely=.5, anchor=CENTER)
60
61     backBtn = Button(root, text = " < Back", command=lambda:cardLib(root), border=0)
62     backBtn.place(relx=.2, rely=.1, anchor=CENTER)
```

This frame will allow the user to input a deck name they want to create and click set name, it will run a method called *cardNaming* where it will need the cardName variable and root as the frame. Users can enter any name they want for the deck.

The user also has a choice to open an existing file and create a deck from that file.

If the user chooses to make the cards manually, they will go to the createCards frame.

```
64 def createCards(root):
65     # Create cards page, user can input manually front and back / word and desc to cards
66     cleanPage(root)
67     global word
68     word = StringVar()
69     global desc
70     desc = StringVar()
71     global cardName
72
73     wordLbl = Label(root, text="Enter the word:", font=("Roboto", 30), bg=back)
74     wordLbl.place(relx=.25, rely=.2, anchor=CENTER)
75
76     wordInput = Entry(root, width=20, textvariable=word)
77     wordInput.place(relx=.25, rely=.3, anchor=CENTER)
78
79     descLbl = Label(root, text="Enter the description:", font=("Roboto", 30), bg=back)
80     descLbl.place(relx=.75, rely=.2, anchor=CENTER)
81
82     descInput = Entry(root, width=20, textvariable=desc)
83     descInput.place(relx=.75, rely=.3, anchor=CENTER)
84
85     addBtn = Button(root, text = "+ add", command=lambda: addCard(word, desc, wordInput, descInput))
86     addBtn.place(relx=.5, rely=.4, anchor=CENTER)
87
88     doneBtn = Button(root, text = "Done creating", command=lambda: finishAdd(root))
89     doneBtn.place(relx=.8, rely=.6, anchor=CENTER)
90
91     cancelBtn = Button(root, text = "Cancel", command=lambda: cancelCreate(root, cardName))
92     cancelBtn.place(relx=.2, rely=.6, anchor=CENTER)
```

This frame will have two input (entry in tkinter) fields, where the user will input the front and back side of the card. The user can use the *addBtn* to add both variables to a list from the function *addCard*.

User can use the *doneBtn* when they are done creating and adding the deck. It will run the *finishAdd* method.

The user can also cancel the making using the *cancelBtn*, it will run the *cancelCreate* method.

After creating the deck, the user will go to the card library and the user can click on any card to learn them and it will bring the user to the learnCards frame.

```

94 def learnCards(root,list):
95     # Learn cards page, where user can "flip" card by clicking them
96     cleanPage(root)
97
98     global i
99     i = 0
100
101     show = StringVar()
102     show.set(str(list[i][0]))
103
104 > def flip(show,list):-
114
115 > def nextWord(show,list):-
126
127 > def prevWord(show,list):-
138
139     cardDis = Button(root,textvariable=show, wraplength=500,font=("Roboto",30),command = lambda: flip(show,list), height=10, width=30)
140     cardDis.pack(pady=10)
141
142     nextBtn = Button(root, text = "Next Word", command=lambda: nextWord(show,list))
143     nextBtn.place(relx=.7, rely=.7, anchor=CENTER)
144
145     backBtn = Button(root, text = "Previous Word", command=lambda: prevWord(show,list))
146     backBtn.place(relx=.3, rely=.7, anchor=CENTER)
147
148     finishBtn = Button(root, text = "Finish learning", command=lambda:cardLib(root))
149     finishBtn.place(relx=.9, rely=.9, anchor=CENTER)

```

It has a global variable so it will let the button change names using the 3 methods inside the frame. They are placed inside the frame caused by a global variable that can be very hard to work with across multiple files.

cardDis will act as a card in this app. When clicked it will change the text using the *flip* method.

```

104 def flip(show,list):
105     """
106     Method to flip card
107     Manipulate button text
108     """
109     word = show.get()
110     if word == str(list[i][0]):
111         show.set(str(list[i][1]))
112     else:
113         show.set(str(list[i][0]))

```

- Flip method will change the button text.
- The show variable is a global variable in the frame. It acts as a string variable with a string from a list as it's value.
- The flip will get the show string and store it in the 'word' variable.
- When clicked, it will check if the word is the same as the front text, it will set show to the back text, and vice versa.

```

115 def nextWord(show,list):
116     """
117     Method to go to next word from list
118     """
119     global i
120     if i == (len(list)-1):
121         i = 0
122         show.set(str(list[i][0]))
123     else:
124         i += 1
125         show.set(str(list[i][0]))

```

- nextWord method will set the show variable to the next string of the list, also using the global i.
- If it reaches the last card, it will set it to 0 and will start from the beginning again.

```

127     def prevWord(show, list):
128         """
129         Method to go to previous word from list
130         """
131         global i
132         if i == 0:
133             i = len(list)-1
134             show.set(str(list[i][0]))
135         else:
136             i -= 1
137             show.set(str(list[i][0]))

```

- prevWord method have the functionality as nextWord method. But instead of +1, it will -1 the global variable i.
- If i reaches 0, it will set it to the len-1 or to the last card in the list.

methods.py

This file is filled with methods used for this app to work.

First, there's a global variable.

```

14     global cards
15     cards = []

```

This will be used later to append list of a list of fronts and backs of the cards.

```

18     def cardNaming(n, frame):
19         """
20         Function to create
21         deck name from user input
22         """
23         # Get the name from Entry input
24         name = n.get()
25
26         # Won't allow user to create a card with empty name
27         if name == "":
28             error = messagebox.showerror(title="Card Naming", message="Name can't be empty")
29         else:
30             # Write the name into a cardlist file to store it
31             with open("storage/cardlist.txt", "a") as f:
32                 f.write(name + "\n")
33             # Call to run the createCards frame
34             frames.createCards(frame)

```

The cardNaming method can be found in the createName frame.

After the user input the name, it will run this code and to process a tkinter string variable, we need to use the get method and assign it to a variable so later we can write it to a txt file called cardlist.txt, to store the deck name. I implemented file handling to add names to the file.

I also added an error handling, if the user set the name with an empty string, it will pop up a message box from tkinter with a show error box that will tell the user that the name can't be empty.

After setting the name and there are no errors, it will bring the user to the createCards frame.

Next method will be createWithFile that allows users to make a deck from a file from their device. It will ask the user to open a file using a askopenfilename method from filedialog from tkinter.

Then it will run the readFile method with the file path as one of the parameters.

```

37 def createWithFile(frame):
38     """
39     Method to create card deck
40     from user's existing file
41     """
42     # Get file from path
43     file = filedialog.askopenfilename(initialdir="/", title="Select existing file", filetypes=[("Text Files", "*.txt")])
44
45     if file == "":
46         # If the user cancel to add file, it will go back to the window and add nothing to the storage
47         frames.createName(frame)
48     else:
49         # Call to run readFile method
50         readFile(file, frame)

```

The `readFile` allows the add the file name using `splittext`, and `basename` method from `os` to get just the file name instead of the full path as the name.

Then it will convert the lines from the txt file into a temporary list variable. Using the `split` method it will split the one line of `front\back` into two elements of a list.

Then it will write the list to the `cardStorage.txt` file and run the `learnCards` frame with the list from the recent file as the words.

```

112 def readFile(file, frame):
113     """
114     Method to read user's file
115     """
116     # Add the file name to cardlist file as card's deck name
117     with open("storage/cardlist.txt", "a") as f:
118         f.write(os.path.splitext(os.path.basename(file))[0]+"\\n")
119
120     # Open the user file and change to list
121     with open(file, "r") as f:
122         global words
123         words = []
124         for i in f:
125             i = i.replace("\\n", "")
126             words.append(i)
127
128         for i in range(len(words)):
129             words[i] = words[i].split("\\\\")
130
131     # Add the list to cardStorage file
132     with open("storage/cardStorage.txt", "a") as f:
133         f.write(str(words)+"\\n")
134
135     # Will let the user to learn the cards once they add the file
136     frames.learnCards(frame, words)

```


If the user chose to set a name and create a deck manually. It will go to a createCards frame and run some methods inside such as addCard method.

```
49 def addCard(front, back, wordInput, descInput):
50     """
51     Method to add one card to the list
52     from user input and file
53     """
54     global cards
55
56     # Access global list and append value from user input to list
57     cards.append([front.get(), back.get()])
58
59     # Clean the entry box
60     deleteEntry(wordInput, descInput)
```

wordInput and descInput are parameters to get global variables from the entry widget from tkinter, it will be the parameter for deleteEntry.

This method will append the front and back side of the card to a temporary list called cards from earlier. Use the get function from tkinter to access the string from a tkinter string variable.

Then it will run the deleteEntry method.

```
63 def deleteEntry(word, desc):
64     """
65     Method to clear the entry user input
66     in card making page after user add a pair
67     """
68     word.delete(0,END)
69     desc.delete(0,END)
```

This will make the parameters word and desc that has the value of a tkinter entry widget to be deleted. So that every time the user adds a front and back, it will clear the entry field and the user can input other words without deleting the words manually from the field.

The finsihAdd method will signal the program that the user is done adding cards and it will write the temporary cards list to the cardStorage.txt file.

Then it will set the cards list variable to an empty variable, ready to use to make another new card.

It will then navigate the user to the cardLib frame.

```

72  def finishAdd(frame):
73      """
74      Method to signal program that the user
75      finsih adding cards to deck
76      Will store list to cardStorage
77      """
78      with open("storage/cardStorage.txt", "a") as f:
79          global cards
80          # Write list to cardStorage file, change type to string
81          f.write(str(cards)+"\n")
82
83      # Assign empty list to cards, ready to use to make another deck
84      cards = []
85
86      # Return user to card libarary page
87      frames.cardLib(frame)

```

The deleteName method will delete a deck name from the carflist.txt file. It will take the whole file and will read them and store them to a titles variable (as in the title of the deck). Then it will re-write the file with the same titles except the title that is also the name or in this case the one that wants to be deleted.

```

90  def deleteName(n):
91      """
92      Method to delete deck name from cardlist file
93      """
94      name = n.get()+"\n"
95      with open("storage/cardlist.txt","r") as f:
96          titles = f.readlines()
97      with open("storage/cardlist.txt","w") as f:
98          for title in titles:
99              if title != name:
100                 f.write(title)

```

This deleteName method can be found in the cancelCreate method where can be accessed when creating the cards manually. After deleting the name of the deck, it will empty the cards list and return to the cardLib frame, without adding anything to the storage or creating any buttons.

```

103  def cancelCreate(frame, name):
104      """
105      Method to delete a created deck of cards
106      """
107      deleteName(name)
108      cards = []
109      frames.cardLib(frame)

```

The next method will be the `loadCards` method. This will be run in the `cardLib` frame.

It has a list of buttons and by iterating through the `cardlist` and `cardstorage` file, it will read the files and create `Deck` objects using them.

I used the `ast` module here to access the `literal_eval` method to change a stringed list to a list.

Then I use the button variable to create a `Deck` object and use the corresponding name and words from the iteration so the name of the deck and list will match.

Then will append to the buttons list.

Using the buttons list, will iterate and use `btn` as a 'self' and use the class method `buttonMake` to create the button widgets.

```
139 def loadCards(frame, frame2):
140     """
141     Method to load the cards in card library page
142     """
143     # List to store objects
144     buttons = []
145     # Open file of card list and storage
146     with open("storage/cardlist.txt") as lst, open("storage/cardstorage.txt") as strg:
147         # Access to read the lines
148         name = lst.readlines()
149         words = strg.readlines()
150         # Iterate through the file per line
151         for i in range(len(name)):
152             # Use ast to make a string of list to a list
153             words[i] = literal_eval(words[i])
154             # Object creation
155             button = Deck(frame, name[i], words[i])
156             # Append object to buttons list
157             buttons.append(button)
158     # Iterate through the buttons object list to create button
159     for btn in buttons:
160         # Use method from class to create button
161         btn.buttonMake(frame2, btn.getName(), btn.getList())
```

Since this method's run on the `cardLib` frame, it will pack and show the buttons inside the frame.

Therefore the user can pick which card to learn.

The last method but the most important one will be the `cleanPage` method.

```
164 def cleanPage(root):
165     """
166     Method to clean the window
167     """
168     for widget in root.winfo_children(): # To know the widgets used in that page
169         widget.destroy() # To delete all the widgets with iteration
```

Where it checks every widgets on screen with `winfo_children` and iterates them to use `destroy` method on them, so they will be off the window.

Sample of **cardlist.txt** and **cardstorage.txt**

```
cardlist.txt × ...
storage > cardlist.txt
1 Japanese
2 Numbers
3 Python Module
4 Spanish
5 Med Pre Suff
6

cardStorage.txt ×
storage > cardStorage.txt
1 [['こんにちは', 'hello'], ['おなまえは', "what's your name"]]
2 [['1', 'One'], ['2', 'Two'], ['3', 'Three'], ['4', 'Four']]
3 [['Pandas', 'data analysis - to clean and analyze data']]
4 [['Hola', 'Hello']]
5 [['A-, an- ', 'Lack of or without'], ['-ation', 'Indication']]
6
```

This will allow users to save the cards even though they exit the program.

cardStorage has a list filled with lists that represent cards.

Read: [['front1', 'back1'], ['front2', 'back2']]

Demo video

<https://youtu.be/yPBkN3F1d2E>

References

<https://docs.python.org/3/library/tkinter.html>

https://www.tutorialspoint.com/python/python_gui_programming.htm

<https://www.youtube.com/watch?v=YXPyB4XeYLA>