T1A3 -Terminal App - Adam Hutt

Coder Academy 2023 -Term 1

Application Idea - Flashcard App

Introduction

The Idea for the applications is to be a customisable flashcard app for the user to study any topic they want. Flashcards are an ideal way to study or learn a new topic as they rely on repetition, force the user to actively recall something and due to the introduction of the internet and smartphones, they are now portable. Having used some of these apps to study Japanese language I wanted to try give it a try.

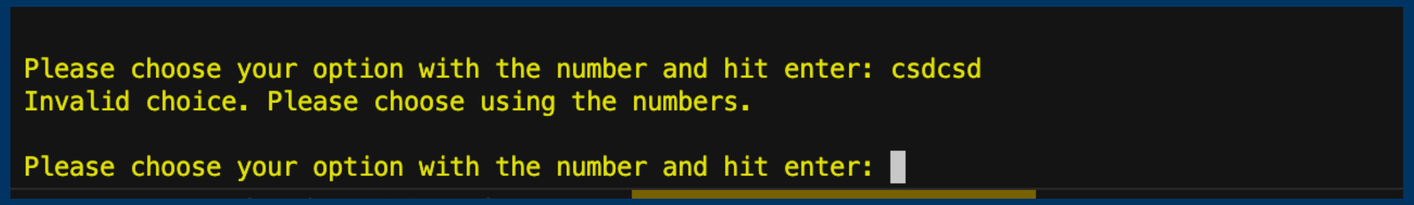
In terms of the coding for this project. I decided to implement using OOP. Although one of the keys features of OOP - creating instances was not frequently used , other features of OOP like inheritance were greatly useful to me.

Overview

Main Menu

The main menu functions as a point of navigation on the app. It clearly displays what the user can perform with the app, by choosing the corresponding number . This structure is how the user will navigate the app. It asks the user for input and the user selects the corresponding number as to what they want to . The headers and messaging also are different colours to provide distinction between what to interact with.

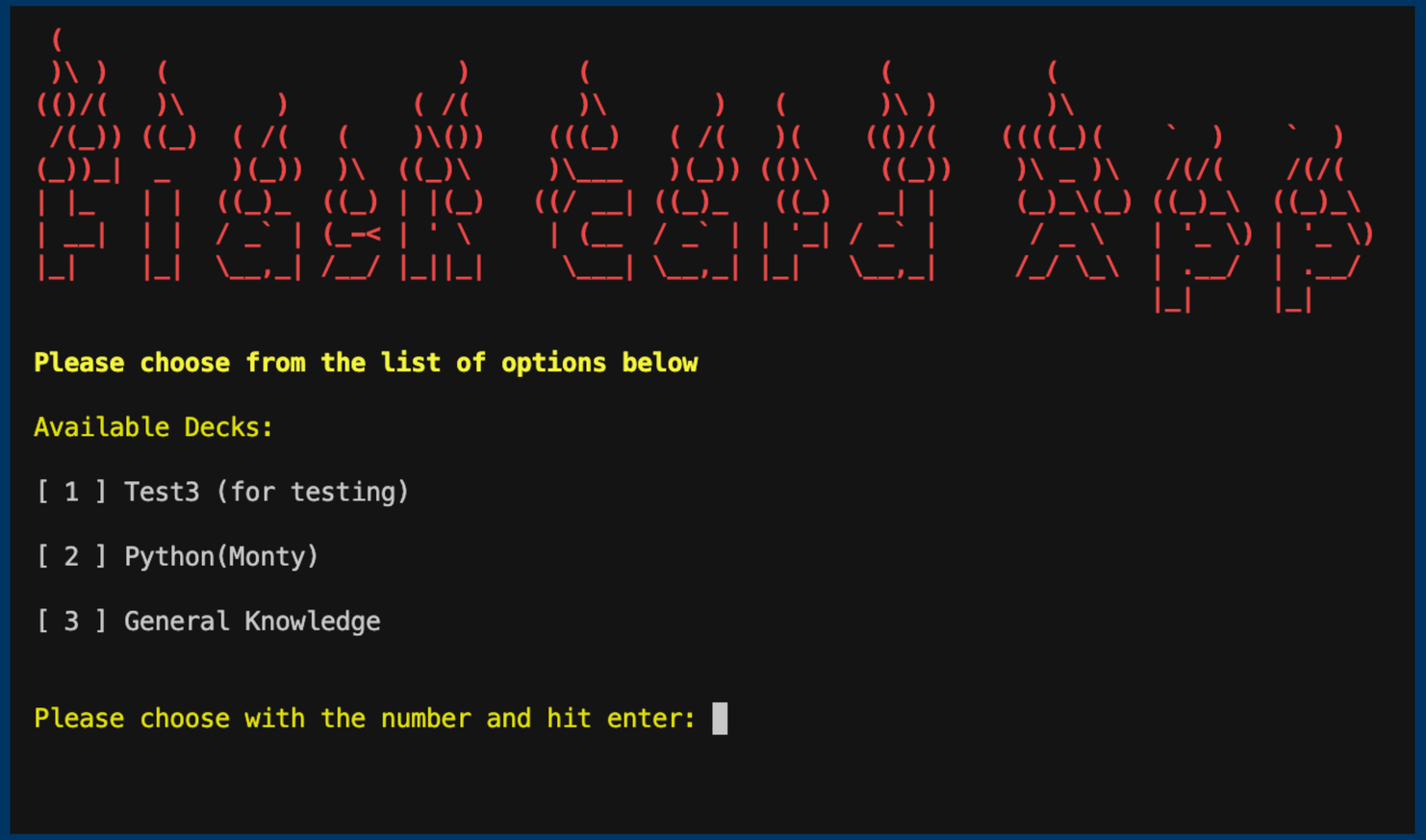
The page also has some ASCII art as a header to be more visible to the user. If the user accidentally enters a wrong number or letter, a message will display to the user that their choice was invalid and await a correct input to proceed. Below is the error message shown to the user.



Main Menu

For your reference, you can see the menu and the information displayed to the user. The menu displays options to utilise the Apps features. Within options 2 and 3 , there are multiple features the user can use to their needs and manage the topics they want to study . The first menu will take the user to use the apps primary function , which is to Study, or flip through a deck of cards to improve their memory on the topic.

Study Menu

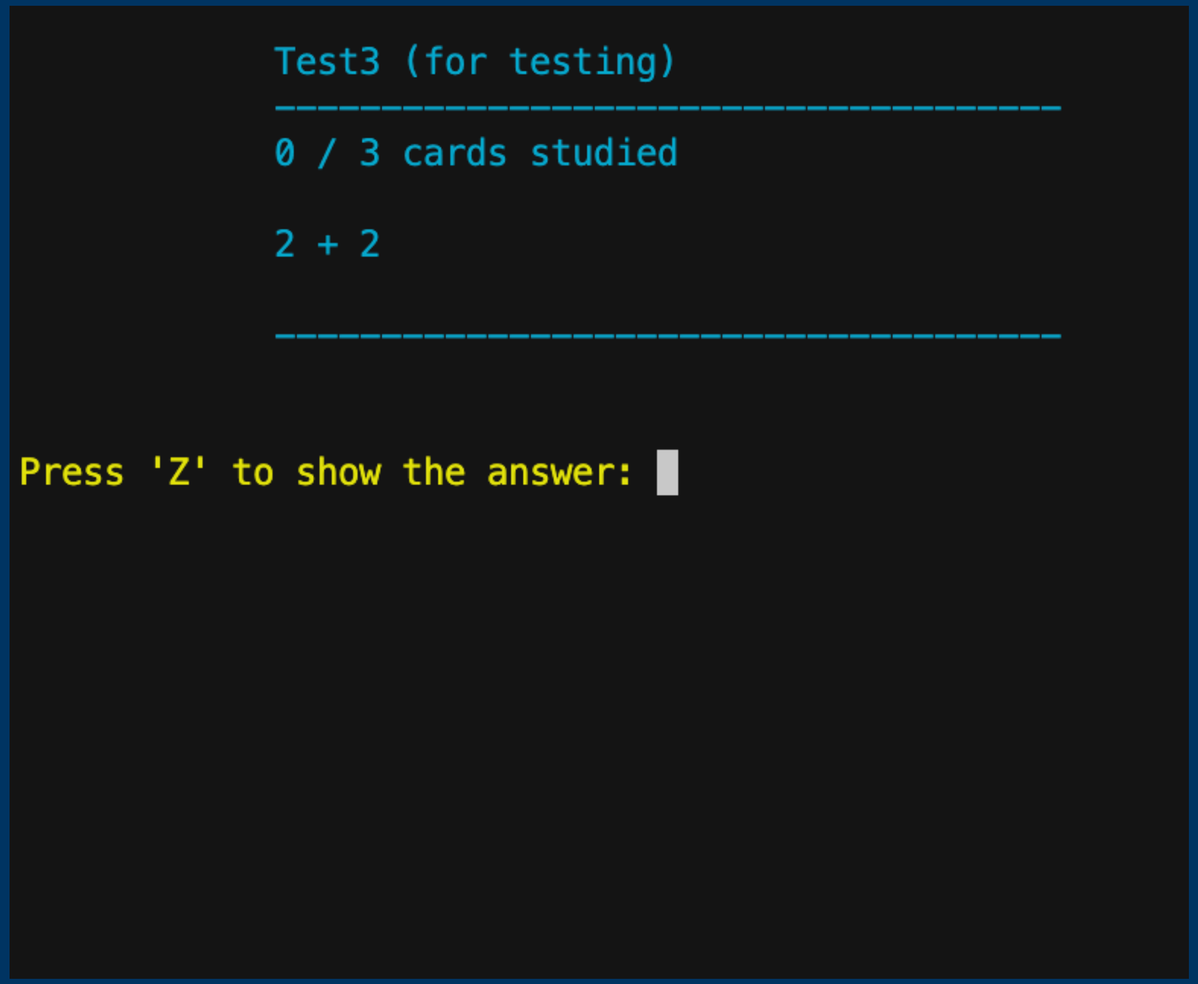


This is the study menu, where the user can access their decks. The display is similar to the main menu but in place of navigation options are the users available decks to study. The deck is then chosen using the number corresponding to the deck they want to study. Also similarly to the main menu, and throughout the function, the user can input anything and not crash the application and display guidance to them.

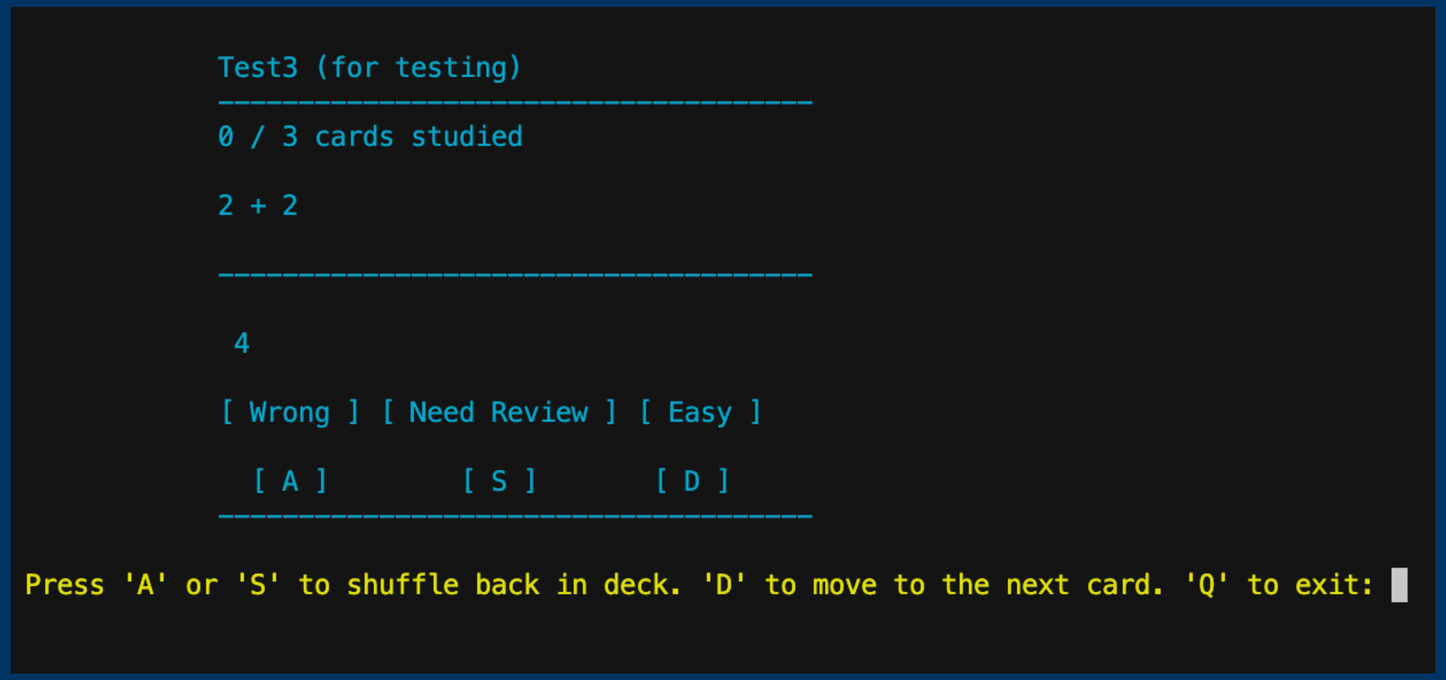
Study a Deck (Feature 1)

The study feature allows the user to study a deck of cards. Each card has a card with a phrase/Question on the front and the user as to remember the phrase or answer on the back of the card. As you can see here the user has selected the Test 3 deck . The user will see this screen , consider the answer and then have to type “z” and hit enter into the yellow input, to reveal the other side. This deck has 3 cards which will appear out of order randomly, until the user passes them all. As you can see, we have the deck name , progress through the deck (0/3) and the content (2+2) for the user to see.

Card 1 (2+2) - Example



Study a Deck (Feature 1)



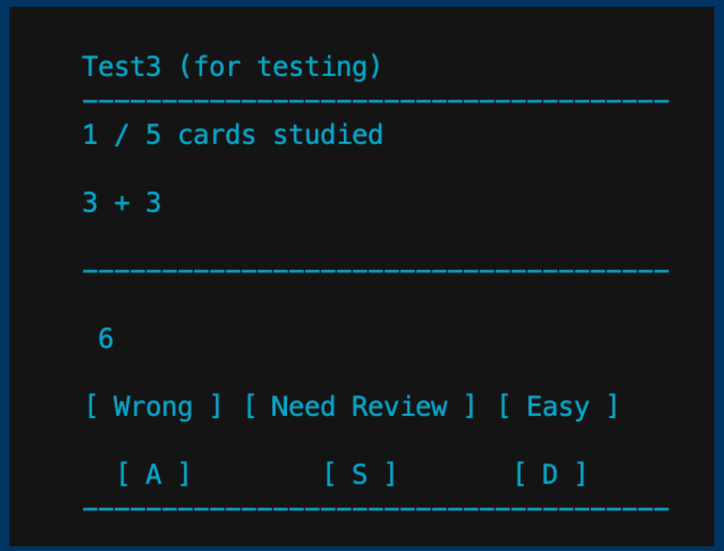
The user has revealed the answer. For optimised use the user needs to determine if the info was easy to remember (D), did it take a lot of thinking to remember and they need to review? (S), or completely wrong (A). The user can also quit the deck from here or if they want to quit (Q). The users input here , will effect the deck.If the user selects A …

Still Card 1 (2+2)

…The card will be be copied twice and two more copies of this card will be re-shuffled back into the deck. Increasing the deck size by two. To complete the deck, The user will have to remember the other side of the answer easily (D) two more times to complete the deck. After inputting their answer the next card in the deck , will be shown randomly and the tally will change to one card studied. So the more wrong answers the more cards the user will have to remember and more time spent on them and study required.

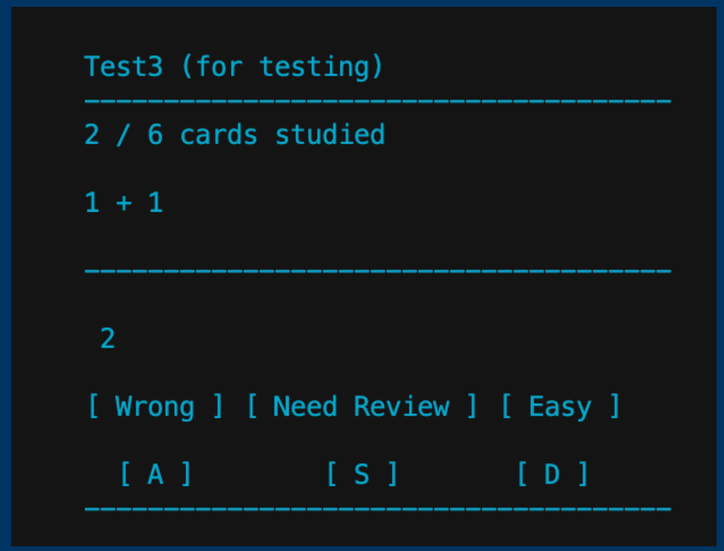
Study a Deck (Feature 1 - Continued from previous page

Card 2 (3+3)



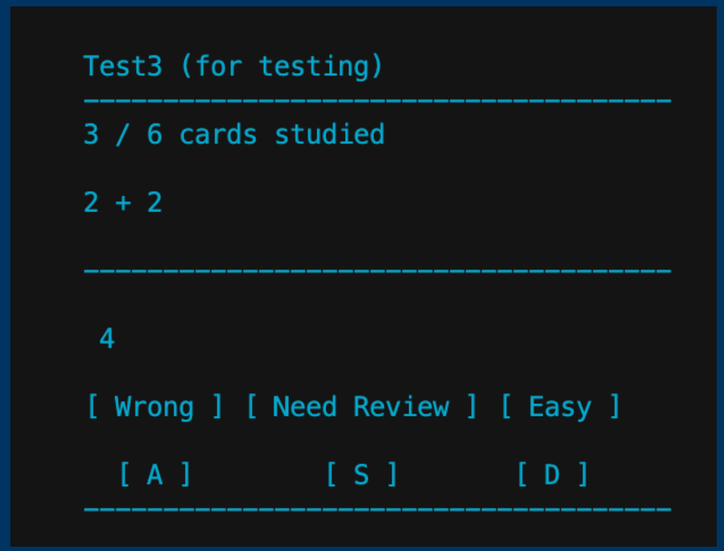
“A” was selected on previous screen so that card 1 will be shown again twice. And you can see the total cards to study have increased by two. The user on this current card will hit “S”.

Card 3 (1+1)



“S” was selected so the total has gone up one meaning card 2 will be repeated once. Now a new card is displayed and its card 3. We know the answer so so we will hit “D” on this card.

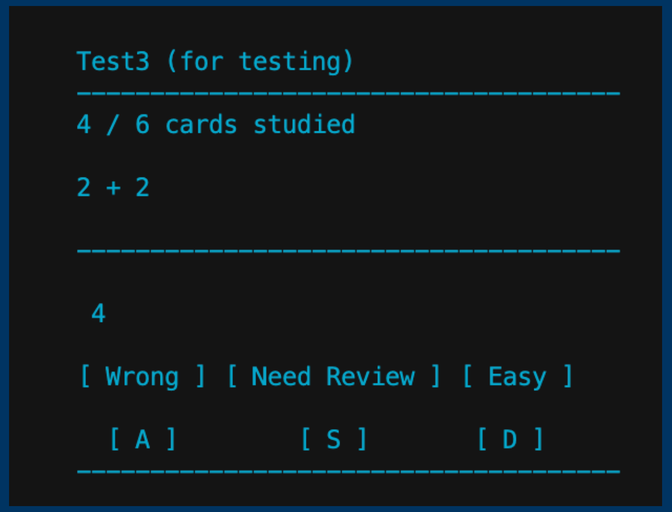
Card 1 (1st rep)



Card 1 (1st repeat appears in our deck again. We know the answer so we hit “D” for easy. Note the number of total cards does not go up, when the answer is easy “D” to the user.

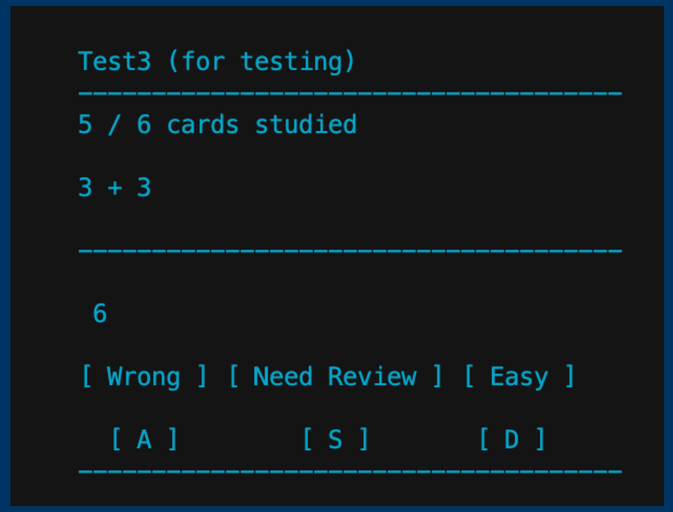
Study a Deck (Feature 1 - Continued from previous page

Card 1 (2nd rep)



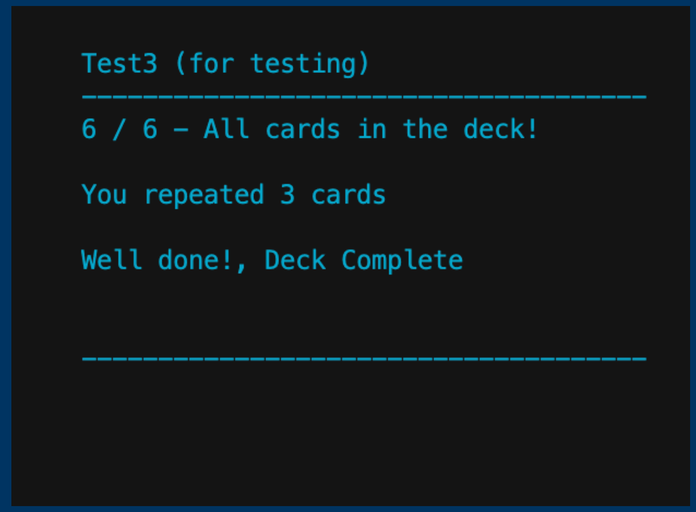
The second repeat for card 1 is shown. And we will hit “D” as we remember the answer.

Card 2 (1st and only rep)



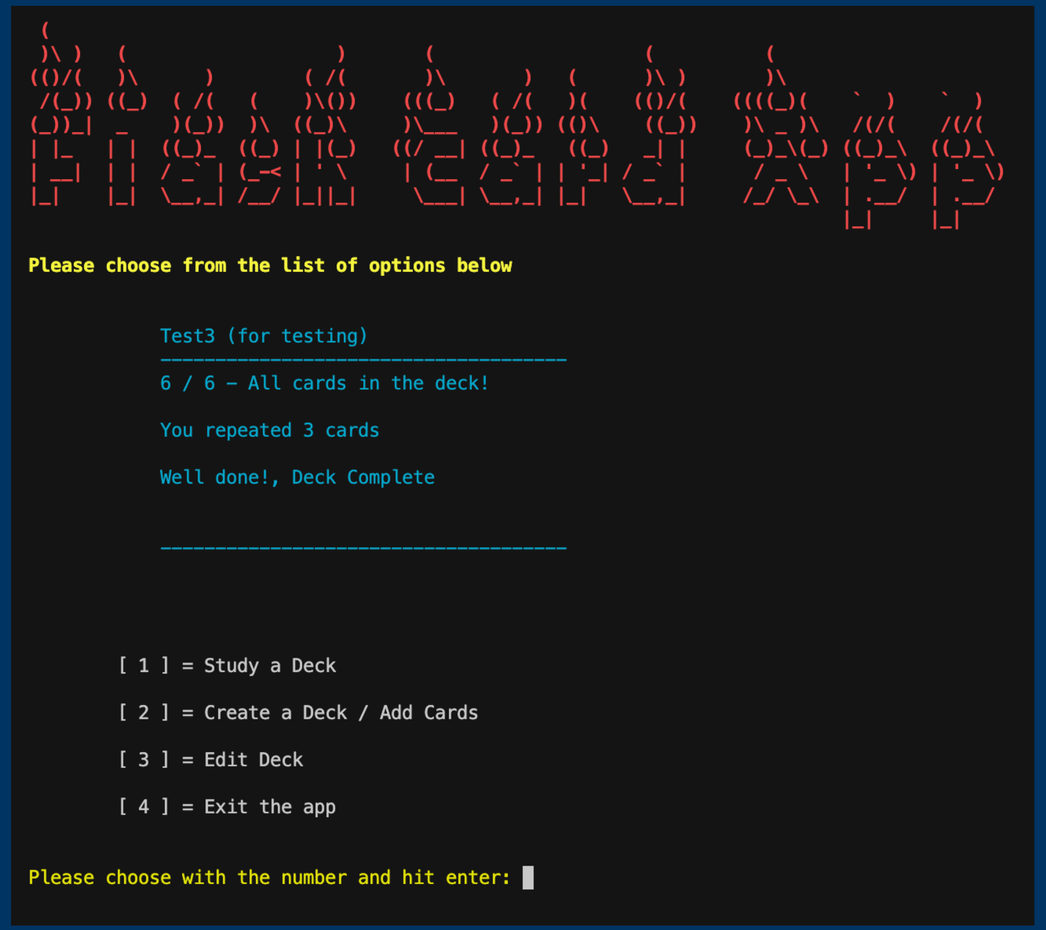
The first and only repeat for card 2 because we hit “S” - Need Review. We hit “D” on this current card.

Deck Completed



Deck finishes and displays a message that we finished the deck. It tallied how many cards we repeated as well. Card 2 - 1 repeat (“S”) and Card 1 - 2 repeats(“A”).

Study a Deck (Feature 1 - Continued from previous page



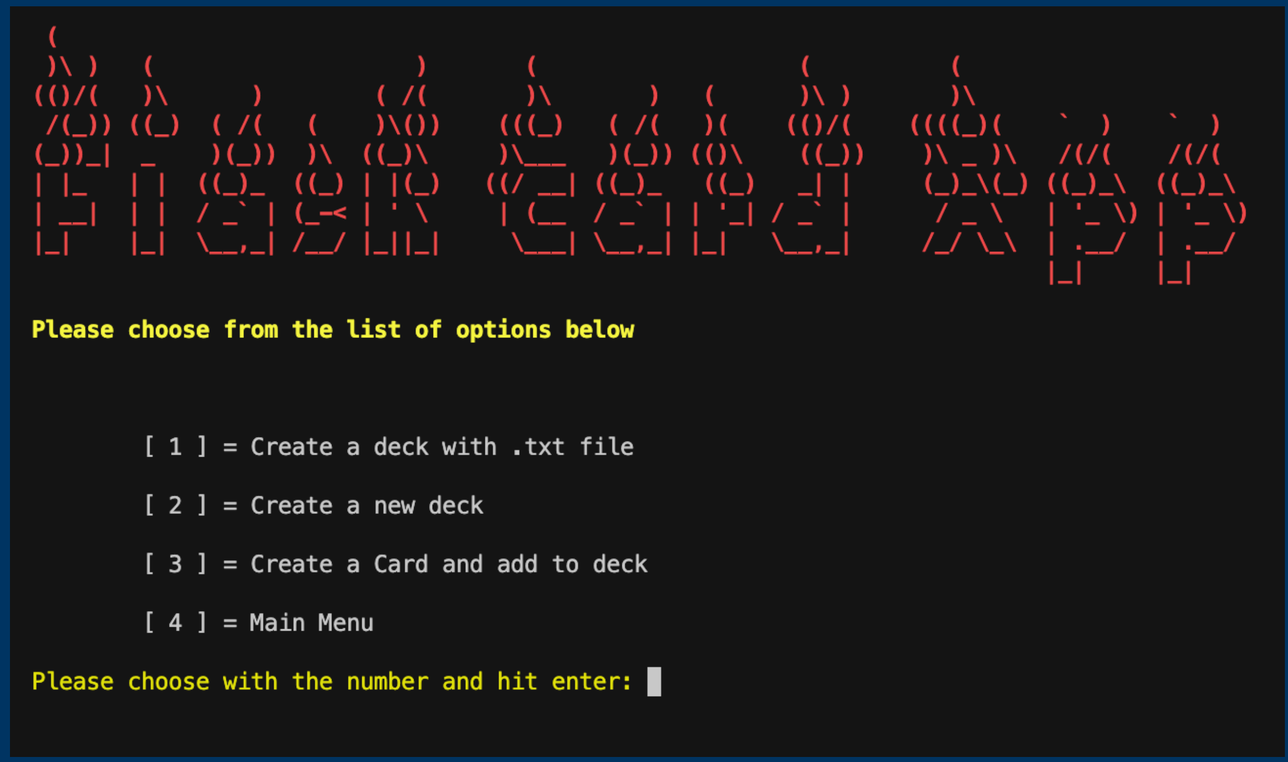
The last card also has reverts back to the main menu options and allows the user to navigate from there.

In summary, the user determines how well they remembered the cards content and based on this, can determine if they want to repeat twice, once or not again. This feature helps the user to remember the content on the card based on their own understanding of the answer.

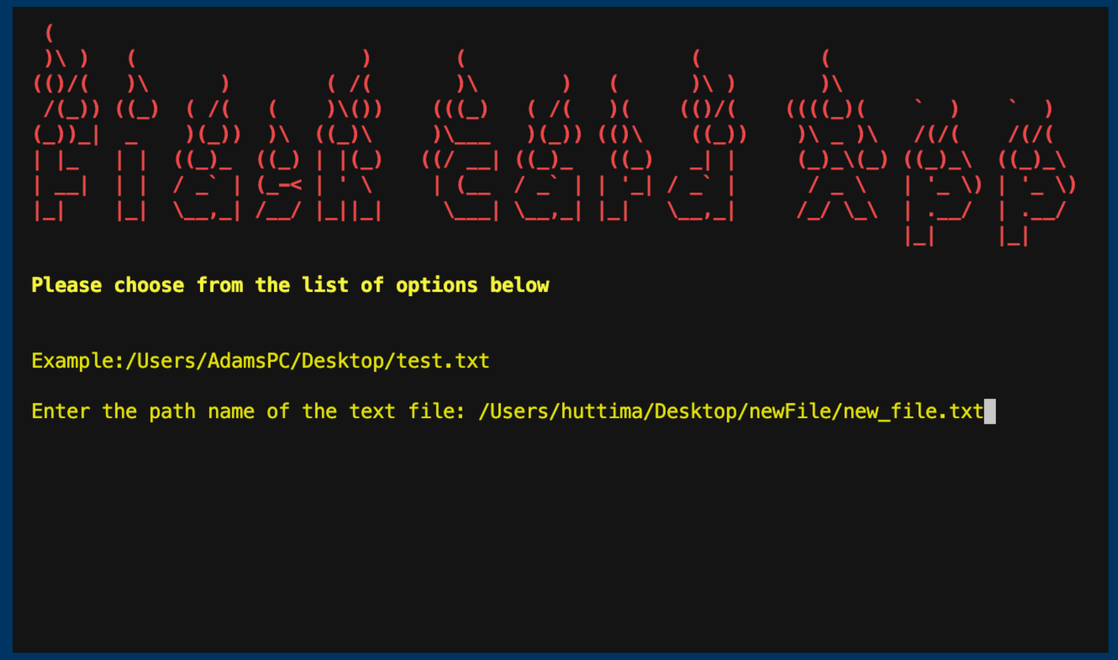
Create a Deck /Card (Feature 2)

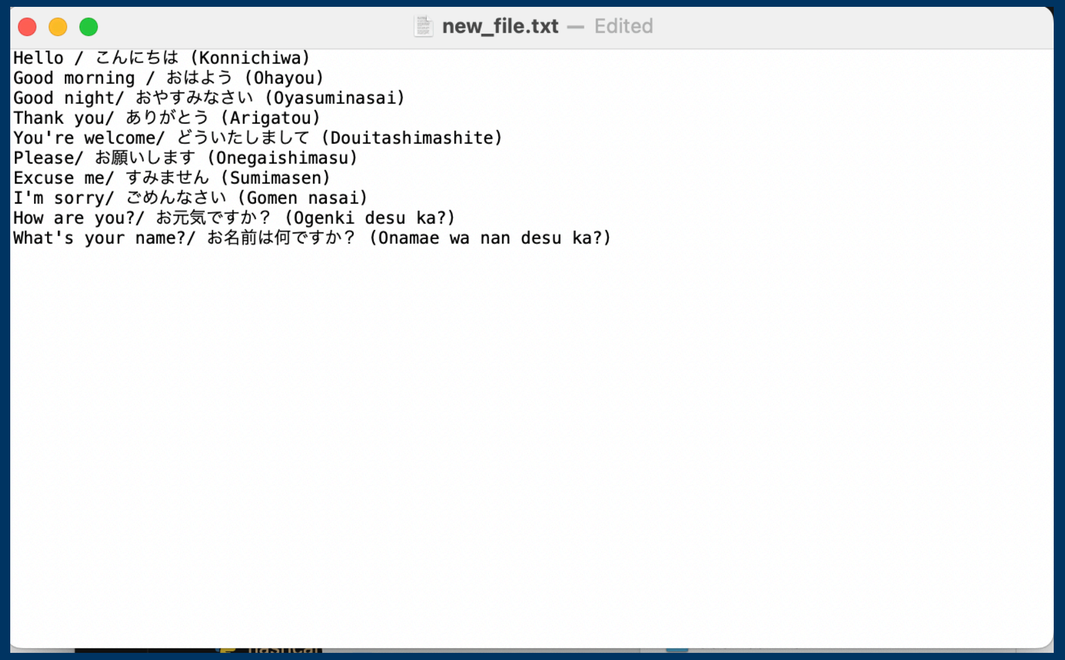
Feature 2 is creating a deck and the cards within the deck. This is the create a deck menu. The user has two options to create a deck. The first is uploading a .txt file (option 1) which automatically splits based on a “/ “ symbol within the file to seperate front and back. The 2nd method is to create the deck manually with option 2 and then again manually add cards to it, using option 3. Option 1 is the quicker method as it creates multiple cards at once and the user can type in one file. Option 3 is to be used to update any deck the user has and add cards once the user fully remembers everything in a deck.

Both features have different purposes but overall help the user to customise their study to their required liking.



Create a Deck /Card (Feature 2) - .txt file

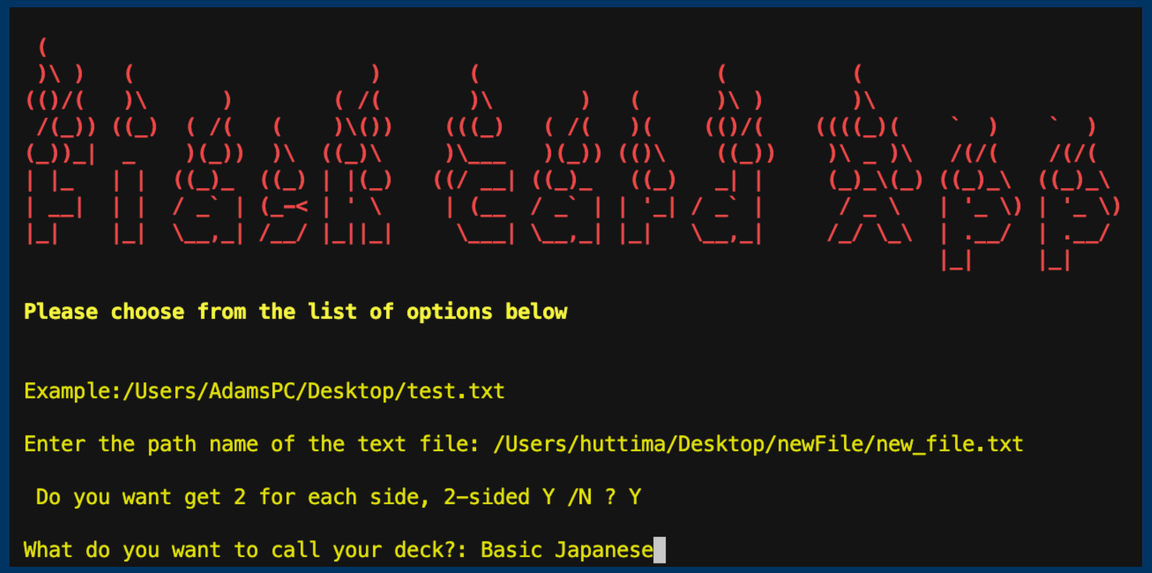




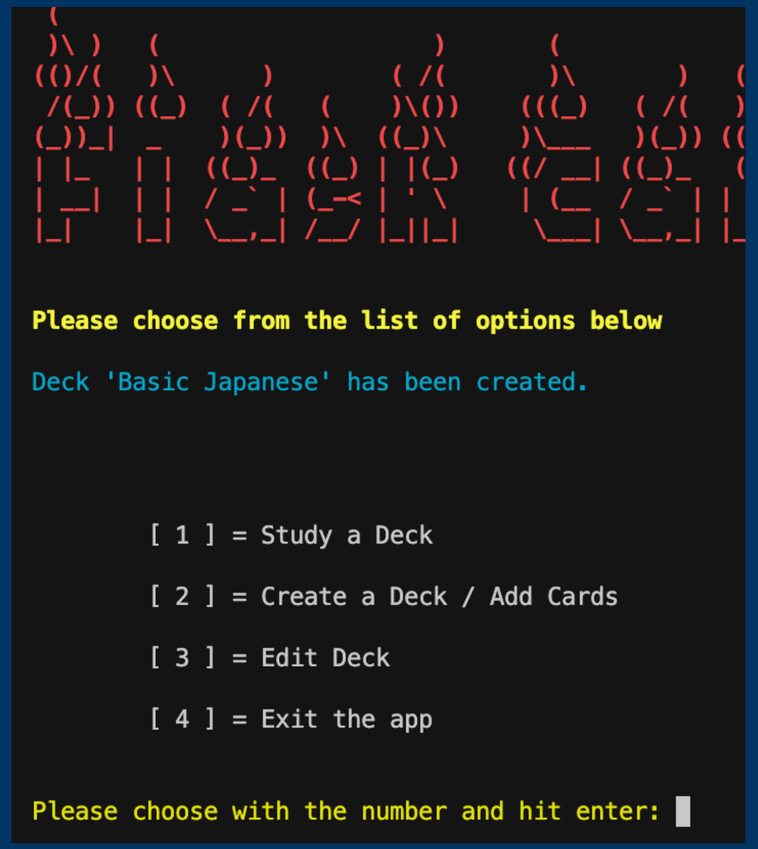
This is the create with .txt file screen. The user enters the directory of the text file including the .txt and hits enter. There is also an example of an address for he user to get an idea of the syntax of a file path. Ive typed in and example there.

Each line will create a card, as mentioned before, the “/“ character is used to seperate between front and back. So this document once uploaded will create 10 cards (10 lines)

Create a Deck /Card (Feature 2) - .txt file

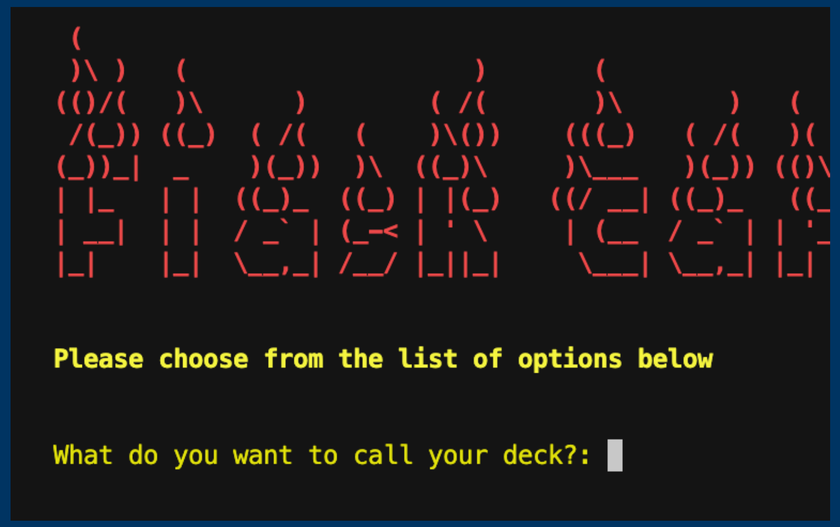


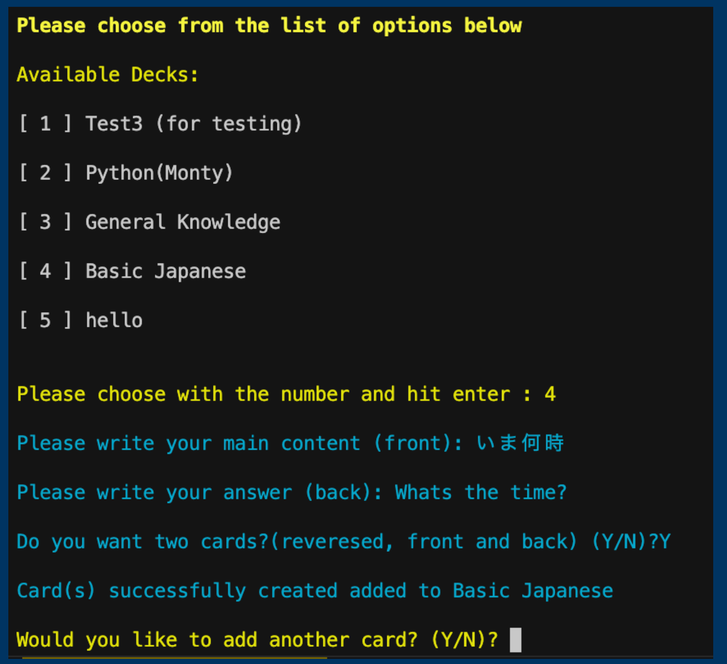
After entering a valid file path , the user is prompted if they want two cards for each line in the .txt file, this means reversed front and back. This is especially useful for language study, so from previous .txt, answering “Y” will create 20 cards, rather than 10. After advising Y/N, it prompts the user to enter the name of the deck.



After entering a valid name and answering the prompts the user receives a message being advised of the deck creating and now can be studied in the study menu.

Create a Deck /Card (Feature 2) - Create Deck / Add cards



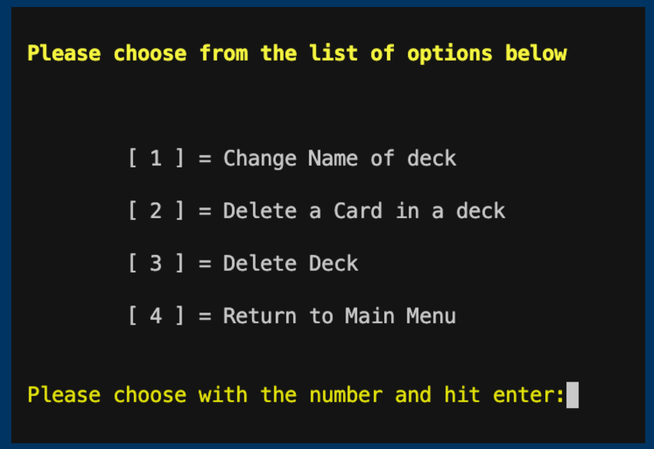


This is the screen for creating a deck, once the user enters a valid name, an empty deck is created. The user cannot study this deck though as it doesn’t have any cards as yet. The user can manually create a card and add it to a deck, like so.

This screen is for creating and adding a card, as you can see the user selects a deck, inputs front content, inputs back, decides whether to reverse and then creates and adds to the selected deck. The user is then prompted to create another one and starts the process again. This is useful for when the user already knows a deck well and wants to add more information to learn.

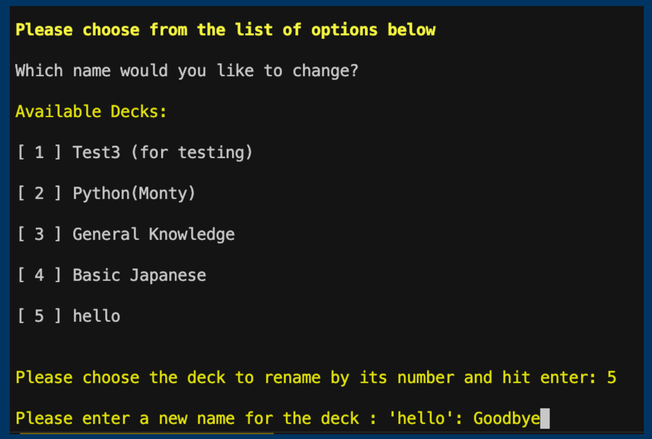
Edit a Deck (Feature 3) -Change deck name, Delete deck, Delete card

Edit Menu



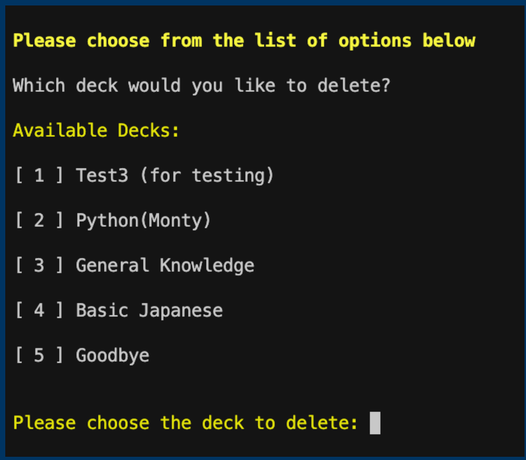
This is the screen for creating a deck, once the user enters a valid name, an empty deck is created. The user cannot study this deck though as it doesn't have any cards as yet. The user can manually create a card and add it to a deck, like so.

Change Deck name



This is the screen for option 1, the user selects the deck they want to change the name of, and then enter a valid name to change to. It will not accept a double as a name . This can be used if the subject of the deck changes due to the user adding more cards.

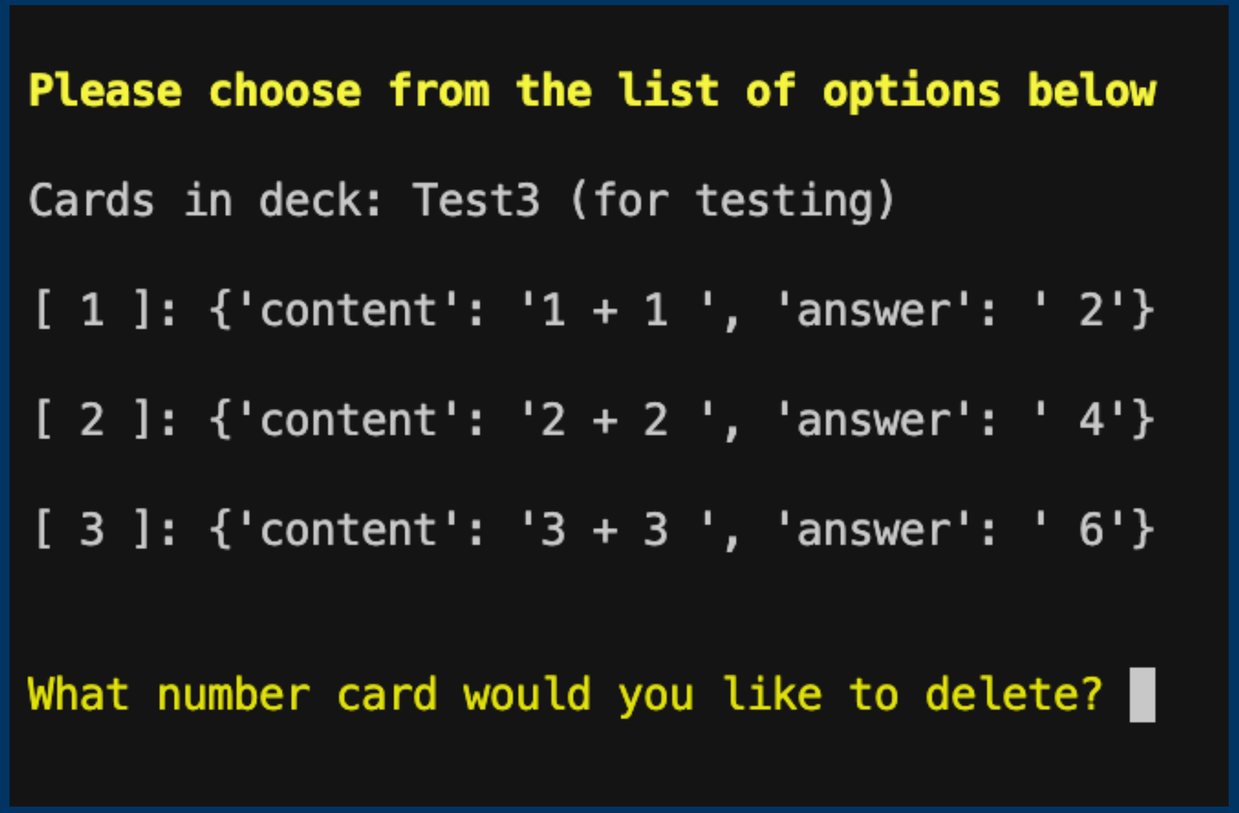
Delete Deck



This is the screen for option 3. The user selects the deck they want to delete and inputs it and the deck is deleted and then displays message saying deleted.

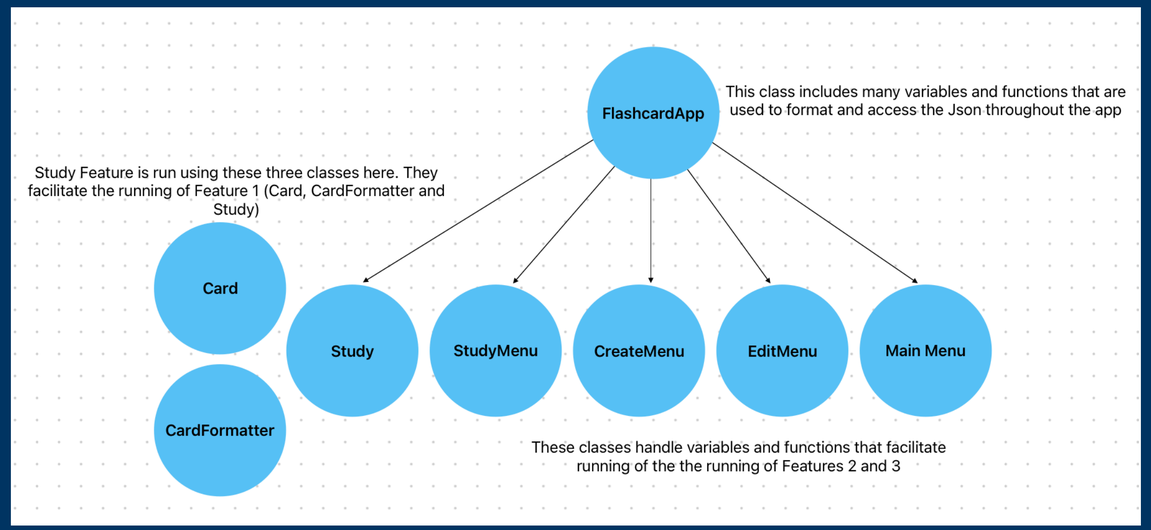
Edit a Deck (Feature 3) -Change deck name, Delete deck, Delete card

Delete card



This is the screen for deleting a particular card in a deck, The user selects the deck, and then is prompted with a list of all the cards in the selected deck like the photo to the left. The user than selects the card they want to delete, inputs there choice and then receives a message saying the card has been deleted. This is useful if the user makes a mistake when creating a card or wants to re-do a card and add information to a new car.

Code Overview




```
def display_available_decks(self):
    with open('decks.json', 'r') as json_file:
        self.x = json.load(json_file)

    self.ava_deck_names = []

    for card_name_deck in self.x:
        for self.single_deck in card_name_deck.keys():
            self.ava_deck_names.append(self.single_deck)

    if not self.ava_deck_names:
        self.post_function_options(self.c_text + "There are no decks available"
                                                 " - please create one first")
    else:
        print(self.y_text + "Available Decks:\n")
        for i, key in enumerate(self.ava_deck_names, start=1):
            print(f"[ {i} ] {key}\n")
```

This is the basic layout of my code. As mentioned I implemented this code using OOP. I had the the FlashcardApps variables and functions (objects) be used throughout the app. I found this difficult at first but realised how it assisted in making my code dryer and saved me doubling up on similar code I had already written. Some examples of code used throughout are display_available_decks used in the CreateMenu , Edit Menu and StudyMenu. post_function options was another really useful function that sends the user back to the MainMenu when something has been created or implemented and also takes an string as an argument to print a message confirming the action as been completed.

Feature 1 - Study a Deck

```python
class StudyMenu(FlashCardApp):
    def __init__(self):
        super().__init__()
        self.change_screen_to_menu()
        self.display_available_decks()

        selected_index = int(input(self.y_text +"\nPlease choose with the number"
                                   " and hit enter: "))
        if 1 <= selected_index <= len(self.ava_deck_names):
            selected_deck = self.x[selected_index - 1]
            for sd_val in selected_deck.values():
                if sd_val == [{}]:
                    self.post_function_options(self.c_text + "That deck has no "
                                               "cards, please add cards to use")
                else:
                    CardFormatter(selected_deck)
        else:
            (self.y_text + "Please choose with the number")
```

```python
class CardFormatter:
    def __init__(self,deck):

        formatted_card_bank = []

        for k,v in deck.items():
            deck_name = k
            for card in v:
                card_front = card["content"]
                card_back = card["answer"]
                new_card = Card(card_front, card_back, deck_name)
                formatted_card_bank.append(new_card)
            # this list is the info as objects
            study_deck = Study(formatted_card_bank)
            study_deck.run_deck()
            #run through the study function
```

```python
class Card:
    '''makes an object with required information'''
    def __init__(self, front, back, deck_name):
        self.front = front
        self.back = back
        self.deck_name = deck_name
```

```python
def run_deck(self):
    ''' Runs formmated deck and allows user to repeat or pass'''
    deck_len_ext = len(self.shuffled_deck)

    while self.card_num < deck_len_ext:
        current_card = self.shuffled_deck[self.card_num]
        clear()
        print(self.c_text +f'''
        {current_card.deck_name}
        ---------------------------------------
        {self.card_num} / {len(self.shuffled_deck)} cards studied

        {current_card.front}

        ---------------------------------------
        ''')
        show_card_input = input(self.y_text +"\nPress 'Z' to show"
                                " the answer: ").lower()

        if show_card_input == 'z':
            clear()
            print(self.c_text + f'''
        {current_card.deck_name}
        ---------------------------------------
        {self.card_num} / {len(self.shuffled_deck)} cards studied

        {current_card.front}


        ---------------------------------------

        {current_card.back}

        [ Wrong ] [ Need Review ] [ Easy ]

          [ A ]       [ S ]      [ D ]
        ---------------------------------------
            ''')
```

```python
    next_card = input(self.y_text +"Press 'A' or 'S' to shuffle back"
                      " in deck. 'D' to move to the next card."
                      " 'Q' to exit: ").lower()

    if next_card == 'q':
        self.post_function_options("You quit the deck")
    elif next_card == 's':
        self.shuffled_deck.append(self.shuffled_deck[self.card_num])
        deck_len_ext += 1
        self.card_num += 1
    elif next_card == 'a':
        self.shuffled_deck.append(self.shuffled_deck[self.card_num])
        self.shuffled_deck.append(self.shuffled_deck[self.card_num])
        deck_len_ext += 2
        self.card_num += 1
    elif next_card == 'd':
        self.card_num += 1

    # Incorrect input goes to start of while loop

message =(self.c_text + f'''
{current_card.deck_name}
---------------------------------------
{self.card_num} / {len(self.shuffled_deck)} - All cards in the deck!

You repeated {self.card_num - len(self.for_deck)} cards

Well done!, Deck Complete
```

The study feature begins in the study menu class. Once the user selects the deck from the study menu. The Cardformatter Class is initilized and with the selected deck (a dict) and loops through it and takes the necessary information (Front, Back, Deck Name) and is formatted into an object in the Card class. The cards are then appended to an empty list called formatted_card_bank. From here this list of objects (cards) is then made into an instance called study_deck. And then this instance is run through another class called Study. In study the function run_deck is run and prints the information from each object in the card shape. Run Deck function then utilizes an if/elif inside a while loop to gather the users input and add cards and progress through the list of objects. The while loop also handles any unexpected inputs from the user and refreshes the card screen.

Feature 2 - Create a Deck with txt file.

The CreateMenu class handles all variables a functions related to the options displayed in the Create menu. Depending on the users choice it runs a function that facilitates the request. For example for Create a Deck with .txt feature, the user selects the option from the menu and then, begins by creating the deck in the self.create_deck_from_file function. This function gets the lines from the file from a separate function “get_lines_from_file” and uses the os package to check the file path is correct from the user and search for .txt file.

If correctly located, it will open and read the lines and return them. The returned lines are assigned to a variable called lines_from_file in the create_deck from file. From this point a for loop is used to read the lines and split based on a slash ”\” and assign the split line into two variables - content and answer. The user is then asked if they want 2-sided cards and they can respond yes or no , either way the variables are run through a seperate function which formats the content and answer into a dicitonary. The information in the a dictionary (cards) are appended to an empty list which is the deck which is passed into push_deck_to_json.

The deck is now created and the function push_deck_to _json can now be run with the deck (list from self.create_deck_from_file). This function is a while loop which takes uses an input to get the deck name, checks the name doesn’t exist with a for loop, if statement and boolean values. The file is then formatted to be {deck name : [{ 1+1 }, { 2 }] } and then appended to the open Json file and then written into the Json file.

Feature 2 - Create a Deck with txt file. (Photos)

```
if create_menu_choice == 1:
    self.change_screen_to_menu()
    self.push_deck_to_json(self.create_deck_from_file())
```

```
def create_deck_from_file(self):
    new_deck =[]
    print( self.y_text +"\nExample:/Users/AdamsPC/Desktop/test.txt\n")
    file_path = input(self.y_text +"Enter the path name of the text file: ")
    lines_from_file = self.get_lines_from_file(file_path)
    try:
        txt_inp_swap = input("\n Do you want get 2 for each side, "
        "2-sided Y/N ?: ")

        for line in lines_from_file:
            content, answer = line.strip().split('/')

            if txt_inp_swap.lower() == "y":
                new_deck.append(self.create_and_reverse(content, answer))
                new_deck.append(self.create_and_reverse(answer, content))
            elif txt_inp_swap.lower() == "n":
                new_deck.append(self.create_and_reverse(content, answer))
            else:
                ("Invalid input. Please enter 'Y' or 'N'.")


    except ValueError:
        self.post_function_options("Formatting error, please check .txt "
                                   "file and ensure  '/' used once in"
                                   " a line to seperate ")
    except TypeError:
        self.post_function_options("Please ensure file is .txt")

    return new_deck
```

```
def push_deck_to_json(self,deck_cards):
    while True:
        name_input = input(self.y_text +"\nWhat do you want to call"
                                        " your deck?: ")
        with open('decks.json', 'r') as json_file:
            d = json.load(json_file)

        name_exists = False
        for deck in d:
            if name_input in deck.keys():
                name_exists = True
                break

        if not name_exists:
            template_created_dict = {name_input: deck_cards}
            d.append(template_created_dict)

            with open('decks.json', 'w') as json_file:
                json.dump(d, json_file, indent=2)

            self.post_function_options(f"Deck '{name_input}' has"
                                       " been created.\n")
            break
        else:
            print("\nThis name already exists. Please enter another\n")
```

```
def get_lines_from_file(self,file_path):
    try:
        while True:
            if os.path.isfile(file_path):
                with open(file_path, 'r') as c:
                    return c.readlines()
            else:
                self.post_function_options("Invalid file address , Please"
                                           " check and enter again.")

    except ValueError:
        print("Please reformat the text file must be - 'front / back'"
              " - and another card on a new line ")
```

```
def create_and_reverse(self,content,answer):
    card_info ={
        "content":content,
        "answer": answer
        }
    return card_info
```

Feature 3 - Delete a Card in a deck

The EditMenu class handles all editing based features on the app. I will walk through the logic of the “Delete a card in a deck” feature.

Once the user selects to Delete a card from a deck. It intializes an inherited function from the FlashcardApp class which generates the header and clears the screen. It then prints a statement asking for the deck the user wants to delete from. After this another inherited function displays all the decks available to amend. And finally the delete_card_in_deck function is run and the an input is requested to select the deck.

From here a for loop in selected_deck.items() is run, gathering the key and the values , if there is no cards in that deck, the user will return to the main menu and displayed a message, informing them them there is no cards in this deck.

The screen is then cleared and using a for /enumerate loop we display all the cards from the selected deck numbered and use an input function for the user to select one. Planning the input is valid a variable is created and the selected card {dict} is removed from the list using .pop . Following this the new information inherited from the FlashcardApp class (self.x) is opened and written into (updated). Removing the users selected card from the json file.

Feature 3 - Delete a Card in a deck (Photos)

```
def delete_card_in_deck(self):
    while True:
        try:
            del_card_deck_choice = int(input(self.y_text + "\nPlease choose "
                                             "the deck the card contains:"
                                             " "))
            if 1 <= del_card_deck_choice <= len(self.ava_deck_names):
                selected_deck = self.x[del_card_deck_choice - 1]
                for key, value in selected_deck.items():
                    if not value:
                        self.post_function_options("There are no cards "
                                                   "in that deck")
                        break

                    self.change_screen_to_menu()
                    print(f"Cards in deck: {key}\n")
                    for i, deck in enumerate(value, start=1):
                        print(f"[ {i} ]: {deck}\n")

                    edit_c_number = int(input(self.y_text + "\nWhat number ca"
                                             "rd would you like to "
                                             "delete? "))
                    if edit_c_number == 0:
                        break

                    if 1 <= edit_c_number <= len(value):
                        deleted_card = value.pop(edit_c_number - 1)
                        with open('decks.json', 'w') as json_file:
                            json.dump(self.x, json_file, indent=2)
                        self.post_function_options(f"Card {edit_c_number}: {deleted_card}
                    else:
                        print("Invalid card number. Please enter a valid "
                              "number.")

                    break
                else:
                    print("Invalid deck choice. Please enter a valid deck"
                          " number.")
        except ValueError:
            print("Invalid input. Please enter a number")
```

```
elif edit_menu_choice == 2:
    self.change_screen_to_menu()
    print("Which deck is the card you would you like to"
          " delete? \n")
    self.display_available_decks()
    self.delete_card_in_deck()
```

Error Handling

My code has error handling throughout. This is to ensure the app doesn't crash or exit until the user selects to exit . Ive used methods like putting in a while loop, if /else statements and using try statements with exceptions. Please see examples below.

```
def get_lines_from_file(self,file_path):
    try:
        while True:
            if os.path.isfile(file_path):
                with open(file_path, 'r') as c:
                    return c.readlines()
            else:
                self.post_function_options("Invalid file address , Please"
                                           " check and enter again.")

    except ValueError:
        print("Please reformat the text file must be - 'front / back'"
              " - and another card on a new line ")
```

```
def create_deck_from_file(self):
    new_deck =[]
    print( self.y_text +"\nExample:/Users/AdamsPC/Desktop/test.txt\n")
    file_path = input(self.y_text +"Enter the path name of the text file: ")
    lines_from_file = self.get_lines_from_file(file_path)
    try:
        txt_inp_swap = input("\n Do you want get 2 for each side, "
        "2-sided Y/N ?: ")

        for line in lines_from_file:
            content, answer = line.strip().split('/')

            if txt_inp_swap.lower() == "y":
                new_deck.append(self.create_and_reverse(content, answer))
                new_deck.append(self.create_and_reverse(answer, content))
            elif txt_inp_swap.lower() == "n":
                new_deck.append(self.create_and_reverse(content, answer))
            else:
                ("Invalid input. Please enter 'Y' or 'N'.")


    except ValueError:
        self.post_function_options("Formatting error, please check .txt "
                                   "file and ensure  '/' used once in"
                                   " a line to seperate ")
    except TypeError:
        self.post_function_options("Please ensure file is .txt")
```

```
while True:
    try:
        user_main_choice = int(input(self.y_text +"\nPlease choose your "
                                   "option with the number and hit"
                                   " enter: "))
        if user_main_choice == 1:
            StudyMenu()
        elif user_main_choice == 2:
            CreateMenu()
        elif user_main_choice == 3:
            EditMenu()
        elif user_main_choice == 4:
            sys.exit()
        else:
            print("Invalid Input, please try again")

    except ValueError:
        print("Invalid choice. Please choose using the numbers.")
    except UnboundLocalError:
        self.post_function_options("Theres no cards in this deck.")
```

Challenges, Favourite Parts

- I had originally planned to install a keyboard listener package and use it in the Study a deck feature. I installed and coded it correctly but the terminal screen was flickering and was causing some problems with my already installed exception handling, so I decided to take out as I already had 4 packages.

- I enjoyed using OOP and got a good idea of how it functions. Although I haven't used all its features and utilized it to its full potential , it was enjoyable regardless.

- Fixing bugs was quite a stressful task , but I used the debugging function built-in to VS code. I enjoyed the step-into button , so you could see where it was going run. I also found stack overflow a useful website which had some useful information.

- I took a more structured , planned approach compared to the portfolio due to the task requirements and was less stressful as I could see where I was up to and what I needed to complete by certain times.

- I found myself getting distracted and continuing with trying to make other features even though they weren't necessary, just because I wanted to challenge and make sure I could do it. That's why I have over three features

Thanks