S.A. Nethini Pabodhya Perera

Report on Python Final Coursework

The following report contains the analysis of the following python coursework containing an application for a world rally cross championship management.

Table of Content

```
# K.P.Seniru Sonal Pthirana
# Final Coursework

import random

class colors: # Class to import colors and fonts to the program
    RED = '\033[31m'
    RESET = '\033[0m'
    YELLOW = '\033[33m'
    BLUE = '\033[34m'
    GREEN = "\033[0;32m"
    CYAN = "\033[0;36m"
    BOLD = "\033[1m"
    ITALIC = "\033[3m"


def add_driver(details): # Function to add driver details to the database
    name = input("Enter driver's name: ")
    age = int(input("Enter driver's age: "))
    team = input("Enter driver's team: ")
    car = input("Enter driver's car: ")
    points = int(input("Enter the driver's points: "))
    # Using Dictionaries as the database to store driver's details
    details[name] = {'Age': age, 'Team': team, 'Car': car, 'Points':
points}
    print(colors.YELLOW + "Driver details added successfully!" +
colors.RESET)

def delete_driver(details): # Function to delete a driver and driver's
detail from the databse
    name = input("Enter the name of the driver to delete: ")
    if name in details:
        del details[name]
        print(colors.YELLOW + "Driver details deleted successfully!" +
colors.RESET )
    else: # If the Driver is not on the database
        print(colors.RED + "Driver not found!" + colors.RESET)

def update_driver(details): # Function to update details of a driver
    name = input("Enter the name of the driver to update: ")
    if name in details:
        age = int(input("Enter driver's new age: "))
        team = input("Enter driver's new team: ")
        car = input("Enter driver's new car: ")
        details[name]['Age'] = age
        details[name]['Team'] = team
        details[name]['Car'] = car
        print(colors.YELLOW + "Driver details updated successfully!" +
colors.RESET)
    else: # If the Driver is not on the databse
        print(colors.RED + "Driver not found!" + colors.RESET)

def simulate_race(details): # Function to simulate a Race between the
drivers in the database
```

```
    race_date = input("Enter the date of the race (YYYY-MM-DD): ")
    race_location = input("Enter the location of the race: ")
    race_results = []

    for name, driver in details.items():
        # Randomly assigning a position to each player
        position = random.randint(1, len(details))
        points = 0
        if position == 1:
            points = 10
        elif position == 2:
            points = 7
        elif position == 3:
            points = 5

        race_results.append((name, position, points))
        details[name]['Points'] += points
    # Sort the drivers ny their postions
    race_results.sort(key=lambda x: x[1])

    with open('races.txt', 'a') as file: # Open the file races and append
        file.write(f"Race Date: {race_date}\n")
        file.write(f"Location: {race_location}\n")
        for result in race_results:
            file.write(f"Driver: {result[0]}, Position: {result[1]},
Points: {result[2]}\n")
        file.write("\n")

    print(colors.YELLOW + "Race simulated and results recorded!" +
colors.RESET)

def display_standings(details): # Function to view the rally crossing
table
    sorted_details = sorted(details.items(), key=lambda x: x[1]['Points'],
reverse=True)#sorting the drivers in descending order
    print(colors.CYAN + colors.BOLD + "\n--- Championship Standings ---")
    print(colors.BLUE + colors.BLUE + "{:<15} {:<10} {:<10} {:<10}
{:<10}".format("Name", "Age", "Team", "Car", "Points") +colors.RESET)
    for driver, info in sorted_details:
        print("{:<15} {:<10} {:<10} {:<10} {:<10}".format(driver,
info['Age'], info['Team'], info['Car'], info['Points']))

def view_race_table(): # Funtion to view the race table sorted according
to the date
    try:
        with open('races.txt', 'r') as file: # open the file races and
read
            print("\n--- Race Table ---")
            print(file.read())
    except FileNotFoundError:# If the file not found
        print(colors.RED + "No race data found." + colors.RESET)

def save_data(details): # Function to save the data to a file
    with open('drivers.txt', 'w') as file:# open the file drivers and
```

```
write
        for name, info in details.items():

file.write(f"{name},{info['Age']},{info['Team']},{info['Car']},{info['Poin
ts']}\n")
    print(colors.YELLOW + "Data saved successfully!" + colors.RESET)

def load_data(details): # Function to load the data from saved file
    try:
        with open('drivers.txt', 'r') as file: # open the file drivers and
read
            lines = file.readlines()
            for line in lines:
                line = line.strip()
                if line:
                    name, age, team, car, points = line.split(',')
                    details[name] = {'Age': age, 'Team': team, 'Car': car,
'Points': int(points)}
                    print(colors.YELLOW + "Data has been loaded from the
file" + colors.RESET)
    except FileNotFoundError:# If the file is not found
        print(colors.RED + "No previous data found." + colors.RESET)

# Main Menu
driver_details = {}

print(colors.CYAN + colors.ITALIC + colors.BOLD + "\n--- Rally Cross
Management System ---" + colors.RESET)
print(colors.BLUE + "\nType ADD for adding driver details")
print(colors.BLUE +"Type " + colors.GREEN + "DDD" +colors.BLUE + " for
deleting")
print(colors.BLUE +"Type " + colors.GREEN + "UDD" +colors.BLUE + " for
updating driver details" )
print(colors.BLUE +"Type " + colors.GREEN + "VCT" +colors.BLUE + " for
viewing the rally cross standings table")
print(colors.BLUE +"Type " + colors.GREEN + "SRR" +colors.BLUE + " for
simulating a random race")
print(colors.BLUE +"Type " + colors.GREEN + "VRL" +colors.BLUE + " for
viewing race table sorted according to the date")
print(colors.BLUE +"Type " + colors.GREEN + "STF" +colors.BLUE + " to save
the current data to a text file")
print(colors.BLUE +"Type " + colors.GREEN + "RFF" +colors.BLUE + " to load
data from the saved text file")
print(colors.BLUE +"Type " + colors.GREEN + "ESC" +colors.BLUE + " to exit
the program" + colors.RESET)


while True: # Looping the program until manually breaks

    choice = input("\nEnter your choice: ")

    if choice == 'ADD':
        add_driver(driver_details)
    elif choice == 'DDD':
        delete_driver(driver_details)
    elif choice == 'UDD':
        update_driver(driver_details)
    elif choice == 'VCT':
        display_standings(driver_details)
    elif choice == 'SRR':
        simulate_race(driver_details)
    elif choice == 'VRL':
        view_race_table()
    elif choice == 'STF':
        save_data(driver_details)
    elif choice == 'RFF':
        load_data(driver_details)
    elif choice == 'ESC':
        save_data(driver_details)
        print(colors.YELLOW + "Exiting the program.....")
        break
    else:
        print(colors.RED + colors.BOLD + "Invalid choice. Please try 
again." + colors.RESET)
```

This python program contains a command line application for a world rally cross championship management. This program enables the user to select an option from the following functions.

1. Adding driver details (ADD)
2. Deleting a driver (DDD)
3. Updating driver details (UDD)
4. Viewing the rally cross standing table (VCT)
5. Simulating a random race (SRR)
6. Viewing race table sorted according to the date (VRL)
7. Save the current data to a text file (STF)
8. Load data from the saved text file (RFF)
9. Exit the program (ESC)

*The user can enter the word given in the brackets next to the option to choose a function they want to do. If the user enters a wrong input, it will display “Invalid choice. Please try again “and the user can enter their choice again.

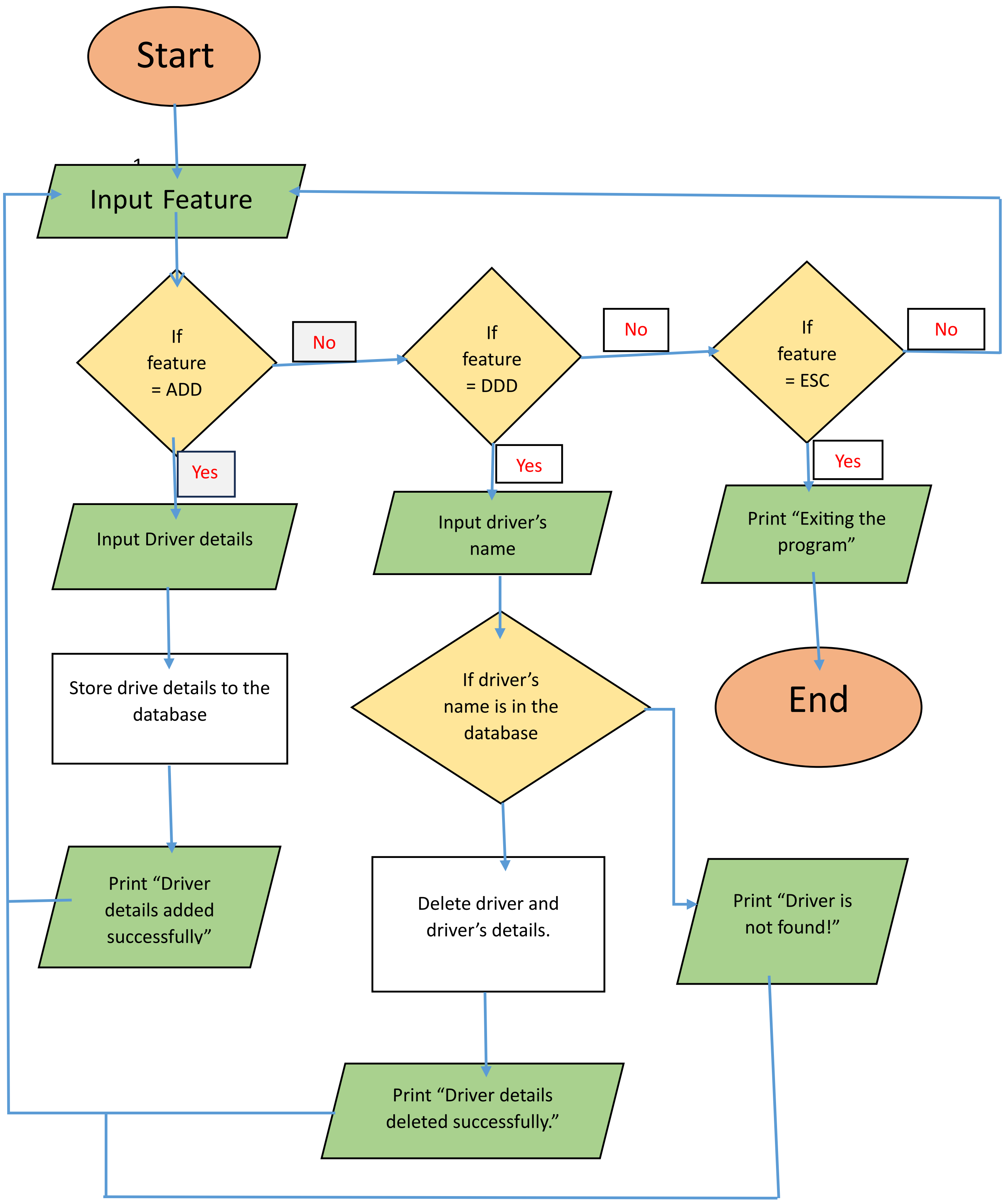
*In this program user can add a driver and the driver’s details to the database. The User can delete a driver by searching their name from the database. The User can also update details of a driver by searching their name.

*Through this program the user can view the championship standings, ordered by points in descending order. All the details regarding the drivers can be seen from this function.

*This program is also able to simulate a race among the drivers and assign points and a position to each driver in the database. After that the program is also able to display all the information about the races sorted according to the date.

*This program also has the function to save all the details of the drivers to a file. Then the system can also load the current data from the text file to enable resume capabilities.

Flow charts for function 1:ADD and 2:DDD


Start
Input Feature
If feature = ADD
No
If feature = DDD
No
If feature = ESC
No
Yes
Input Driver details
Yes
Input driver's name
Yes
Print "Exiting the program"
Store drive details to the database
If driver's name is in the database
End
Print "Driver details added successfully"
Delete driver and driver's details.
Print "Driver is not found!"
Print "Driver details deleted successfully."

1. Adding Driver details (ADD)

```
def add_driver(details): # Function to add driver details to the database
    name = input("Enter driver's name: ")
    try:
     age = input("Enter driver's age: ")
    except ValueError:
        print(colors.RED + "Integer required" + colors.RESET)
        age = int(input("Enter driver's age: "))
    team = input("Enter driver's team: ")
    car = input("Enter driver's car: ")
    try:
      points = int(input("Enter the driver's points: "))
    except ValueError:
         print(colors.RED + "Integer required" + colors.RESET)
         points = int(input("Enter the driver's points: "))
    #Using Dictionaries as the database to store driver's details
    details[name] = {'Age': age, 'Team': team, 'Car': car, 'Points':
points}
    print(colors.YELLOW + "Driver details added successfully!" +
colors.RESET)
```

This Function is used to adding driver details to the program database. From this function user can add driver details such as,

- Name
- Age
- Team
- Car
- Points

After the user inputs data, the program creates a dictionary for each driver according to their name. In this dictionary all the keys are assigned, and values are added to the dictionary when the user inputs the driver details. Age and Points are recorded as integers when the user inputs them. When points are recorded as integers, it is easier to add points to each driver according to their races. If the user doesn't input an integer as age and points ,the message, "**Integer Required**" will display in red color.

After the all the details are added, there will be an output printing **"Driver details added successfully!"** in yellow.

2.Deleting Driver details (DDD)

```
def delete_driver(details): # Function to delete a driver and driver's
detail from the databse
    name = input("Enter the name of the driver to delete: ")
    if name in details:
        del details[name]
        print(colors.YELLOW + "Driver details deleted successfully!" +
colors.RESET )
    else: # If the Driver is not on the database
        print(colors.RED + "Driver not found!" + colors.RESET)
```

This function is used for deleting a driver. The user can search for the driver by searching the driver's name through the database.

First the program searches for the driver in the database. If the driver is found, the program will delete that driver and the driver's detail from the database. After deleting the driver, the program would display an output saying, **"Driver details deleted successfully!"** in yellow color.

If the driver cannot be found in the database program will display, **"Driver not found"** in red color.

3.Updating Driver details (UDD)

```
def update_driver(details): # Function to update details of a driver
    name = input("Enter the name of the driver to update: ")
    if name in details:
        try:
          age = int(input("Enter driver's new age: "))
        except ValueError:
            print(colors.RED + "Integer required" + colors.RESET)
            age = int(input("Enter driver's new age: "))
        team = input("Enter driver's new team: ")
        car = input("Enter driver's new car: ")
        details[name]['Age'] = age
        details[name]['Team'] = team
        details[name]['Car'] = car
        print(colors.YELLOW + "Driver details updated successfully!" +
colors.RESET)
    else: # If the Driver is not on the database
        print(colors.RED + "Driver not found!" + colors.RESET)
```

This function allows the user to update the details of a driver. User can search the driver by searching their name. Then the program will allow the user to change the details of that driver.

After the user enters the name of the driver the program will search for the driver in the database. If the driver is found the program will let the user change the driver's age, team, and car. After the necessary details are updated, the program will print **"Driver details are updated successfully!"** in yellow color.

If the user doesn't input and integer as age the message, "**Integer Required"** will display in red color.

If the driver doesn't exist in the database, the program will print **"Driver not found "**in red color.

4.Viewing the rally cross table (VCT)

```
def display_standings(details): # Function to view the rally crossing
table
    sorted_details = sorted(details.items(), key=lambda x: x[1]['Points'],
reverse=True)#sorting the drivers in descending order
    print(colors.CYAN + colors.BOLD + "\n--- Championship Standings ---")
    print(colors.BLUE + colors.BLUE + "{:<15} {:<10} {:<10} {:<10}
{:<10}".format("Name", "Age", "Team", "Car", "Points") +colors.RESET)
    for driver, info in sorted_details:
        print("{:<15} {:<10} {:<10} {:<10} {:<10}".format(driver,
info['Age'], info['Team'], info['Car'], info['Points']))
```

This function sorts the driver by the points each driver has earned and displays them in descending order. First the program sorts the drivers according to their points in descending order. After that the topic **"Championship standings"** will be displayed in the color cyan and in bold. Then topic for each column will be displayed,

"Name" "Age"   "Team"   "Car"   "Points"

Under those topics the driver details will be displayed in descending order of each driver's points.

5.Simulating a random race (SRR)

```
def simulate_race(details): # Function to simulate a Race between the
drivers in the database
    race_date = input("Enter the date of the race (YYYY-MM-DD): ")
    race_location = input("Enter the location of the race: ")
    race_results = []

    for name, driver in details.items():
        # Randomly assigning a position to each player
        position = random.randint(1, len(details))
        points = 0
        if position == 1:
            points = 10
        elif position == 2:
            points = 7
        elif position == 3:
            points = 5

        race_results.append((name, position, points))
        details[name]['Points'] += points
    # Sort the drivers by their postions
    race_results.sort(key=lambda x: x[1])

    with open('races.txt', 'a') as file: # Open the file races and append
        file.write(f"Race Date: {race_date}\n")
        file.write(f"Location: {race_location}\n")
        for result in race_results:
            file.write(f"Driver: {result[0]}, Position: {result[1]},
Points: {result[2]}\n")
        file.write("\n")

    print(colors.YELLOW + "Race simulated and results recorded!" +
colors.RESET)
```

This function simulates a race among the drivers in the database.

First the user must input the date and the location of the race for the data to record more accurately. Then it will open a list named “race_results”. Then the program will assign a random position for each driver between 1 and length of drivers in details database. Each driver gets points according to their position,

1 position = 10 points

2 position = 7 points

3 position = 5 points

Then the drivers will be sorted according to their position. The points that were received here will add to the current points of each player.

Then the program will create a file name races (if that file doesn’t already exist) and save the race details in that file. Firstly, it will save the location and the date of the race. Then Drivers name, position and points will be recorded (sorted according to their positions). Then the program will print **“Race simulated, and results recorded!”** in yellow color.

6.Viewing the race table sorted according to the date (VRL)

```
def view_race_table(): # Funtion to view the race table sorted according
to the date
    try:
        with open('races.txt', 'r') as file: # open the file races and
read
            print("\n--- Race Table ---")
            print(file.read())
    except FileNotFoundError:# If the file not found
        print(colors.RED + "No race data found." + colors.RESET)
```

This function displays the race table.

At first the function reads the file “races” which was created while simulating races.

Then it will print the topic **“Race table”.** The program will print the content of that file which are,

- Date of the race
- Location of the race
- Driver’s name - Position - Points

If the program doesn’t find a file named races, it will print **“No race data found.”** in color red.

7.Save the current data to a file (STF)

```
def save_data(details): # Function to save the data to a file
    with open('drivers.txt', 'w') as file:# open the file drivers and
write
        for name, info in details.items():

file.write(f"{name},{info['Age']},{info['Team']},{info['Car']},{info['Poin
ts']}\n")
    print(colors.YELLOW + "Data saved successfully!" + colors.RESET)
```

Firstly, this function creates a file, named drivers (if the file doesn’t already exist). Then all the driver details will be saved in this file.

After all the data has been saved the program will print **“Data saved successfully!”** in yellow color.

8.Load data from the saved text file (RFF)

```
def load_data(details): # Function to load the data from saved file
    try:
        with open('drivers.txt', 'r') as file: # open the file drivers and
read
            lines = file.readlines()
            for line in lines:
                line = line.strip()
                if line:
                    name, age, team, car, points = line.split(',')
                    details[name] = {'Age': age, 'Team': team, 'Car': car,
'Points': int(points)}
                    print(colors.YELLOW + "Data has been loaded from the
file" + colors.RESET)
    except FileNotFoundError:# If the file is not found
        print(colors.RED + "No previous data found." + colors.RESET)
```

This function enables the program to load the current data from the text file to enable resume capabilities.

Firstly, the program opens the file "drivers" which was created earlier and reads that file. Then the data from the file will be added to the database. Then the program will print **"Data has been loaded from the file"** in yellow color.

If the file is not found the program will display **"No previous data found."** In red color.

9.Exit the program

```
elif choice == 'ESC': # Manually exiting the program
    save_data(driver_details)
    print(colors.YELLOW + "Exiting the program.....")
    break
```

When the user enters "ESC", save_data function will save the current data to a file. Then the program will display **"Exiting the program……"** in color yellow and the program will end.

# Main Menu

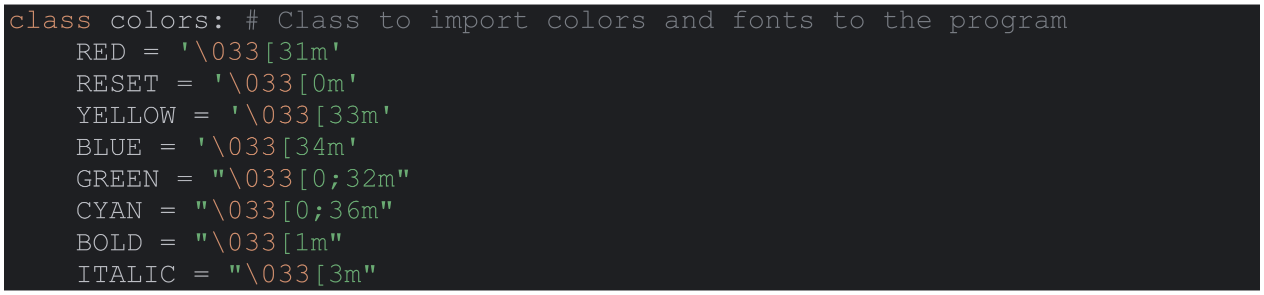
```
driver_details = {}

print(colors.CYAN + colors.ITALIC + colors.BOLD + "\n--- Rally Cross
Management System ---" + colors.RESET)
print(colors.BLUE + "\nType ADD for adding driver details")
print(colors.BLUE +"Type " + colors.GREEN + "DDD" +colors.BLUE + " for
deleting driver")
print(colors.BLUE +"Type " + colors.GREEN + "UDD" +colors.BLUE + " for
updating driver details" )
print(colors.BLUE +"Type " + colors.GREEN + "VCT" +colors.BLUE + " for
viewing the rally cross standings table")
print(colors.BLUE +"Type " + colors.GREEN + "SRR" +colors.BLUE + " for
simulating a random race")
print(colors.BLUE +"Type " + colors.GREEN + "VRL" +colors.BLUE + " for
viewing race table sorted according to the date")
print(colors.BLUE +"Type " + colors.GREEN + "STF" +colors.BLUE + " to save
the current data to a text file")
print(colors.BLUE +"Type " + colors.GREEN + "RFF" +colors.BLUE + " to load
data from the saved text file")
print(colors.BLUE +"Type " + colors.GREEN + "ESC" +colors.BLUE + " to exit
the program" + colors.RESET)


while True: # Looping the program until manually breaks

    choice = input("\nEnter your choice: ")

    if choice == 'ADD':
        add_driver(driver_details)
    elif choice == 'DDD':
        delete_driver(driver_details)
    elif choice == 'UDD':
        update_driver(driver_details)
    elif choice == 'VCT':
        display_standings(driver_details)
    elif choice == 'SRR':
        simulate_race(driver_details)
    elif choice == 'VRL':
        view_race_table()
    elif choice == 'STF':
        save_data(driver_details)
    elif choice == 'RFF':
        load_data(driver_details)
    elif choice == 'ESC': # Manually exiting the program
        save_data(driver_details)
        print(colors.YELLOW + "Exiting the program.....")
        break
    else: # Invalid input was entered
        print(colors.RED + colors.BOLD + "Invalid choice. Please try
again." + colors.RESET)
```

This contains the main menu of the program. Firstly, the program displays all the functions that the user can do with the program. Function and the topic are displayed in blue color and the key word for each function is displayed in green color.

After that the while loop begins and it will keep looping until the program manually breaks. The program asks the user to enter the key word to do each function of the program.

If an invalid input was entered by the user, the program will display **"Invalid choice. Please try again."** in red color.

# class colors



```
class colors: # Class to import colors and fonts to the program
    RED = '\033[31m'
    RESET = '\033[0m'
    YELLOW = '\033[33m'
    BLUE = '\033[34m'
    GREEN = "\033[0;32m"
    CYAN = "\033[0;36m"
    BOLD = "\033[1m"
    ITALIC = "\033[3m"
```

This class has been used to import colors and fonts to the program. This is used to give different colors and fonts to different messages so that the user can navigate through the program very easily.

Test Plan

Test component	Test no.	Test input	Expected results	Actual Results
ADD – Adding a Driver	01	Name – John Age – 20 Team – Benz Car – Benz AMG Points - 10	Display – “Driver details added successfully!”	“Driver details added successfully!”
DDD – Deleting a driver	02	Name - John	Display – “Driver deleted successfully!”	“Driver details deleted successfully!”
DDD – Deleting a driver who is not on the database	03	Name - John	Display – “Driver not found!”	“Driver not found!”

Conclusion and Assumptions

This program is an application for a world rally cross championship management. In this program the user can use different functions to manage a database of drivers and simulate a race and record its data. Users can also save all the data to files and can access these files through the program.

All the errors that can occur have been addressed and all the functions are working properly.

Reference List

Documents

- Mr. Iresh Bandara – Class materials

Website

- freeCodeCamp.org - https://www.freecodecamp.org/news/python-lambda-function-explained/#:~:text=lambda%20is%20a%20keyword%20in,what%20you%20want%20to%20achieve
- Simplilearn YouTube Chanel - https://www.youtube.com/watch?app=desktop&v=P8MdDCTbMOI