

Card Game

Computing NEA Practice

Deadline 25/04/23

HARRISON WRIGHT

Problem and Analysis	5
Game Identification	5
The rules:	5
Limitations	5
Design plan	5
Problem Decomposition	6
Potential solutions	7
Chosen solution and justification	7
Design	8
Overall system design	8
Flow Chart of the proposed game	8
Login subroutine	9
Register Subroutine	10
Calculate Winner Subroutine	11
Update Leaderboard Subroutine	12
Display Leaderboard Subroutine	12
Database data requirements	13
Input, process, storage and output	14
Form structure	14
Login form	14
Register form	14
Validation	15
Creation	16
System security and integrity of data	16
Password protection	16
Encryption	16
User interface design	17
Leaderboard page	20
Functions and Classes Pseudocode	24
Functions (Database and Game focused)	24
Classes (GUI focussed)	28
Test plan	41
Testing	41
Login Page:	41
Registration Page:	42
Procedure and Classes listing	44
Important Variables listing	46
How does the system meet the objectives	47
Improvements & Extensions to the game	47
Evaluation	48

Problem and Analysis

Game Identification

The game has a deck of 30 unique cards. There are 3 card colours (red, black or yellow) and each card has a number (1, 2, 3, 4, 5, 6, 7, 8, 9, 10).

The 30 cards are then shuffled and stored in the deck.

The rules:

- Player 1 takes the top card from the deck and Player 2 takes the next card.
- If both players have the same card colour, then the player with the highest number wins.
- If both players have different card colours, the winning colour is shown below:
RED beats **BLACK**
YELLOW beats **RED**
BLACK beats **YELLOW**
- The winner of each round keeps both players cards
- The players keep playing until there are no more cards left

Limitations

There are a range of problems I need to consider when I design and create this system.

- I want this program to be able to run on limited hardware and use the least amount of processing power available.
- I also want the program to be quick to run and not require a long loading time. This will increase user enjoyment as they do not have to waste time.
- I want the program to allow the game to be played multiplayer. I'll have to create a way for two accounts to be able to login and play together as well as still allowing single player capabilities.

Design plan

Things I want to include in my program:

- Only allow authorised players to play. I need to write code to access and write to a database that stores the player names and passwords. Additionally, I want the passwords to be encrypted in some way to have some sort of security.
- Perform all the instructions to carry out the game. This includes: shuffling the deck, dealing the cards and calculating a winner.
- Display a leaderboard. I will need to create another database that stores the top scores and a way to display the top 5 in order to the user. This also means that the

database needs to store the name of the player and their quantity of cards to be accessed later.

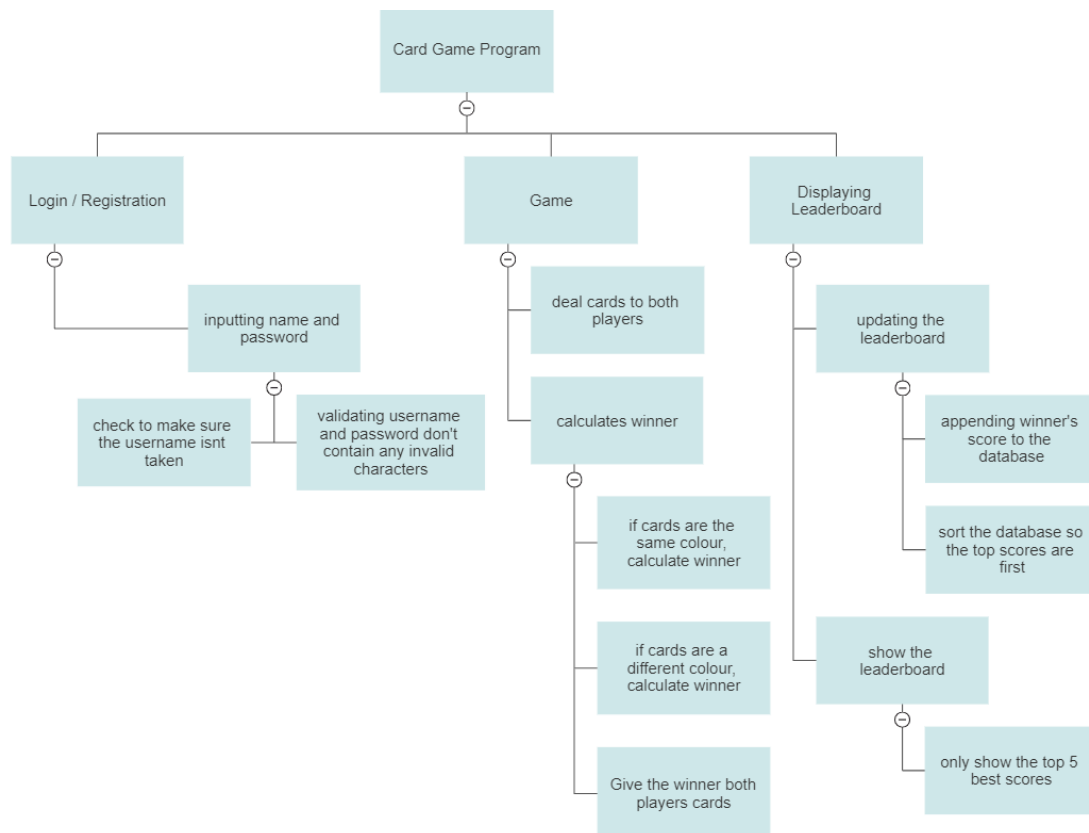
- Some form of security. I do not want to just be storing a file with every password on it in plain text. Instead, I should encrypt the password in some way to make my program more secure.
- I also want my program to be multiplayer and allow for a second account to be able to play as well.

Problem Decomposition

The problem can be decomposed into a set of much smaller steps. This is an initial idea of what these steps are:

- Login / Register system
- A database to store those credentials and a way to read / write to the database
- A fully working game. Including: shuffling deck, dealing cards, calculating winner
- A leaderboard system
- A database to store the leaderboard information and a way to read / write to it

I have also created a hierarchy chart to show the most important elements this in a visual format:



Potential solutions

There are multiple ways in which I can create a suitable card game program. Below are 3 different approaches I could take with their positives and negatives. All the programs need some sort of Database to store data and all systems need to have a Graphical User Interface (GUI).

#	Suggested Solution	Positives	Negatives
1	Python 3.10 + Tkinter + CSV	I already know Python, Tkinter and CSV. No added software is needed.	A log of programming is required, especially Tkinter.
2	Python 3.10 + Flask + CSV	I already know Python and CSV.	I am not very confident with Flask and without proper knowledge of HTML and CSS.
3	Python 3.10 + Tkinter + SQLite3	I already know Python, Tkinter and I know SQL syntax, meaning it will not be too difficult to learn	I do not know how to use SQLite3 and I will have to learn how to implement it.

Chosen solution and justification

I have chosen solution 1 as I already possess the skill set required to create the game with the chosen software.

Other beneficial factors:

- The game will not require any other software
- Tkinter (graphical user interface) is a module of Python so the syntax will be easy to implement.
- Python has lots of resources and a large community to help me.
- Python requires low amounts of processing power to run

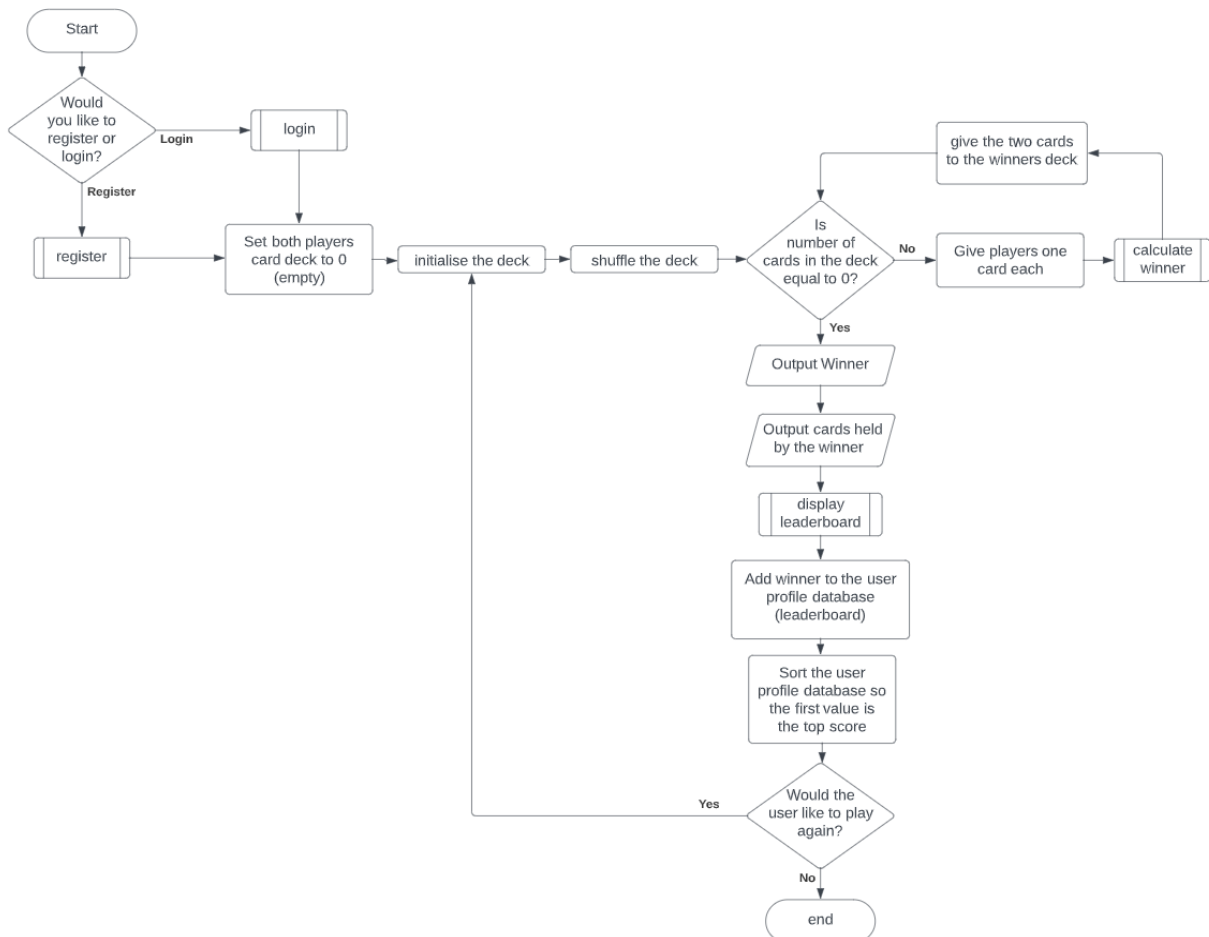
Design

Overall system design

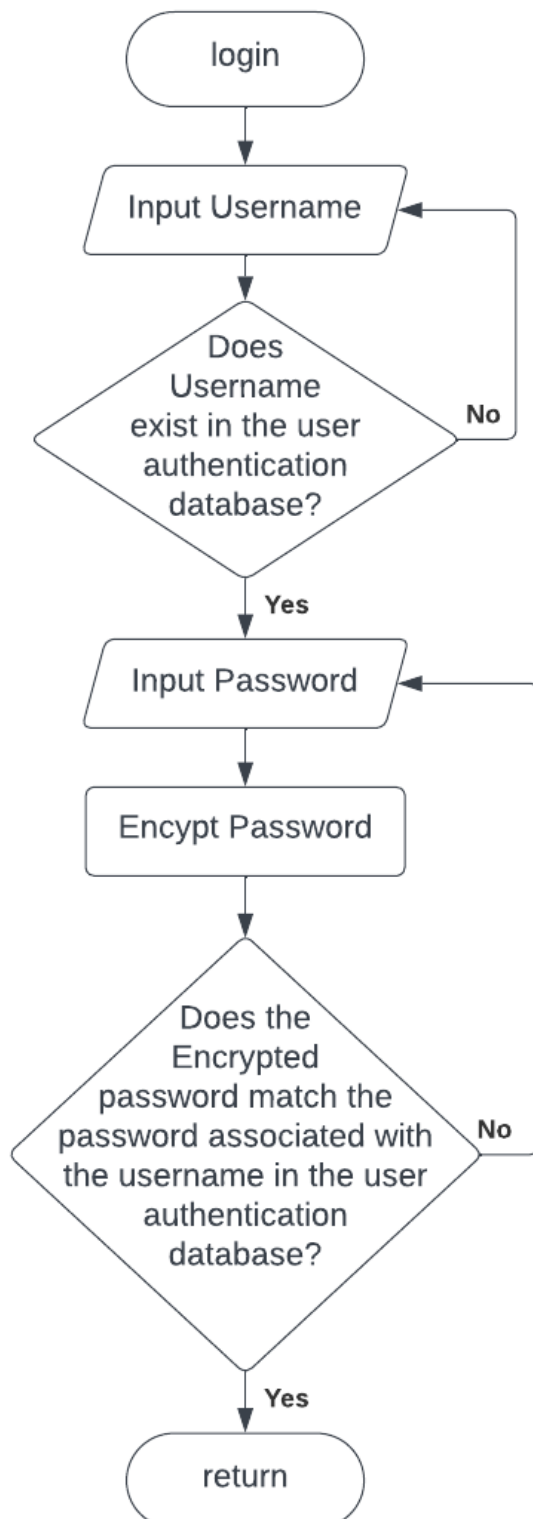
The card game will be password protected so only authenticated users can play. After login, the user will be greeted with a homepage. The homepage will have buttons for different pages, including: “Play”, “How to Play”, “Leaderboard” and a button to also logout. The “Play” button will take the user to the game start and they can begin playing the card game. The “How to Play” button will explain the rules to the player and the leaderboard will display the current top 5 highest scores.

Flow Chart of the proposed game

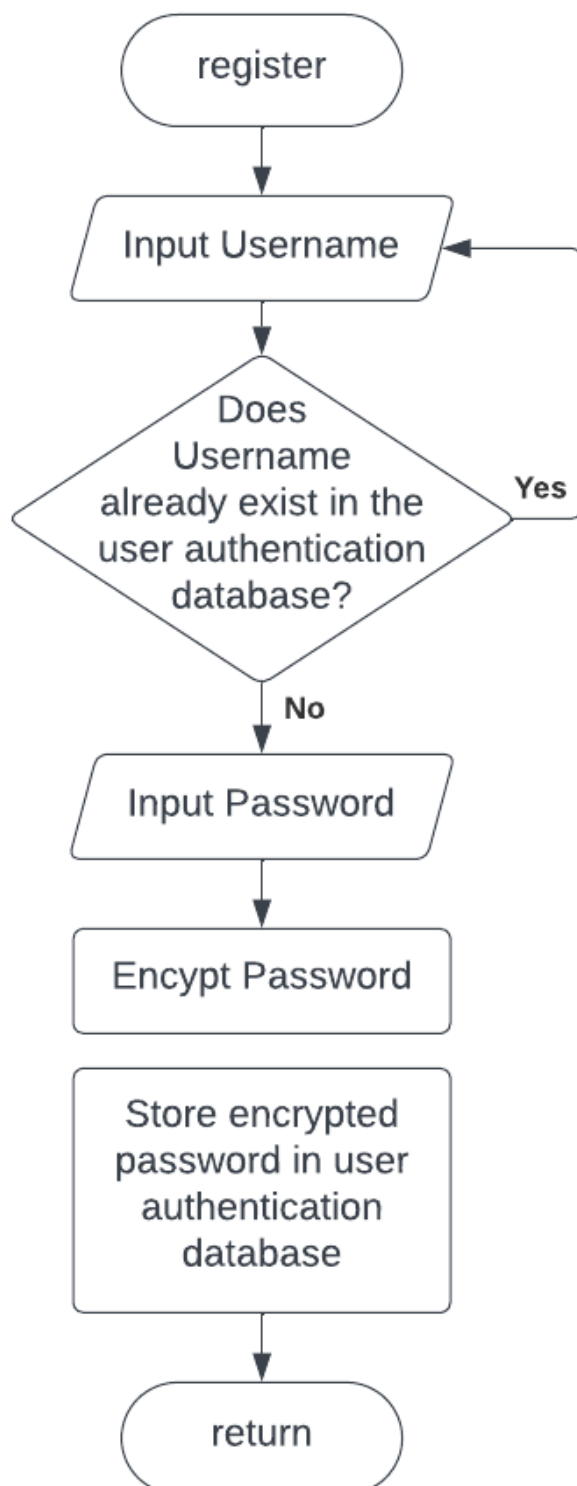
This flowchart shows that the user will be prompted with a login / register screen. After that, the game will begin, running through all the processes until the deck is empty.



Login subroutine



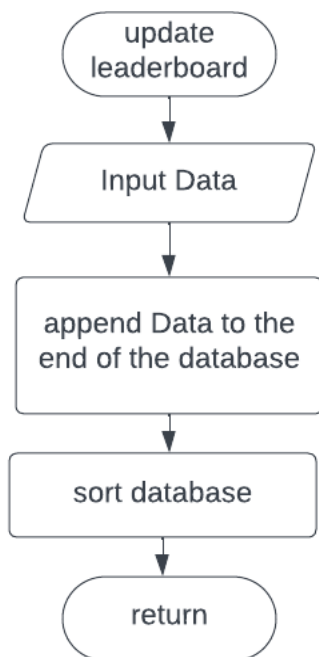
Register Subroutine



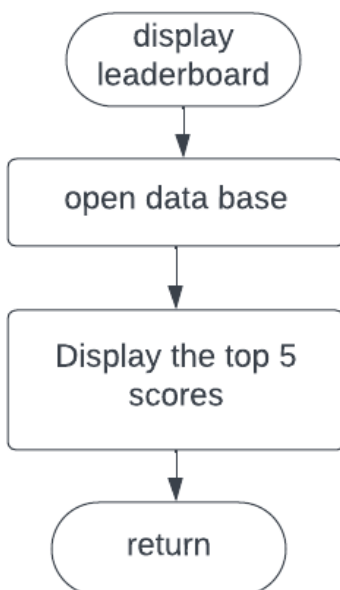
Calculate Winner Subroutine



Update Leaderboard Subroutine



Display Leaderboard Subroutine



Database data requirements

The data used in the program will be put into a database so the information can be stored. The database will be made up of two separate tables and linked by username. All the databases will use the delimiter “|” and be stored as the following:

User Profile Database					
Field Name	Field Purpose	Field Type	Field Size	Example Data	Validation
username	To store the username of the user into the database	String	3-24	Harrison	Only alphabetical characters. Must not be blank. Length between 3-24.
Length of cards	To store the user's length of cards	String	16-30	22	Field must not be blank
List of cards	To store the list of cards that the winner is holding	Array	16-30	[['Black', 10], ['Red', 2], ['Black', 7], ['Red', 8], ['Black', 9], ['Red', 9]]	Field must not be blank

Authentication Database					
Field Name	Field Purpose	Field Type	Field Size	Example Data	Validation
name	To store the username of the user into the database	String	3-24	Harrison	Only alphabetical characters. Must not be blank. Length between 3-24.
password	To store the user's password in an encrypted form	String	4-30	043e2b48	Field must not be blank. Length between 4-30.

Input, process, storage and output

The system is going to need to input and output data. This section shows the inputs below.

Input	Process	Storage	Output
Login <ul style="list-style-type: none">UsernamePassword	Authenticate login credentials. If player1 is not logged in, log in to player1. If player2 is not logged in but player1 is, log in to player2	Stores the Player's name for future use in the game.	Credentials correct <i>or</i> Credentials incorrect
Register <ul style="list-style-type: none">UsernamePassword	Check, if the username entered already, exists and authenticate the login.	Stores the new username and password together. Stores the Player's name for future use in the game.	Credentials correct <i>or</i> Credentials incorrect
Selecting Menu Option <ul style="list-style-type: none">Mouse click on the button	Call function to the selected option.	N/A	Show the selected screen

Form structure

Users can pass information into the system through the login and register forms.

The name of the username will be used to identify the user within the program. If the user has filled out all of the boxes correctly and then presses the submit button, the data will then be saved to the database and saved into variables to be used by the program.

Login form

The login form consists of two input boxes for the user to input their username and password and a button to submit. When the login button is pressed, if the username and password are correct, the main menu will be shown giving users full access to the program. However, if there is an error when logging in, a suitable error message appears.

Register form

The register form consists of two input boxes for the user to input their username and password and a button to submit. When the register button is pressed, if the username has not previously been used, then the main menu will be shown giving users full access to the program. However, if there is an error when logging in, a suitable error message appears.

Validation

Validation and verification checks will be used to ensure the user has inputted data that is correct. Certain fields will only allow you to input alphanumeric symbols or within a certain length.

If the validation checks are completed correctly, the program will save the information to a database. If the validation encounters an error, the data will not be saved and a suitable error message will appear. This will repeat until the user correctly inputs the data.

Validation Check	Description	Example Field	Valid Data	Erroneous data	Error Message
Length	Checks that the data entered has the correct number of characters.	The length of the password creator is greater than 4.	Qwerty12345	123	The length of the data is incorrect.
Range	Checks that the data is within a set range.	The length of the password being between 4-30.	Qwerty12345	35	The data is not inside acceptable range.
Presence	When the input data is compulsory, the user is not able to miss a field out.	Username being entered.	Greenland	(BLANK)	The field needs an input

The user will require feedback when the system encounters an error.

Creation

System security and integrity of data

The security and integrity of data is a major concern when making a coding project. Data needs to be secure.

Password protection

To stop unauthorised access, the entire game will be password protected. When the program starts, there is an immediate login/registration screen. This screen verifies the user's identity. However, passwords can be guessed therefore making them unsafe. This is a big threat to any system.

Encryption

There are many different encryption methods that can be used to protect stored data.

Here is an example of how it works with the password: "qwerty12345".

The user inputs the hypothetical password qwerty12345.

qwerty12345

The function "encrypt_decrypt(password)" is then called and takes a password and a creds object that has a key attribute. It iterates through the characters of the password and uses the ^ operator (XOR) to combine the corresponding characters of the password and creds key. (The creds key is a 31-character string of random characters and is imported at the beginning of the program)

It then appends the result of this operation to a string called result. The chr() function converts an integer to its corresponding ASCII character, while the ord() function converts a character to its corresponding ASCII integer value.

At the end of the loop, the function returns the result string, which represents the XOR result of each character in the password with the corresponding character in the creds key. This operation is commonly used to encrypt and decrypt data and the result can be seen below:

;QL 7↑Du

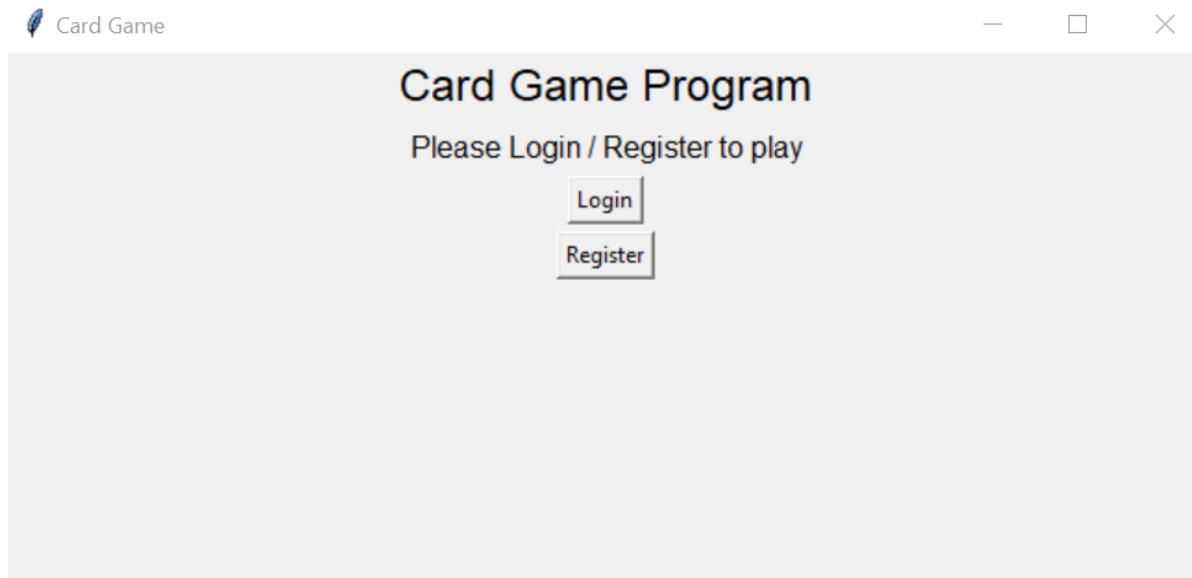
I then use the python in-built module .encode(), transforming the password into this:

b'\x1b<;Q\x1c7\x00\x18\x00Du'

I then use the in-built module .hex(), converting my password value into a long string of hex codes. Below is an example of what qwerty12345 will be saved as in the database.

1b3c3b511c370018004475

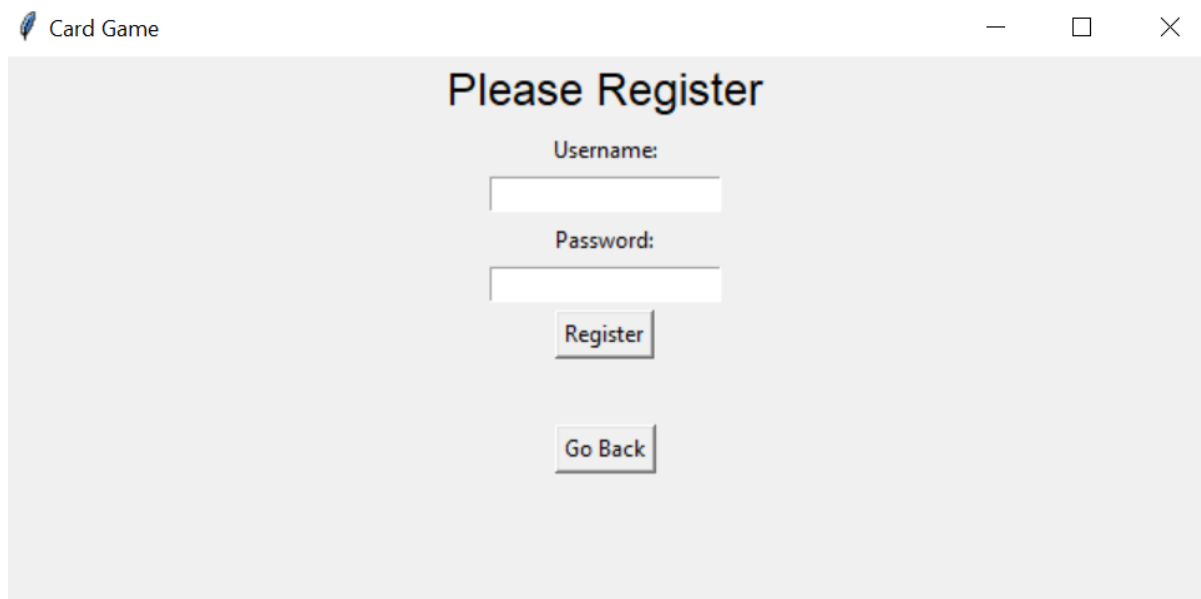
User interface design



Home page

This is the initial home page of the program when it is first run. I wanted the design to be consistent throughout the game to make it as easy as possible for the user to navigate.

When either of those two buttons are clicked it takes the user to that page so they are able to either Login or Register, depending on which they clicked.

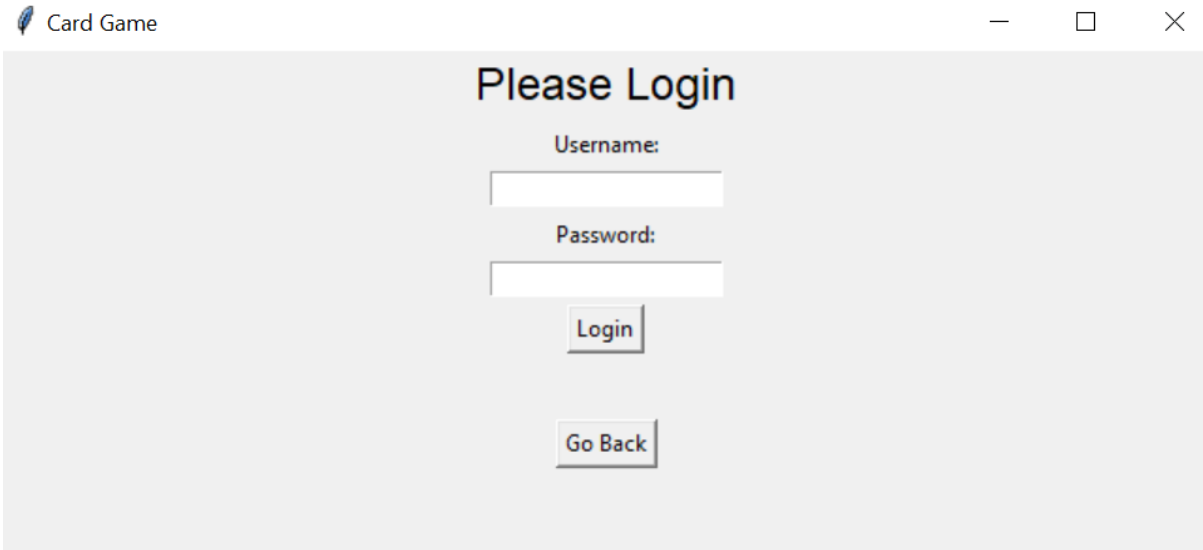


Registration page

The registration page is shown when the Register button is pressed on the home page. It prompts the user to enter their username and password. Presuming the data is entered correctly, once these details have been inputted, by pressing the register button will submit the form and authorise the user.

If the username is too short, too long, uses non-alphanumeric symbols or has already been taken, an appropriate error message is displayed just below the input boxes. I will cover this more later on.

There is also a “Go Back” button to allow the user to go back to the Home page. This is useful because if the user didn’t mean to press register and instead wanted to press login, they can easily correct their mistake.

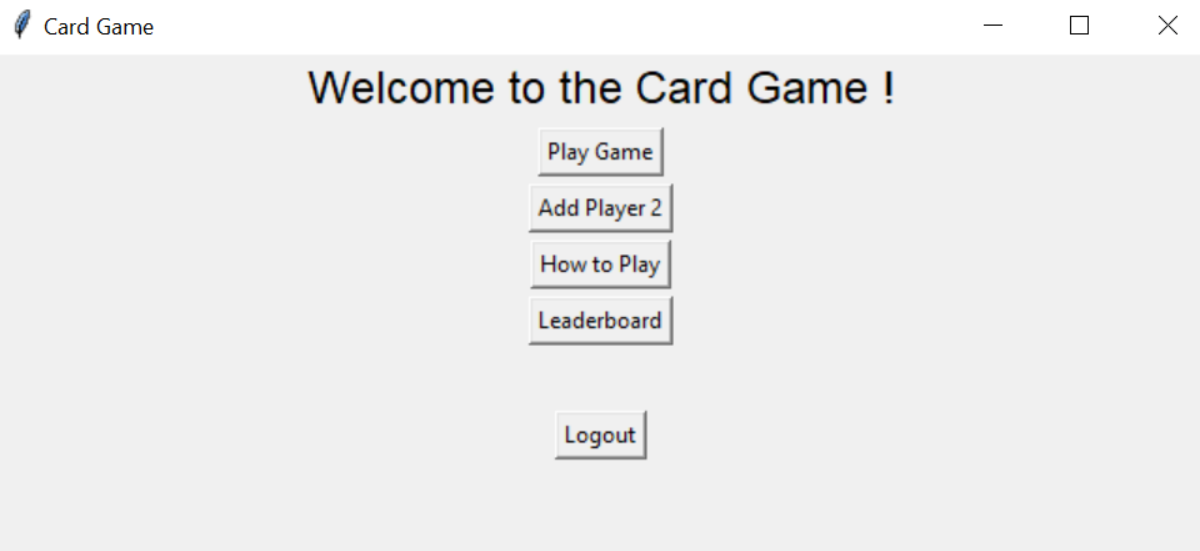
The image shows a web browser window with the title 'Card Game'. The main content area has a light gray background and is titled 'Please Login' in a large, bold, black font. Below the title, there are two input fields: the first is labeled 'Username:' and the second is labeled 'Password:'. Both labels are in a smaller, bold, black font. Below the password field, there are two buttons: 'Login' and 'Go Back', both in a smaller, bold, black font. The 'Login' button is positioned directly below the password field, and the 'Go Back' button is positioned below the 'Login' button. The browser window also shows standard window controls (minimize, maximize, close) in the top right corner.

Login page

The Login page is shown when the Login button is pressed on the Home page. When the Login button is pressed, this page takes the user inputted data from the boxes and opens the database to check that the encrypted passwords are the same. If the passwords are both the same then it authorises the user.

If the user has entered incorrect data, then an error message will appear below the input forms to inform the user.

Additionally, there is a “Go Back” button to allow users to return to the Home page.



Game page

Once the user has been authenticated, they will be taken to this page. They have multiple options to the main features of the game.

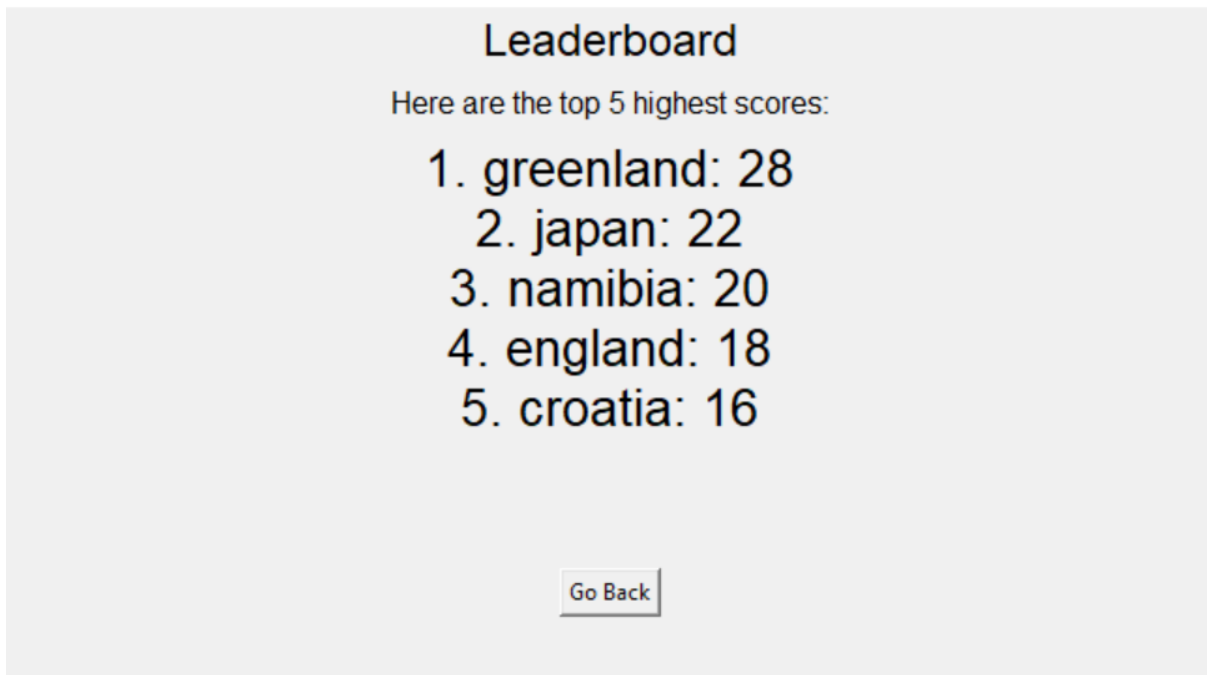
If they would like to begin playing the game, they can press the “Play Game” button. This will immediately enter them into the game.

If they want to add a second player, then the user can press “Add Player 2”. When this button is pressed, it returns the user back to the home page but the registration button is greyed out. This means that the user has to sign in. When they sign in, it assigns Player 2 to the new sign in.

The button “How to Play” takes the user to a detailed screen on the rules of the game. Although the players have no input on the gameplay, I think it is best that users are able to learn how to play the game in their own time.

The “Leaderboard” button takes the user to a screen that displays the current top 5 highest scoring players in a single round. I believe this is better than a running-total as it allows for users to continue playing to beat their highest score.

Finally, if the user signed into the wrong account or wants to switch who Player 2 is logged in as, there is an option to log out. This button takes the user back to the original Home screen where they are prompted to login or register.



Leaderboard page

The leaderboard screen looks like this. It allows the user to view all the information they need at once. There is also a “Go Back” button to allow the user to return to the Game page.

How to Play

The game has a deck of 30 unique cards.
There are 3 card colours (red, black or yellow)
and each card has a number (1, 2, 3, 4, 5, 6, 7, 8, 9, 10).
The 30 cards are then shuffled and stored in the deck.

The rules:

Player 1 takes the top card from the deck
and Player 2 takes the next card.

If both players have the same card colour:
The player with the highest number wins.

If both players have different card colours, the winning colour is shown below:

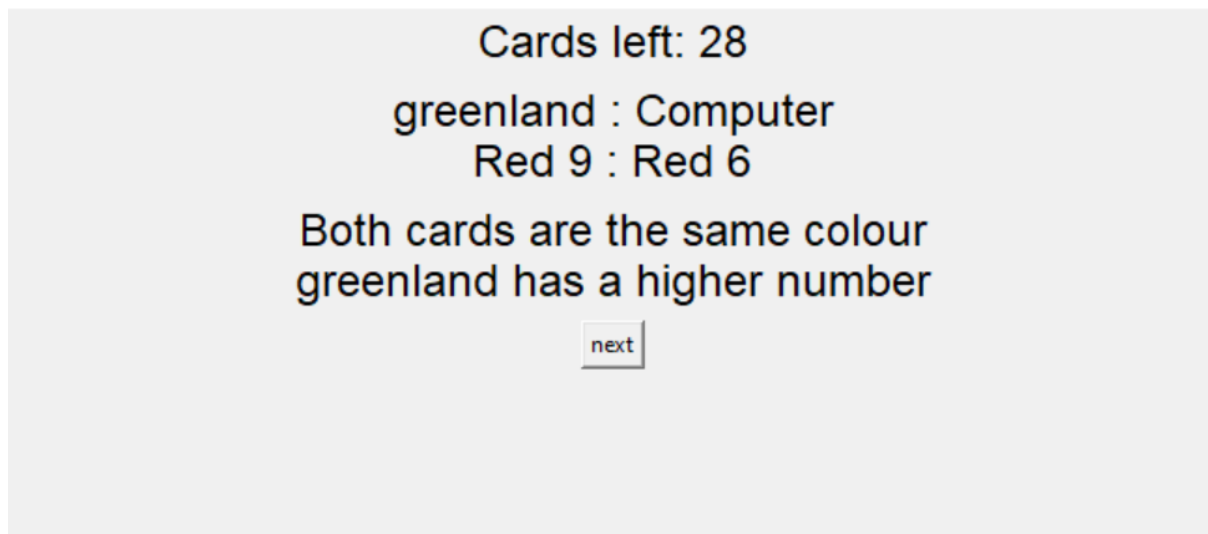
RED beats BLACK
YELLOW beats RED
BLACK beats YELLOW

The winner of each round keeps both players cards
and the players keep playing until there are no more cards left

Go Back

How to Play page

This page shows the How to Play page and teaches the user the rules of the game. I have chosen to follow the rules of the game very strictly and have not strayed from its original idea. There is also a “Go Back” button to allow the user to go back to the Game page.



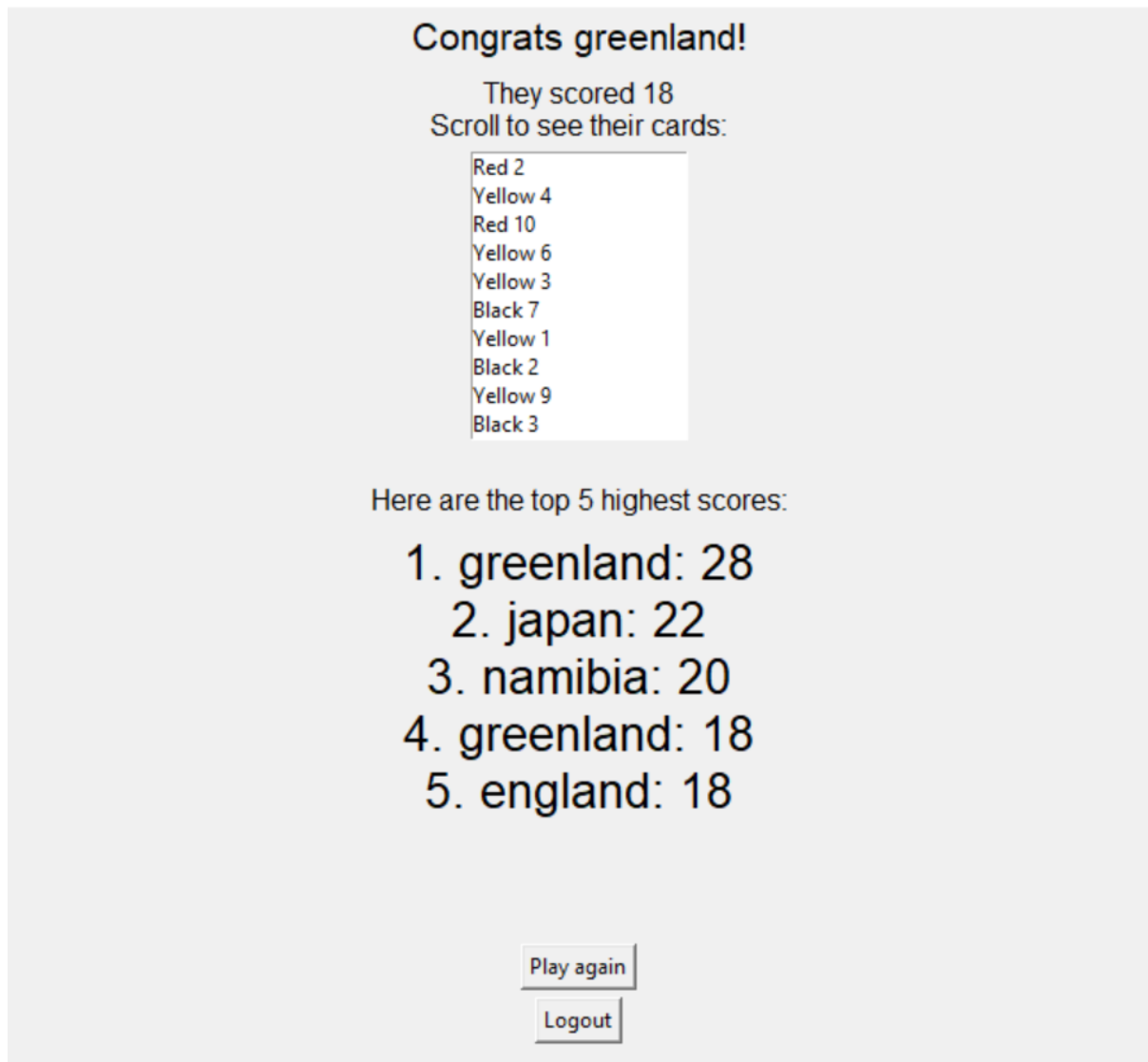
Gameplay page

This is the actual gameplay page. I have chosen to show lots of information on this page. The button “Next” at the bottom of the page updates this page and is used as the way to progress through the game.

At the top of the page I have included the number of cards left in the deck. This means the user can keep track of how many rounds have been and how many rounds are left.

Just below, I have the player’s name (in this example I am playing against the Computer) as well as what card they currently are holding.

I also wanted to have text that explains the gameplay and to describe what is happening in the game, as well as who won each round. This makes it clear immediately to the user who won that round and also allows for individuals who may not know the rules as well to still follow along.



End Screen page

This is the final main page of my program. Once the user has finished the game they are immediately taken to this end screen. This screen shows who won, the number of cards they won and a list of the cards they won with.

I have also decided to update and display the leaderboard at this stage. If they scored high enough, then their score would immediately be shown.

At the bottom there are two buttons, "Play Again" will take the user back to the Game page without logging anyone out. This means that a second round can immediately be played straight away after the last. There is also a "Logout" button that will sign out of any accounts that are currently logged in. It then returns the player to the original home page as you must be signed in to be authorised to play the game.

Functions and Classes Pseudocode

Functions (Database and Game focused)

This function initialises a game by creating a deck of cards, shuffling them, and dealing 30 random cards to two players.

```
def initialise_game():
```

```
    # Create a list of cards with each colour and value combination
```

```
    cards = []
```

```
    for colour in ["Red", "Black", "Yellow"]:
```

```
        for value in range(1, 11):
```

```
            cards.append([colour, value])
```

```
    # Shuffle the deck of cards to be in a random order
```

```
    random.shuffle(cards)
```

```
    # Create empty hands for two players
```

```
    global player_1
```

```
    global player_2
```

```
    player_1 = []
```

```
    player_2 = []
```

```
    # Deal 30 random cards to the two players
```

```
    return random.sample(cards,30)
```

This function deals the first card from the shuffled deck to each player and removes those cards from the deck.

```
def deal_cards():
```

```
    # Give the first card on top of the shuffled deck to player 1 and remove it from the deck.
```

```
    player_1.insert(0, shuffled_cards[0])
```

```
    shuffled_cards.pop(0)
```

```
    # Give the first card on top of the shuffled deck to player 2 and remove it from the deck.
```

```
    player_2.insert(0, shuffled_cards[0])
```

```
    shuffled_cards.pop(0)
```

```
# This function calculates the winner of the round based on the rules of the game.
```

```
def calculate_winner():
```

```
    # Initialise variables to False by default
```

```
    same_colour = False
```

```
    player_1_higher_number = False
```

```
    # Check if the cards have the same colour
```

```
    if player_1[0][0] == player_2[0][0]:
```

```
        same_colour = True
```

```
    # Check if player 1's number is higher than player 2's number
```

```
    if player_1[0][1] > player_2[0][1]:
```

```
        player_1_higher_number = True
```

```
        player_1_winner = True
```

```
    # If player 1's number is not higher than player 2's number,
```

```
    # then player 2's number must be lower than player 1's number.
```

```
    else:
```

```
        player_1_higher_number = False
```

```
        player_1_winner = False
```

```
    # If the cards have different colours, the winner colour is chosen.
```

```
    # Red beats Black, Black beats Yellow, and Yellow beats Red.
```

```
    if (player_1[0][0] == "Red"):
```

```
        if (player_2[0][0] == "Black"):
```

```
            player_1_winner = True
```

```
        else:
```

```
            player_1_winner = False
```

```
    if (player_1[0][0] == "Black"):
```

```
        if (player_2[0][0] == "Yellow"):
```

```
            player_1_winner = True
```

```
        else:
```

```
            player_1_winner = False
```

```
    if (player_1[0][0] == "Yellow"):
```

```
        if (player_2[0][0] == "Red"):
```

```
            player_1_winner = True
```

```
        else:
```

```
            player_1_winner = False
```

```
    # Return a tuple with the results of the round
```

```
    return same_colour, player_1_higher_number, player_1_winner
```

```
# This function takes a row as an input and returns the second item in the row.
```

```
def get_second_column(row):  
    # Return the second item in the row  
    return row[1]
```

```
# This function reads the csv file named "user profile database.csv" and sorts its rows  
based on the second column. The sorted rows are then written back to the same csv file.
```

```
import csv
```

```
def database_sort():  
    # read the csv file into a list of rows  
    with open("user profile database.csv", "r") as csvfile:  
        rows = list(csv.reader(csvfile, delimiter="|"))  
  
    # sorts the rows based on the second column  
    sorted_rows = sorted(rows[1:], key=get_second_column, reverse=True)  
    # possible alternate solution:  
    # sorted_rows = sorted(rows[1:], key=lambda x: x[1], reverse=True)  
  
    # write the sorted rows back to the csv file  
    with open("user profile database.csv", "w", newline="") as csvfile:  
        writer = csv.writer(csvfile, delimiter="|")  
        writer.writerows([rows[0]] + sorted_rows)
```

```
# This function takes a password as input and encrypts it using a key stored in the creds  
module. The encryption is done using XOR bitwise operation on each character of the  
password and the key.
```

```
import creds
```

```
def encrypt_decrypt(password):  
    # Remove any trailing newline character from the password  
    password = password.rstrip('\n')  
    result = ""  
    for i in range(len(password)):  
        # XOR operation on each character of the password and the key
```



```
result += chr(ord(password[i]) ^ ord(creds.key[i]))
return result
```

This function checks if a given username exists in the authentication database. It returns True if the username is found, False otherwise.

```
import csv
```

```
def check_username_exists(username):
```

```
    # Open the authentication database file
```

```
    with open("authentication database.csv", "r") as csvfile:
```

```
        # Use csv.reader to read the file
```

```
        reader = csv.reader(csvfile, delimiter=",")
```

```
        # Loop through each row in the file
```

```
        for row in reader:
```

```
            # Check if the lowercase version of the first column (username) matches the lowercase input
```

```
            if (row[0]).lower() == username.lower():
```

```
                # Return True if the username is found
```

```
                return True
```

```
    # Return False if the username is not found
```

```
    return False
```

#This function inserts a row of data into a CSV file, sorts the data based on the third column, and overwrites the original file with the sorted data.

```
def insert_into_database(data):
```

```
    with open("user profile database.csv", "a", newline="") as csvfile:
```

```
        writer = csv.writer(csvfile, delimiter=",")
```

```
        writer.writerow(data)
```

```
    with open("user profile database.csv", 'r') as f:
```

```
        lines = f.readlines()
```

```
    rows = [line.strip().split(',') for line in lines]
```

```
    sorted_rows = sorted(rows[1:], key=lambda row: row[2], reverse=True)
```

```
    sorted_lines = [rows[0]] + sorted_rows
```

```
    sorted_contents = [','.join(row) + '\n' for row in sorted_lines]
```

```
    with open("user profile database.csv", 'w') as f:
```

```
f.writelines(sorted_contents)
```

Classes (GUI focussed)

The following are part of the class Window(tk.Tk):

```
def __init__(self):
    # Initialises the Tkinter window
    tk.Tk.__init__(self)

    # Sets the title and size of the window
    self.title("Card Game")
    self.geometry("650x600")

    # Creates and displays the title and subtitle on the home screen
    self.home_title_text = tk.Label(self, text="Card Game Program", font=("Arial", 18))
    self.home_title_text.pack(pady=2)
    self.home_subtitle_text = tk.Label(self, text="Please Login / Register to play",
font=("Arial", 12))
    self.home_subtitle_text.pack(pady=2)

    # Creates and displays the login and registration buttons on the home screen
    self.home_login_button = tk.Button(self, text="Login",
command=self.show_login_page)
    self.home_login_button.pack(pady=2)
    self.home_register_button = tk.Button(self, text="Register",
command=self.show_register_page)
    self.home_register_button.pack(pady=2)

    # Creates an empty error label and an empty space label
    self.error_label = tk.Label(self, text="error", font=("Arial", 12))
    self.space_label = tk.Label(self, text="")
```

```
# This function hides all the child widgets of the current widget.
def hide(self):
    # Loops through all the child widgets of the current widget
```

```
for widget in self.winfo_children():
    # Forgets (removes) the packing configuration of each widget
    widget.pack_forget()
```

```
# This function displays the home page of the card game program.
def show_home_page(self):
    # Calls the hide() function to hide all the child widgets of the current widget
    self.hide()

    # Accesses the global variable player_1_logged_in
    global player_1_logged_in

    # Displays the home screen with appropriate buttons and labels
    self.home_title_text.pack(pady=2)
    self.home_subtitle_text.pack(pady=2)
    self.home_login_button.pack(pady=2)

    # Disables the register button if the player is already logged in
    if player_1_logged_in == True:
        self.home_register_button.config(state="disabled")
    else:
        self.home_register_button.config(state="normal")

    self.home_register_button.pack(pady=2)
```

```
# This function displays the login page of the card game program.
def show_login_page(self):
    # Calls the hide() function to hide all the child widgets of the current widget
    self.hide()

    # Displays the login screen with appropriate labels, entries, and buttons
    self.login_title_text = tk.Label(self, text="Please Login", font=("Arial", 18))
    self.login_title_text.pack(pady=2)

    self.login_username_label = tk.Label(self, text="Username:")
    self.login_username_label.pack(pady=2)
    self.login_username_entry = tk.Entry(self)
    self.login_username_entry.pack(pady=2)
```

```

self.login_password_label = tk.Label(self, text="Password:")
self.login_password_label.pack(pady=2)
self.login_password_entry = tk.Entry(self, show="*")
self.login_password_entry.pack(pady=2)

self.login_submit_button = tk.Button(self, text="Login", command=self.submit_login)
self.login_submit_button.pack(pady=2)

self.space_label.pack(pady=5)

self.go_back_button = tk.Button(self, text="Go Back",
command=self.show_home_page)
self.go_back_button.pack(pady=2)

```

#This function is responsible for submitting the user's login information and checking if the credentials are correct. If the username and password match the records in the authentication database, the player's name is recorded and they are logged in. The function then calls the show_game_page() function. If the credentials do not match, an error message is displayed.

```

def submit_login(self):
    global player_1_logged_in
    global player_2_logged_in
    global player_1_name
    global player_2_name

    # Get the username and password entered by the user
    username = self.login_username_entry.get()
    password = self.login_password_entry.get()

    # Check if the account is already logged in
    if username == player_1_name:
        self.error_label.config(text="Account already logged in !")
        self.error_label.pack(pady=2)
        return

    # Encrypt the password and convert it to hex format
    password = encrypt_decrypt(password).encode().hex()

    # Open the authentication database and check if the username and password match
    any records
    with open("authentication database.csv", newline="") as csvfile:
        reader = csv.reader(csvfile, delimiter=",")

```

```

next(reader) # skip the first row
username_found = False
for row in reader:
    if (row[0] == username) and (row[1] == password):
        username_found = True

    # If player 1 is already logged in, log in player 2 instead
    if player_1_logged_in == True:
        player_2_logged_in = True
        player_2_name = username

    # If player 1 is not logged in, log them in
    if player_1_logged_in == False:
        player_1_logged_in = True
        player_1_name = username

    # Call the show_game_page() function
    self.show_game_page()

# If the username and password do not match any records, display an error
message
if username_found == False:
    self.error_label.config(text="Sorry, incorrect credentials !")
    self.error_label.pack(pady=2)

```

```

#This function displays the register page of the card game program

def show_register_page(self):
    self.hide() # hides all widgets from the current page

    # create and display the title for the register page
    self.register_title_text = tk.Label(self, text="Please Register", font=("Arial", 18))
    self.register_title_text.pack(pady=2)

    # create and display the label and entry box for the username
    self.register_username_label = tk.Label(self, text="Username:")
    self.register_username_label.pack(pady=2)
    self.register_username_entry = tk.Entry(self)
    self.register_username_entry.pack(pady=2)

    # create and display the label and entry box for the password
    self.register_password_label = tk.Label(self, text="Password:")
    self.register_password_label.pack(pady=2)
    self.register_password_entry = tk.Entry(self, show="*")

```

```

self.register_password_entry.pack(pady=2)

# create and display the register button
self.register_submit_button = tk.Button(self, text="Register",
command=self.submit_register)
self.register_submit_button.pack(pady=2)

# create and display an empty label for spacing purposes
self.space_label = tk.Label(self, text="")
self.space_label.pack(pady=5)

# create and display the "Go Back" button
self.go_back_button = tk.Button(self, text="Go Back",
command=self.show_home_page)
self.go_back_button.pack(pady=2)

```

```

def submit_register(self):
    #Function to handle registration form submission

    # Get the username entered by the user
    username = self.register_username_entry.get()

    # Check if the username contains only alphanumeric characters
    # If not, display an error message and return
    if not username.isalnum():
        self.error_label.config(text="Please only use A-Z and 0-9 in your username !")
        self.error_label.pack(pady=2)
        return

    # Check if the username already exists in the database
    # If yes, display an error message and return
    elif check_username_exists(username):
        self.error_label.config(text="Sorry, this username already exists !")
        self.error_label.pack(pady=2)
        return

    # Check if the length of the username is between 3 and 24 characters
    # If not, display an error message and return
    elif (len(username) < 3) or (len(username) > 24):
        self.error_label.config(text="Username must be 3-24 characters long !")
        self.error_label.pack(pady=2)
        return

    # If all checks pass, convert the username to lowercase
    else:

```

```

        username.lower()

    # Get the password entered by the user
    password = self.register_password_entry.get()

    # Check if the length of the password is at least 4 characters
    # If not, display an error message and return
    if (len(password) < 4):
        self.error_label.config(text="Password is too short !\nPlease make it more than 3
characters long")
        self.error_label.pack(pady=2)
        return
    # Check if the length of the password is less than 30 characters
    # If not, display an error message and return
    elif (len(password) > 30):
        self.error_label.config(text="Password is too long !\nPlease make it less than 30
characters long")
        self.error_label.pack(pady=2)
        return

    # Encrypt the password using a simple encryption function and encode it as
    hexadecimal
    password = encrypt_decrypt(password).encode().hex()

    # Create a list of the username and encrypted password
    data = [username,password]

    # Write the data to a CSV file for authentication
    with open("authentication database.csv", "a", newline="") as csvfile:
        writer = csv.writer(csvfile, delimiter="|")
        writer.writerow(data)

    # Show the game page
    self.show_game_page()

```

#This function displays the game page of the card game program

```

def show_game_page(self):
    global player_2_logged_in

    # Hide current page
    self.hide()

    # Create a title label

```

```

self.game_title_text = tk.Label(self, text="Welcome to the Card Game
!",font=("Arial",18))
self.game_title_text.pack(pady=2)

# Button to start playing the game
self.game_gameplay_button = tk.Button(self,text="Play
Game",command=self.show_gameplay_page)
self.game_gameplay_button.pack(pady=2)

# Button to add a second player, displayed only if player 2 is not logged in
if player_2_logged_in == False:
    self.game_addplayer_button = tk.Button(self,text="Add Player
2",command=self.show_home_page)
    self.game_addplayer_button.pack(pady=2)
# If player 2 is logged in, the add player button is not displayed
elif player_2_logged_in == True:
    self.game_addplayer_button.forget()

# Button to learn how to play
self.game_howtoplay_button = tk.Button(self,text="How to
Play",command=self.show_howtoplay_page)
self.game_howtoplay_button.pack(pady=2)

# Button to see the leaderboard
self.game_leaderboard_button =
tk.Button(self,text="Leaderboard",command=self.show_leaderboard_page)
self.game_leaderboard_button.pack(pady=2)

# Create some space between the buttons and the logout button
self.space_label.pack(pady=5)

# Button to logout
self.game_logout_button = tk.Button(self,text="Logout",command=self.logout)
self.game_logout_button.pack(pady=2)

```

#This function logs all current users logged in and returns them to the original Login page

```

def logout(self):
    global player_1_logged_in
    global player_2_logged_in
    global player_1_name
    global player_2_name

```



```
# Reset global variables
player_1_logged_in = False
player_2_logged_in = False
player_1_name = ""
player_2_name = "Computer"
```

```
# Show the home page
self.show_home_page()
```

```
#This function shows the gameplay page for the user to play the card game
```

```
def show_gameplay_page(self):
    # Hide current page
    self.hide()

    # Display cards left label
    self.gameplay_cardsleft_text = tk.Label(self, text="Cards left: 30",font=("Arial",18))
    self.gameplay_cardsleft_text.pack(pady=2)

    # Display player's cards label
    self.gameplay_playerscards_text = tk.Label(self, text="",font=("Arial",18))
    self.gameplay_playerscards_text.pack(pady=2)

    # Display game play description label
    self.gameplay_description_text = tk.Label(self, text="Press Next",font=("Arial",18))
    self.gameplay_description_text.pack(pady=2)

    # Display update button to advance to next card
    self.gameplay_update_button = tk.Button(self, text="next",
command=self.gameplay_update)
    self.gameplay_update_button.pack(pady=2)

    # Initialise game and shuffle the deck of cards
    global shuffled_cards
    shuffled_cards = initialise_game()

    # Call gameplay_update function to display the first card
    self.gameplay_update()
```

```
# This variable updates the gameplay page. It draws cards and calculates the winner and then displays it so the user knows what is happening.
```

```
def gameplay_update(self):
```

```
    global shuffled_cards
    global player_1
    global player_2
    global winner
    global winning_cards
```

```
# while the shuffled cards are not empty (while it hasn't ended)
```

```
# check to see if player 2 is computer
```

```
    if len(shuffled_cards) == 0:
        if len(player_1) > len(player_2):
            winner = player_1_name
            winning_cards = player_1
        else:
            if player_2_name != "Computer":
                winner = player_2_name
                winning_cards = player_2
            else:
                winner = "Computer"
                winning_cards = player_2
    self.show_end_page()
    return
```

```
deal_cards()
```

```
same_colour, player_1_higher_number, player_1_winner = calculate_winner()
```

```
description = ""
```

```
if same_colour == True:
```

```
    description += "Both cards are the same colour\n"
```

```
    if player_1_higher_number == True:
```

```
        description += f"{player_1_name} has a higher number"
```

```
    else:
```

```
        description += f"{player_2_name} has a higher number"
```

```
else:
```

```
    if player_1_winner == True:
```

```
        description += f"{player_1_name} is the winner\n"
```

```
    else:
```

```
        description += f"{player_2_name} is the winner\n"
```

```
# updates how many cards are left in the deck
```

```
cards_left = "Cards left: " + str(len(shuffled_cards))
```

```
if len(shuffled_cards) == 30:
```

```

        playerscards = f"{player_1_name} : {player_2_name}\n"
    else:
        playerscards = f"{player_1_name} : {player_2_name}\n{player_1[0][0]}
{player_1[0][1]} : {player_2[0][0]} {player_2[0][1]}"

# updates the description of the round
self.gameplay_description_text.config(text=description)
self.gameplay_cardsleft_text.config(text=cards_left)
self.gameplay_playerscards_text.config(text=playerscards)

if player_1_winner == True:
    player_1.insert(0,player_2[0])
    player_2.pop(0)

else:
    player_2.insert(0,player_1[0])
    player_1.pop(0)

```

#This function shows the end page to show the game stats, including the winner and their cards

```

def show_end_page(self):
    # hide the current page
    self.hide()

    # get global variables
    global winner
    global winning_cards

    # create a label showing the winner
    self.end_gameplay_description = tk.Label(self, text=f"Congrats {winner}!",
font=("Arial",16))
    self.end_gameplay_description.pack(pady=2)

    # create a label showing the winner's score and list of cards won
    self.leaderboard_score_text = tk.Label(self, text=f"They scored
{len(winning_cards)}\nScroll to see their cards:", font=("Arial",12))
    self.leaderboard_score_text.pack(pady=2)

    # create a listbox showing the winner's cards
    listbox = tk.Listbox(self, width=20, height=10)
    for i in winning_cards:
        listbox.insert(tk.END, i)
    listbox.pack()

```

```

# create a label showing the top 5 highest scores
self.leaderboard_subtitle_text = tk.Label(self, text="\nHere are the top 5 highest
scores:", font=("Arial",12))
self.leaderboard_subtitle_text.pack(pady=2)

# if the winner is not the computer, insert their details into the leaderboard database and
sort it
if winner != "Computer":
    insert_into_database([winner, len(winning_cards), winning_cards])
    database_sort()

# open the leaderboard database file, read the contents and store in a list
with open("user profile database.csv", newline="") as csvfile:
    reader = csv.reader(csvfile, delimiter=",")
    next(reader) # skip the first row
    lines = []
    for row in reader:
        lines.append(row)

# create strings for the top 5 highest scores with the format "rank. name: score"
rank1 = "1. " + str(lines[0][0]) + ": " + str(lines[0][1]) + "\n"
rank2 = "2. " + str(lines[1][0]) + ": " + str(lines[1][1]) + "\n"
rank3 = "3. " + str(lines[2][0]) + ": " + str(lines[2][1]) + "\n"
rank4 = "4. " + str(lines[3][0]) + ": " + str(lines[3][1]) + "\n"
rank5 = "5. " + str(lines[4][0]) + ": " + str(lines[4][1]) + "\n"

# concatenate the strings for the top 5 highest scores into one string
leaderboard_ranks = rank1 + rank2 + rank3 + rank4 + rank5

# create a label showing the top 5 highest scores
self.leaderboard_list_text = tk.Label(self, text=leaderboard_ranks, font=("Arial", 20))
self.leaderboard_list_text.pack(pady=2)

# create space between leaderboard and buttons
self.space_label.pack(pady=5)

# create a button to show the game page again
self.show_game_page_button = tk.Button(self, text="Play again",
command=self.show_game_page)
self.show_game_page_button.pack(pady=2)

# create a button to logout
self.game_logout_button = tk.Button(self, text="Logout", command=self.logout)
self.game_logout_button.pack(pady=2)

```

```

#This function shows the page that explains to the user how to play

def show_howtoplay_page(self):
    # Hide the current page
    self.hide()

    # Create the title text for the how-to-play page
    self.howtoplay_title_text = tk.Label(self, text="How to Play",font=("Arial",18))
    self.howtoplay_title_text.pack(pady=2)

    # Create the subtitle text for the how-to-play page
    self.howtoplay_subtitle_text = tk.Label(self, text="The game has a deck of 30 unique
cards.\nThere are 3 card colours (red, black or yellow)\nand each card has a number (1,
2, 3, 4, 5, 6, 7, 8, 9, 10).\nThe 30 cards are then shuffled and stored in the deck.\n\n\nThe
rules:\nPlayer 1 takes the top card from the deck\nand Player 2 takes the next card.\n\nIf
both players have the same card colour:\nThe player with the highest number wins.\n\nIf
both players have different card colours, the winning colour is shown below:\nRED beats
BLACK\nYELLOW beats RED\nBLACK beats YELLOW\n\n\nThe winner of each round
keeps both players cards\nand the players keep playing until there are no more cards
left",font=("Arial",12))
    self.howtoplay_subtitle_text.pack(pady=2)

    # Create a label for spacing
    self.space_label.pack(pady=5)

    # Create a button to go back to the game page
    self.go_back_to_game_button = tk.Button(self, text="Go Back",
command=self.show_game_page)
    self.go_back_to_game_button.pack(pady=2)

```

```

#This function shows the leaderboard page to display the current top 5 best scores

def show_leaderboard_page(self):
    # Hide the current page
    self.hide()

    # Add the leaderboard title
    self.leaderboard_title_text = tk.Label(self, text="Leaderboard",font=("Arial",18))
    self.leaderboard_title_text.pack(pady=2)

    # Add the leaderboard subtitle

```

```

self.leaderboard_subtitle_text = tk.Label(self, text="Here are the top 5 highest
scores:",font=("Arial",12))
self.leaderboard_subtitle_text.pack(pady=2)

# Open the user profile database file and read in the data
with open("user profile database.csv", newline="") as csvfile:
    reader = csv.reader(csvfile, delimiter=",")
    next(reader) # skip the first row
    lines = []
    for row in reader:
        lines.append(row)

# Get the top 5 highest scores and format them as strings
rank1 = "1. " + str(lines[0][0]) + ": " + str(lines[0][1]) + "\n"
rank2 = "2. " + str(lines[1][0]) + ": " + str(lines[1][1]) + "\n"
rank3 = "3. " + str(lines[2][0]) + ": " + str(lines[2][1]) + "\n"
rank4 = "4. " + str(lines[3][0]) + ": " + str(lines[3][1]) + "\n"
rank5 = "5. " + str(lines[4][0]) + ": " + str(lines[4][1]) + "\n"

# Combine the top 5 scores into one string
leaderboard_ranks = rank1 + rank2 + rank3 + rank4 + rank5

# Add the leaderboard string to the page
self.leaderboard_list_text = tk.Label(self, text=leaderboard_ranks, font=("Arial", 20))
self.leaderboard_list_text.pack(pady=2)

# Add some space between the leaderboard and the "go back" button
self.space_label.pack(pady=5)

# Add the "go back" button
self.go_back_to_game_button = tk.Button(self, text="Go Back",
command=self.show_game_page)
self.go_back_to_game_button.pack(pady=2)

```

Additionally, here is the final part of the code:

```

player_1_logged_in = False
player_2_logged_in = False

player_1_name = ""
player_2_name = "Computer"

```

```
window = Window()
window.mainloop()
```

Testing

Test plan

Firstly, I will test the visual appearance. By checking to make sure everything is spaced out correctly and in the right place will mean it is more consistent throughout. I will then read all the text on the game to make sure there are no spelling errors and the correct boxes / buttons lead to the correct place.

I will then go on to run debugging on the source code to check for any errors. I will be using black box testing to find any bugs or problems with my code.

Black box testing ensures the functions and features of the program are working correctly. By working out expected output from an inputted data set, if the output from the program is as expected, black box testing is completed

I only have two places with user input (excluding clicking boxes), the Login page and the Register page. Both these pages use this data inputted from the user in a form to access the database so I need to make sure that this works currently without any issues.

Login Page:

The Login Page takes two inputs, the username and password.

Test	Description	Data Type	Expected	Error Message
------	-------------	-----------	----------	---------------

Number			Result	Displayed
#1	What happens when both the username and password are empty?	Empty string / Empty	No access given	"Account already logged in !"
#2	Input username but wrong password.	String	No access given	"Sorry, Incorrect credentials !"
#3	Username and password with non-alphanumeric characters	String	No access given	"Sorry, Incorrect credentials !"

There is not a lot I can do to test my login as it is simply checking if the username is found in the database. My code validates strings to be able to not cause any errors in the program.

However, from this test plan I have found out that when the username is empty, the error message "already logged in !" is displayed, which is unusual but luckily not an important bug.

Registration Page:

The Registration Page takes two inputs, the username and password. There is much more room for error as instead of just looking up values in a database, it opens the database and writes to it.

Test Number	Description	Data Type	Expected Result	Error Message Displayed
#1	What happens when both the username and password are empty?	Empty / Empty String	No account created	"Please only use A-Z and 0-9 in your username !"
#2	What happens when the username is empty but there is a password present?	Empty / Empty String	No account created	"Please only use A-Z and 0-9 in your username !"
#3	What happens when there is a username but no password?	Empty / Empty String	No account created	"Password is too short. Please make it more than 3 characters long"
#4	What happens when the username is less than 3 characters	Integer not in range	No account created	"Password is too short. Please make it

	long?			more than 3 characters long"
#5	What happens when the username is more than 30 characters long?	Integer not in range	No account created	"Password is too long. Please make it less than 30 characters long"
#6	What happens when the username is 12 characters long?	Integer in range	Account created	N/A
#7	What happens when the username is already in use but the password is valid?	String	No account created	"Sorry, this username is already in use !"
#8	What happens when the password is already used by another account and the username is valid?	String	Account created	N/A
#9	What happens when the username contains non-alphanumeric characters but the password is valid?	String	No account created	"Please only use A-Z and 0-9 in your username !"
#10	What happens when the password contains non-alphanumeric characters but the username is valid?	String	No account created	N/A

I decided to allow non-alphanumeric characters in passwords to allow for symbols like ! or @ to improve account security.

Procedure and Classes listing

FUNCTION NAME Function description			
Procedure / Function	Description	Parameters	Output / Return Value
initialise_game	This function initialises a game by creating a deck of cards and shuffling them for use.	None	Return shuffled deck
deal_cards	This function deals the first card from the shuffled deck to each player and removes those cards from the deck.	None	None
calculate_winner	This function calculates the winner of the round based on the rules of the game.	None	Return same_colour, player_1_higher_number, player_1_winner
get_second_column	This function takes a row as an input and returns the second item in the row.	row	row[1]
database_sort	This function reads the csv file named "user profile database.csv" and sorts its rows based on the second column. The sorted rows are then written back to the same csv file.	None	None
encrypt_decrypt	This function takes a password as input and encrypts it using a key stored in the creds module. The encryption is done using XOR bitwise operation on each character of the password and the key.	password	Return result

check_username_exists	This function checks if a given username exists in the authentication database. It returns True if the username is found, False otherwise.	username	Return True or Return False
insert_into_database	This function inserts a row of data into a CSV file, sorts the data based on the third column, and overwrites the original file with the sorted data.	data	None
Window.__init__	Initialises the Tkinter window, creates the home landing page for the program.	self	None
Window.hide	Hides all the child widgets of the current widget	self	None
Window.show_home_page	Displays the home page of the card game program	self	None
Window.show_login_page	This function displays the login page of the card game program	self	None
Window,submit_login	This function is responsible for submitting the user's login information and checking if the credentials are correct.	self	None
Window.show_register_page	This function displays the register page of the card game program	self	None
Window.submit_register	This function handles the registration form submission	self	return
Window.show_game_page	This function displays the game page of the card game program	self	None

Window.logout	This function logs all current users logged in and returns them to the original Login page	self	None
Window.show_gameplay_page	This function shows the gameplay page for the user to play the card game	self	None
Window.gameplay_update	This updates the gameplay page. It draws cards and calculates the winner and then displays it so the user knows what is happening	self	return
Window.show_end_page	This function shows the end page to show the game stats, including the winner and their cards	self	None
Window.show_howto_play_page	This function shows the page that explains to the user how to play	self	None
Window.show_leaderboard_page	This function shows the leaderboard page to display the current top 5 best scores	self	None

Important Variables listing

VARIABLE NAME Variable description		
Variable	Description	Data Type
player_1_logged_in	Holds information whether player 1 is logged in	boolean
player_2_logged_in	Holds information whether player 2 is logged in	boolean
player_1_name	Holds the username player 1 logged in with	string
player_2_name	Holds the username player 2 logged in with	string
player_1	Holds player 1's entire deck of cards	array

player_2	Holds player 2's entire deck of cards	array
winner	Stores the winner to be used on the end screen	string
winning_cards	Stores the winner's cards to be used on the end screen	array
shuffled_cards	Stores the shuffled cards in a random order every time	array
username	Input from the user about their username, used to assign player_1_name and player_2_name	string
password	Input from the user about their password, used to authenticate the user	string
row	Used in .CSV files and refers to the entire row	string
data	Used to hold data temporarily before it is stored into the .CSV	array
same_colour	Holds the value if both cards are the same colour	boolean
player_1_higher_number	Holds the value if player 1 has a larger number than player 2	boolean
player_1_winner	Holds the value if player 2 is the winner	boolean

How does the system meet the objectives

The game meets the basic objectives such as multiplayer, storage of player details and a leaderboard.

Improvements & Extensions to the game

There are no administrative accounts and no admin commands to edit the database. Additionally, the major criticism I have is that the graphics of the game are dated. I have not included any visual graphics or animations. Finally, I regret not including a user search to see someone's best scores or a way to change your password or username.

The implementation of any of these extensions will create a more professional program and increase functionality and usability.

Evaluation

I believe the basic functionality of the game was achieved. Leaving room for extension and improvement in the near future. If I were to restart I think I would've built my program with more thoughtfulness into how I could add new ideas.