|  |
| --- |
| H446 Programming Project |
| F1 Fantasy League |
| Lord Williams School |
| Tom Draper (Student)  7-7-2021 |

Table of Contents

[Analysis 2](#_Toc99703167)

[Problem and Explanation: 2](#_Toc99703168)

[Stakeholders 2](#_Toc99703169)

[Interview with stakeholders 3](#_Toc99703170)

[Research 3](#_Toc99703171)

[API 3](#_Toc99703172)

[GUI 5](#_Toc99703173)

[Client Server Networking 5](#_Toc99703174)

[Encryption 10](#_Toc99703175)

[GDPR and Data Protection Act 13](#_Toc99703176)

[Comparison of Existing Programs 13](#_Toc99703177)

[Desired Features 13](#_Toc99703178)

[Design 14](#_Toc99703179)

[Server Login 14](#_Toc99703180)

[GUI 16](#_Toc99703181)

[Points System 19](#_Toc99703182)

[Client Server Chat 21](#_Toc99703183)

[Class Diagram 22](#_Toc99703184)

[Database Structure 22](#_Toc99703185)

[Development Method 22](#_Toc99703186)

[Development 23](#_Toc99703187)

[Week 1 – Creating Points Distribution 23](#_Toc99703188)

[Week 2 – Points Calculation Algorithm 28](#_Toc99703189)

[Week 3 – Saving Raw Points 29](#_Toc99703190)

[Week 4 – Threading Points Calculation 31](#_Toc99703191)

[Week 5 – Final Points Calculation 32](#_Toc99703192)

[Week 6 – Server-User Login 35](#_Toc99703193)

[Week 7 – Networking for login 38](#_Toc99703194)

[Week 8 – Create User’s Team Database 40](#_Toc99703195)

[Week 9 – Add/ Get Records From Team Database 43](#_Toc99703196)

[Week 10 – Login GUI 46](#_Toc99703197)

[Week 11 – Login Functionality 47](#_Toc99703198)

[Week 12 – Main Window 50](#_Toc99703199)

[Week 13 – Edit Team Menu 58](#_Toc99703200)

[Week 14 – Main Menu Label Edit 63](#_Toc99703201)

[Week 15 – Error Handling 66](#_Toc99703202)

[Week 16 – Chat Room Development 68](#_Toc99703203)

[Testing 72](#_Toc99703204)

[Point Tests 72](#_Toc99703205)

[Login Tests 72](#_Toc99703206)

[Main Menu Tests 74](#_Toc99703207)

[Edit Team Menu Tests 75](#_Toc99703208)

[Evaluation 75](#_Toc99703209)

[Module Dependency Visualization Diagram 75](#_Toc99703210)

[Stakeholder feedback 75](#_Toc99703211)

[Overall Conclusion 76](#_Toc99703212)

[Maintainability and Future of program 76](#_Toc99703213)

[Complexity 76](#_Toc99703214)

[Future of program 76](#_Toc99703215)

[Final Code 76](#_Toc99703216)

[Client 76](#_Toc99703217)

[main.py 77](#_Toc99703218)

[functions.py 77](#_Toc99703219)

[scripts/login.py 79](#_Toc99703220)

[scripts/main\_menu.py 81](#_Toc99703221)

[scripts/chat.py 86](#_Toc99703222)

[Server 86](#_Toc99703223)

[main.py 86](#_Toc99703224)

[login/network.py 88](#_Toc99703225)

[points/points\_recorder.py 91](#_Toc99703226)

[points/points\_distribution.py 95](#_Toc99703227)

[chat/chat\_server.py 97](#_Toc99703228)

# Analysis

## Problem and Explanation:

For my project I am planning on making a F1 Fantasy League Game similar to how Fantasy Football works, but for Formula 1 Racing. It works by assigning each player a price based on their performance in the sport. The user creates a team and the better the team of drivers perform in real life, the more points that team receives. To increase the social competitiveness amongst players you can compare your team’s points with your friends and others through the use of a built in chat room, which could also be used to talk about the sport with others.

## Stakeholders

I have selected a group of F1 fans of ages 16-18 to represent my target audience, so that the program matches what they would expect and want out of a F1 Fantasy League. This will help me obtain information on what they would like and dislike when I present prototypes of the program, and will help me stick to what the target audience would want.

Due to the scope of my project my target audience is any enjoyer of Formula 1, and the chat will have a profanity filter option so that any age can use the program.

## Interview with stakeholders

I asked the stakeholders a few questions so that I know what they want an F1 fantasy league to offer.

Me – “If you were starting a fantasy F1 team what would you expect the service to offer?”

Sam - “A good GUI that shows how points will be attained”

James - “A Chat would be nice so that users could talk to each other”

Me – “What would you expect to change scores?”

James - “Not really sure, but I would expect things like final position to affect scores”

Chris - “I think it would be good if qualifying could affect the scores”

Me – “What do you currently most enjoy about fantasy football?”

Sam - “To compete against others and have the winning team”

Me – “Would you care if the project didn’t have the best looking GUI?”

Chris – “No as long as all that is needed is in it”

James – “But it would be good for it to look nice”

From this interview I was able to extract some stakeholder requests:

* Easy to navigate GUI
* Chatroom
* Good points calculation that includes qualifying results
* Easy to understand how points are distributed

## Research

### API

When thinking of the project I knew I had to find a way to get all the data needed, such as race winners fastest lap and lots of other data to help compile prices and points system. When searching for a way to get this data I discovered a website called [Ergast Developer API](http://ergast.com/mrd/), This is a free API that can provide years of data and is updated after new races.

I researched the use of the API to make sure it was easy to navigate and worked well with Python (so I got It to retrieve all year’s race results since 2015):

#### Example Request

In this example we will request data from the 10th race of 2015

Request:

GET <https://ergast.com/api/f1/2015/10/results.json>

No API key is needed for this request.

Response JSON:

{"MRData":{"xmlns":"http:\/\/ergast.com\/mrd\/1.5","series":"f1","url":"http://ergast.com/api/f1/2015/10/results.json","limit":"30","offset":"0","total":"20","RaceTable":{"season":"2015","round":"10","Races":[{"season":"2015","round":"10","url":"http:\/\/en.wikipedia.org\/wiki\/2015\_Hungarian\_Grand\_Prix","raceName":"Hungarian Grand Prix","Circuit":{"circuitId":"hungaroring","url":"http://en.wikipedia.org/wiki/Hungaroring","circuitName":"Hungaroring","Location":{"lat":"47.5789","long":"19.2486","locality":"Budapest","country":"Hungary"}},"date":"2015-07-26","time":"12:00:00Z","Results":[{"number":"5","position":"1","positionText":"1","points":"25","Driver":{"driverId":"vettel","permanentNumber":"5","code":"VET","url":"http:\/\/en.wikipedia.org\/wiki\/Sebastian\_Vettel","givenName":"Sebastian","familyName":"Vettel","dateOfBirth":"1987-07-03","nationality":"German"},"Constructor":{"constructorId":"ferrari","url":"http:\/\/en.wikipedia.org\/wiki\/Scuderia\_Ferrari","name":"Ferrari","nationality":"Italian"},"grid":"3","laps":"69","status":"Finished","Time":{"millis":"6369985","time":"1:46:09.985"},"FastestLap":{"rank":"5","lap":"63","Time":{"time":"1:26.772"},"AverageSpeed":{"units":"kph","speed":"181.759"}}},

…

]}]}}}}

This json only includes one driver in the list, to decrease size.

Code:



Defines what years you want to search and how many rounds each year.

Sends an API get request to receive the json data which gets converted into a dictionary data structure.

Shows the position, name, end reason and fastest lap time of each driver.

Shows which race the driver data is coming from.

Console output:

As there was a lot of data outputted this is just a snippet of the output near the end.



Shows each divers position at the end of each race.

Displays the drives name and their reason for ending the race.

It also displays each drivers fastest lap, this will help provide extra points if the data is analysed.

This provided a good insight as it showed that the free API I found, provided all the data I would need and would work perfectly for getting the data needed, that can be processed and stored on the server.

### GUI

I looked at many GUI modules for Python or using a web-based interface (Django) .I have been looking at many GUI’s for Python and have compared and ranked them.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rank | Name | Integration with Python | Ease of use/ programming | Best looking result |
| 1 | PyQt | 4/5 | 5/5 | 5/5 |
| 4 | Tkinter | 5/5 | 3/5 | 2/5 |
| 3 | easygui | 3/5 | 3/5 | 3/5 |
| 2 | Kivy | 3/5 | 4/5 | 4/5 |

I looked at using Tkinter as I was taught the basics of this GUI module, so I could easily learn the enough to make my project, but I decided against using Tkinter as it had a very old and outdated looking design and code was hard to follow. So instead I looked at PyQt as this looked much better, good support for widgets and it also had build in software (qt Creator) which helps mould the GUI to whatever I need.

I decided to make a simple word processing program to test the viability of using this as my primary GUI tool.

### Client Server Networking

I needed a way to send data to the clients so that the data on the drivers and their team can be transferred from the server to ensure no cheating can occur and also so that a functional chat group could be active and the only way to do this was through a client-server network

#### Program Login

So that users fantasy team data can be accessed from any device wherever, we have to have some kind of login system so that players can save their data to a server, and it be sent back when they login using there unique credentials.

I decided to test if I would be able to make this system, to I made a simple client server login page that asks for a name and pin to access.

**Server.py:**

The server code sets up the connection and waits for an input, to which is processes the result and returns whether it’s a valid login or not.

The server.py as a test only has a 2-dimensional list to store the correct name and pin, this is not secure or efficient for my project but as I was testing whether I could create a system like this using client-server connection it was enough to store the users name and pin to be searched through.

Code:



Send back accepted or denied to the client.

Searches logins for any matches with the data they sent

Defines Host IP and port

Simple client name and pin storage

Accepts connection from client

Console output:



Shows the inputs and the result from the server.

This was the only input to be incorrect and the code spotted this and denied access.

**Client.py:**

Connects with the server and send the inputs it receives, then waits for a response from the server. After which it processes the response and outputs a Boolean True/False response.

Code:



Defines host IP and port that the client needs to connect to.

Sends the name and pin provided, they had to be sent in one string. Then receives the result processes it and returns True or False

The provided test set to see if the program works correctly.

Console output:



The client shows that the last one was denied and the rest where accepted.

#### Chatroom

I decided to add a chatroom so that the users can talk to each other and give more reasons to use my f1 fantasy league rather than alternatives.

I decided to make a mock-up of a quick client-server chatroom to show the basic aspects.

**Server.py:**

The server has to receive each chat message from the clients and redistribute that message to all the clients so they receive what was sent by one user.



Opens a text file that will log all messages received.

Receive code ‘send’ if the client wants to send a message.

If client wants to receive messages, then the sends the last message sent.

Adds to chatlog and updates message variable based of message what’s received.

Defines host IP and port that the client needs to connect to.

**Client.py:**

The client has to send the server each message that the user inputs, and also receive and display any other messages sent.



The message() subroutine sends the initial string to tell the server what the connection is going to be used for, and then sends the server the string that is received by the message\_constant() subroutine.

To run the message() subroutine constantly.

A subroutine which sends out a receive code so that the server sends back the last message it has received, which the client checks if it is a new message before displaying it.

This runs both the message\_constant() and receive\_messages() subroutines at the same time using multiprocessing.

In conclusion this demo application proved that I can implement a simple chatroom using sockets and Python, and therefore this is realistic to include within this project.

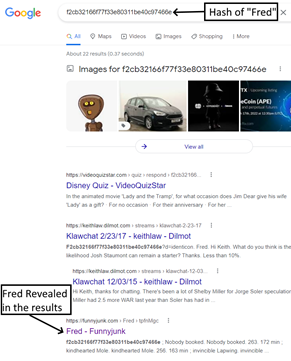
### Encryption

#### Hashing

To prevent potential leakage of password information, you can just never store the actual password in a database, this can be done by making use of a hashing algorithm as you feed your passwords in and the algorithm creates a hash of the password which can then be stored. This means that anytime you must access the database you will not be able to see people’s passwords which further increases security, and the implications of any data breach.

When researching I found that there are many types of hashing algorithms that are used. I researched some of the major hashing algorithms that already have some implementation into Python and compiled them into a small table to compare them.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Hashing Algorithm | Sha256 | Blake2b | Blake3 | Argon2 |
| Speed (MiB/s) | 484 | 1312 | 6164 | 412 |
| Can utilise ‘salt’ | Yes | Yes | Yes | Yes |
| Security | low | medium | medium | Highest |



“Salt” is the inclusion of additional data alongside the password before hashing to prevent the use of large dictionaries of hashed values to decrypt the password. Many unsalted hashes of common words and names can easily be revealed using google.

Conclusion:

Due to speed not being a huge issue as it is only being used to hash the passwords when creating and verifying a password, I believe that Argon2 is the best option to use for my project for its high security and easy implementation. Argon2 also won the Password Hashing Competition.

I wanted to see if it was possible to integrate Argon2 into Python, so I created a simple script that takes an input and verifies it against a hashed password and outputs True or False based on whether the input matches the password:



This if statement checks If the hash has been created, and if it has not been it creates the hash and saves it to test.txt

Opens test.txt and assigns the hash variable as what’s in the file

Imports the argon2 Python module

The contents of test.txt, which stores the hash of the password.

Checks if the hash is out of date and replaces the outdated hash with a new updated hash.

Checks the user input against the stored hash and outputs True or False based of the result.



Using the correct password it outputs True, as expected.

Using the incorrect password it outputs False, as expected.

Console output:

After creating this script and making sure it is easy to integrate into Python. I feel comfortable integrating it into my project to store passwords and to increase security.

#### Encrypt network activity

When researching what algorithms or methods I could use to encrypt the network messages I found an encryption algorithm called AES which is a symmetric block cipher.

This algorithm looked promising, so I created a small encrypt/decrypt script:

When creating this script, I decided that it would be more secure if the day, month and current hour are all used to help make the second key or IV.



Imports the AES encryption and datetime Python modules.

These are the variables for the Key and IV. The IV (key2) changes based off the Hour, day and month the code is run on, so you never get the same number.

This while statement checks that the message is 16 bytes long as the algorithm requires that, it also records how many padding characters it used so they can be removed after decryption.

The function then returns the encrypted string and how much padding was used.

This decrypts the string and removes all the padding which is then returned.

This runs both the functions with ‘#MesSagE63’ as the message and outputs the values that are returned throughout the process.

Console Output:

This is the encrypted string that would be transmitted and it has 5 padding characters.

It outputted the same value that was encrypted, this shows that it works perfectly.

Conclusion:

After creating this script I feel comfortable using the AES algorithm to encrypt the network traffic that will be sent back and forth.

### GDPR and Data Protection Act

Due to the new laws in place I need to ask for permission before storing any personal data and be able to remove or locate and send all data linked to them if requested.

As I will not be storing any sensitive data such as credit information or full name, this means that it will not cause me any great issues. To abide by these rules, I will have a tick box to allow me to store their username and password and any messages they send, and if someone requests their data then all databases will be searched with their username and will send it over to them.

## Comparison of Existing Programs

Currently there is a similar [Official Formula 1 Fantasy](https://fantasy.formula1.com/) web application that is similar to what I would like to achieve. I hope to compete with this program in the future.



Shows next race to be held.

Shows your created teams.

Shows each driver’s name what team they are in and a photo of them, to help anyone that may be confused.

Shows how the price/value of that driver has changed recently, to help people make good decisions about their drivers



The Official Formula1 Fantasy program has lots of good features but does not provide a chatroom to be able to talk with other people about the sport or about their program, this is what my project will include to bring the community together.

When browsing the website, I liked the recent price/value change as this helps people make informed decisions to add their drivers, so I have decided to try and add this to my project to show the change in price over 2 races.

I also saw that they have lots of animated graphics that I do not have the skill or time to implement as I will focus on the minimum viable product.

Their website also includes advertisements, many users may find this annoying and intrusive, so my project will not include any advertising or sponsored content as I do not intend to make money from this program at this current time.

## Required Features

For my project I need to incorporate, for minimum viable product (MVP), with MoSCoW prioritisation (as defined in <https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation> ):

M – A client-server chatroom

M - Web scraping/API usage to get the race results and driver data needed.

M - A scoring system that analyses the data from races, to give each driver a point score.

M – A price algorithm that converts points into cash.

M - Get the next race information and when it occurs and display it.

M - A login system so that user’s data can be transferred to different machines, and so users only get to see their own data

M – An interface to allow the user to edit and display their team

S – A good-looking GUI to show all the data needed and provide a way around the program

S - Password Hashing so no passwords are stored in databases, limiting the effect of any data breaches.

C - Encryption of data that is sent between the server and clients to prevent man in the middle attacks.

W – Introduce complicated animated graphics, this is beyond what is achievable during this MVP, but is recorded here for future development.

## Success Criteria

The implementation of the desired features defined above will be used to evaluate the success of the project, with the relative importance defined by the MoSCoW prioritisation, so successful implementation of all of the must-have “M” features will be considered a success. Including further features in addition will be further improvements beyond a simple achievement of the MVP of the project.

## Limitations of the MVP

The initial MVP developed during this project will have the following limitations:

* The system will only be tested supporting 3 concurrent users.
* The system will initially be deployed on my computer and will not be available when it is switched off.
* The system will initially only be accessible within the local network of the server, and will not be accessible from the internet.

# Design

## Server Login

For my login page I would like to have a username and password login system so that the username can be used later for the chatroom. I designed a flowchart of what the algorithm will be doing:



This flowchart will help me write the login page and help understand how it needs to work.

This flowchart diagram does not include any encryption on the data before sending it to the server but the actual project should incorporate this.

I have designed a simple design for what the login page should look like on the client side.



Title and Logo to engage the user.

Fields to enter the username and password.

A tick box to accept Terms and conditions which contain how their data will be used so I do not violate any GDPR laws.

When submitted it closes the window and sends the data over to the server.

Currently I have no logo or a design choice so the design only consists of what is needed for the login page to function as aesthetics can easily be added or changed in the future. This design was improved as part of the GUI design to help match the design of the other pages.

## GUI

From my stakeholder interview the GUI needs to:

* Be easy to navigate.
* Show the users team value and recent changes.
* Provide easy to use chat.

So, I designed a wireframe of what each page would look like to make sure I include all the features that are required:

Menu:



Team edit screen:



Login and register pages:



Recent driver changes page:



This gives me a reference for when I am creating the GUI so that it can be as close to my visions as possible.

**Wireframe Storyboard:**

Making a GUI page diagram will help show how to get to the different pages of the GUI through the button menu. This shows how navigation in my program will work and how to implement it.



## Points System

As the entire project relies on the value of teams and drivers changing and adapting, the points need to be allocated correctly and fairly. So I have created a points list that can be integrated into the program during development, and they can easily be changed later.

All Drivers and Constructors will start with a base of 250 points, to stop any negative values.

Driver:

Qualifying Points:

Pole position / +3 Points

Second / +2 Points

Third / +1 Point

Made it into Q2 / +5 Points

Made it into Q3 / +5 Points

Qualified better than teammate / +3 Points

Qualified worse than teammate / -2 Points

Race Points:

|  |  |
| --- | --- |
| Race position | Points |
| 1 | 25 |
| 2 | 18 |
| 3 | 15 |
| 4 | 12 |
| 5 | 10 |
| 6 | 8 |
| 7 | 6 |
| 8 | 4 |
| 9 | 2 |
| 10 | 1 |
| 11-20 | 0 |

Race finish / Points as show in table

Fastest Lap / +1 Point

Positions gained from grid / +1 Point per position gained

Positions lost from grid / -1 Point per position lost

Finished above teammate / +3 Points

Finished below teammate / -3 Points

Did Not Finish (DNF) / -25

Constructor:

Qualifying Points:

Driver on pole position / +3 Points

Driver second / +2 Points

Driver Third / +1 Point

Driver made into Q2 / +5 Points per driver

Driver made into Q3 / +5 Points per driver

Both drivers made it into Q3 / +3 Points

Race Points:

Race finish / Points per driver as shown in table

Fastest Lap / +1 Point

Driver positions gained from grid / +1 Point per driver

Driver positions lost from grid / -1 Point per driver

Driver Did Not Finish (DNF) / -25 Points per driver

Finished top 10 / +3 Points

Each user will start with ₤100,000,000 to create their team and to convert the points values above into currency by multiply their point values by 20.

Here is a flowchart the calculation for points:



## Client Server Chat

For the client-server chat I created a flow chart for how it would work on the server and on the client so that it would be easier to properly integrate and add it to my project.

**Server:**



**Client:**



These flowcharts were very helpful as they showed me I would have to make use of multithreading in my project to be able to send and receive messages at the same time.

## Class Diagram

ADD

## Database Structure

For my project I will be using SQLite3 as it is very well integrated into Python, so will be the most simple to set up.

I will need at least 2 tables, one for login details, and one for the user’s team data, with the formats below.

Login table:

|  |  |  |
| --- | --- | --- |
| UserID | Username | Password hash |
| Unique Integer | String | String |

Team Data table:

|  |  |  |  |
| --- | --- | --- | --- |
| UserID | Constructor | Remaining Drivers | Remaining cash |
| Unique Integer | string | List stored as a string | Integer |

## Development Method

For my development I will be using modified RAD (Rapid Action Development) methodology so that with each iteration I can make a prototype and get feedback from the stakeholders on how to improve or adapt the project to help meet deadlines or to make a better product. This means I may improve my project design as development progresses.



Using this diagram I created, at the start of each iteration I will set an expectation list to complete each week then design implement and test the new code. Stakeholder’s feedback will help me evaluate that week to help set the expectation for the next week. This means if I set an expectation that is too much I can move it over to the next iteration and expect a more reasonable amount of work, or if I set my expectation too low I can increase what is expected for the next iteration.

During development I will also be constantly committing my code on GitHub. This will allow me to have version control, so I can roll back to any point if I create an unwanted error or any other reason.

Testing per iteration:

# Development

## Week 1 – Creating Points Distribution

This week I would like to get a server prototype up and running, to do this I need to:

* Get the server to a connectable status.
* Create a points distribution, a file that contains the point values for different tasks.
* Get the server to fetch and sort the information from Ergast F1 API and store the data for each race that has taken place.
* Use the data to calculate and store the points for each driver and constructor.
* Check if the points system creates fair point distribution.

I started by creating a separate json format Python file to store the amounts of points gained and lost through tasks completed by the drivers.

#points\_distribution.py

#Points given for tasks  
assign\_points={  
 'DriverPoints':{  
 'Race':{  
 'Results':{  
 '1':25,  
 '2':18,  
 '3':15,  
 '4':12,  
 '5':10,  
 '6':8,  
 '7':6,  
 '8':4,  
 '9':2,  
 '10':1,  
 'F':-25, #F,D,W,R,N is DNF  
 'D':-25,  
 'W':-25,  
 'R':-25,  
 'N':-25  
 },  
 'fLap':1, #Fastest Lap  
 'PGFG':1, #Points per Position gained From grid  
 'PLFG':-1, #Points per Position lost From grid  
 'FAT':3, #Finished above teammate  
 'FBT':-3, #Finished below teammate  
 },  
 'Qualifying':{  
 'Results':{  
 '1':3,  
 '2':2,  
 '3':1,  
 },  
 'RQ2':5, #Reached Q2  
 'RQ3':5, #Reached Q3  
 'QBT':3, #Quallified Below Teammate  
 'QWT':-3 #Quallified Above Teammate  
  
 },  
 'Constructor':{  
 'Race':{  
 'Results':{  
 '1':25+3,  
 '2':18+3,  
 '3':15+3,  
 '4':12+3,  
 '5':10+3,  
 '6':8+3,  
 '7':6+3,  
 '8':4+3,  
 '9':2+3,  
 '10':1+3,  
 'F':-25, #F,D,W,R,N is DNF  
 'D':-25,  
 'W':-25,  
 'R':-25,  
 'N':-25  
 },  
 'fLap':1, #Fastest Lap  
 'PGFG':1, #Points per Position gained From grid  
 'PLFG':-1, #Points per Position lost From grid  
 },  
 'Qualiflying':{  
 'Results':{  
 '1':3,  
 '2':2,  
 '3':1,  
 },  
 'RQ2':5, #DriverReached Q2  
 'RQ3':5, #Driver Reached Q3  
 'BQ3':3, #Both drivers reach Q3  
 }  
 }  
 }  
  
}

I also created the start of the points recorder for each driver, and it record all driver the names of all f1 drivers

import requests #import needed modules  
import ast  
def get\_drivers():  
 data = requests.get('http://ergast.com/api/f1/drivers.json?limit=1900&offset=30')  
 drivers = ast.literal\_eval(data.content.decode())['MRData']['DriverTable']['Drivers']  
 detail\_driver\_list = []  
 file = open('drivers.txt', 'w+')  
 for driver in drivers:  
 try:  
 driver\_id = driver['driverId']  
 full\_name = f"{driver['givenName']} {driver['familyName']}"  
 nationality = driver['nationality']  
 DoB = driver['dateOfBirth']  
 number = driver['permanentNumber']  
 except Exception:  
 number = 'n/a'  
 detail\_driver\_list.append([driver\_id,full\_name,nationality,DoB,number])  
 file.write(str(detail\_driver\_list))  
  
def get\_race\_data():  
 from datetime import date  
 for year in range(1950,date.today().year+1): #Cycle through all years of f1  
 total\_races\_in\_year = ast.literal\_eval(requests.get(f'http://ergast.com/api/f1/{year}.json').content.decode())['MRData']['total'] #Get all rounds in the year  
 for race in range(1,int(total\_races\_in\_year)+1): #Cycle through all the rounds in the year  
 race\_data = ast.literal\_eval(requests.get(f'http://ergast.com/api/f1/{year}/{race}/results.json').content.decode()) #Get the race data   
 quali\_data = ast.literal\_eval(requests.get(f'http://ergast.com/api/f1/{year}/{race}/qualifying.json').content.decode()) #Get the quali data  
 assign\_driver\_points(race\_data, quali\_data)  
  
def assign\_driver\_points(rData, qData): #use race data to assign points to the driver  
 #print(len(rData['MRData']['RaceTable']['Races'][0]['Results']))  
 for driver in rData['MRData']['RaceTable']['Races'][0]['Results']:  
 driver\_id = driver['Driver']['driverId']  
 finish\_position = driver['position']  
 grid\_position = driver['grid']  
 constructor = driver['Constructor']['name']  
 try: fastest\_lap\_position = driver['FastestLap']['rank']  
 except KeyError: fastest\_lap\_position = 22  
 driver\_points = get\_race\_points(finish\_position ,grid\_position ,constructor ,teammate\_position , fastest\_lap\_position)  
  
def get\_race\_points(fPos, gPos, team, teammatePos, fLapPos): #Get all data required and create  
 points = 0  
 from point\_distribution import assign\_points  
 try: points = points + assign\_points['DriverPoints']['Race']['Results'][fPos] #add points for finishing finishing top 10 or minus 25 for not finishing  
 except KeyError: points = 0  
 points = points + ((int(gPos)-int(fPos)\*assign\_points['DriverPoints']['Race']['PGFG'])) #add points for position gained from grid  
 if fLapPos == 1: points = points + assign\_points['DriverPoints']['Race']['fLap'] #add points for achiving fastest lap  
 if fPos > teammatePos: points = points + assign\_points['DriverPoints']['Race']['FAT'] #add points for finishing above teammate  
 elif fPos < teammatePos: points = points + assign\_points['DriverPoints']['Race']['DBT'] #add points for finishing below teammate  
   
 return points  
  
get\_drivers()  
get\_race\_data()

Due to the research and lack of knowledge the points calculation is taking longer than expected and I was doubting the efficiency due to my program having a Big O notation of O(n2) and as the data sets it has to go through are large and requesting them takes time I was worried about the efficiency of this algorithm before I had finished it, and it would have to include another loop for each team, so I decided to create a new flowchart for my points system which gets each driver of a team’s points at the same time:

This new algorithm uses less loops to bring down the runtime of the points calculator, and will make use of multiprocessing on the subprocess so that it can run even quicker.

This also allows me to calculate each teams’ drivers and calculate the constructor’s points all at once loosing the need for another loop.



## Week 2 – Points Calculation Algorithm

Last week I decided to remake the points calculation algorithm as it did not include a way to calculate constructor’s points and had long runtime due to many nested loops

This week I would like to get the points system up and running, to do this I need to:

* Get the server to fetch and sort the information from Ergast F1 API and evaluate the data for each race that has taken place.
* Use the data to calculate and store the points for each driver and constructor for every race.
* Store this data to an external database/file.
* Check if the points system creates fair point distribution.

I was able to get the points gained and lost for teams and drivers for every race, but this does not make any use of qualifying data which also needs to be included, and it does not store the data to storage yet.

This is the code I have created this week gets data of every year by getting all the race data and process it and return and print the points values for every race.



This subroutine cycles through the years, gets the number of races that year and then cycles through each race that year, where it then gets the race and qualifying data and provides it to the assign\_driver\_points subroutine.



The assign\_driver\_points subroutine extracts the data required and creates a new data structure, filtering the data required and restructures the data into a dictionary with the constructor as the key, this combines drivers with the team they drive for. It then cycles through each team and provides both drivers’ data to the get\_points subroutine.

The get\_points Subroutine uses both drivers data to assign points based on tasks they completed. It uses the numbers from point\_distribution.py to get the total points for each task, then returns the calculated points for each driver and the team.

This code only currently makes use of the race data, so I know next week I need to have it include race data and to save all this data to a file.

## Week 3 – Saving Raw Points

This week I would like to get the points system finally up and running, to do this I need to:

* Use the qualifying data and extract points required to add to the race points.
* Make use of threading to speed up the calculations.
* Make the current year double the points.
* Store this data to an external database/file.
* Check if the points system creates fair point distribution.

When researching to add qualifying data to my program I realised the qualifying result is the same as their starting position on the grid so I can just use the already created gPos variable to assign points

When I started programming I noticed an issue with using the created gPos variable, anyone that took engine penalties or other kind of grid penalties would not gain points for their qualifying position, I decided that to help the runtime and simplify the code that for now it will use the starting grid position of the drivers, if this creates issues in the results of the points then I can introduce it back at a later date.

I started by adding threading to the code so that all the years would run in parallel instead of through nested loops, which dramatically decreases runtime.

It creates a thread for every year and then loops through the list of threads and runs them, then it waits for the last thread to finish before continuing the code. This reduced the runtime from about 92 seconds to 21 seconds.

I also added the qualifying points based of the grid position of each driver

This means that it now calculates all the points it needs to.

Then to save them to a file I created a subroutine save\_points that is called in the assign\_driver\_points file after using the data from the get\_points subroutine.



This means that all the points created by the code will be recorded.

I have noticed one issue when I run the code, and that is that the file created is always a different size, I assume it is due to two threads writing to the file at exactly the same time, so I will need to address this next week.

## Week 4 – Threading Points Calculation

Last week I added the calculation of qualifying data, and made use of threading to speed up the time in which the code takes to run, but by making use of threading it caused an issue where some data was being deleted as art o a race condition between threads.

This week I would like to fix the issues caused by threading, to do this I need to:

* Find an alternative to writing straight to the file.
* Maybe slow down the threads.
* Make sure no data is being deleted

When searching for a solution I found a thread safe [Python queue module](https://docs.python.org/3/library/queue.html) that is a queue data structure, which is thread safe for multiple thread inserting data simultaneously. So I would be able to have all the threads added to the queue and a loop running in parallel that takes data from the queue and writes it to the list.

I started integrating the queue module by initialising the queue at the start of the get\_race\_data subroutine



I had to make the “q” and “finished” variables global so that the save\_to\_file loop and each thread could access the data they need.

The save\_points subroutine only had to be changed slightly so that instead of saving to a file it just adds to the queue.



Then the save\_to\_file subroutine has to take the data from the queue and write it to the external data file.



I decided to run the save\_to\_file subroutine in parallel with the threads by adding it to the start of the list of threads that will be executed and it waits for the save\_to\_file thread to finish before moving on.

But as the bug which deleted some data through threads writing to the same area at the same time did not go away completely, I decided to add a break for every 10 threads so that it has no chance to delete any needed data.

Graphical user interface

Description automatically generated with low confidence

Even with this change it runs much quicker than without threading and barely makes a difference to the total runtime.

## Week 5 – Final Points Calculation

Last week I fixed a major issue with data deletion and implemented a queue system to get that done. This was very helpful and now means I can move on and finish the points calculation and start the user interface and network for the project.

This week I would like to total the points and save them to a file which can easily be navigated, to do this I need to:

* Analyse the raw data.
* Total each driver’s points from both driver1 and driver2 data.
* Total each team’s points from the raw data.
* Save each driver’s points and team points in an easy to navigate data format.

So I created a subroutine that reads the raw data file and takes the team and each driver’s points and groups them with their drivers then totals the points, sorts them and saves the results as a json file and a separate json file for the team points.

Text

Description automatically generated

I noticed I had coded some repetition into my code so I created a list of a list of variables that are used in each loop and then created a loop for each item in the list as shown below.

Text

Description automatically generated

The save\_final\_points subroutine just takes the data and filename in and saves the data into the file and changes all ‘ into “ as required for the json format.

Text

Description automatically generated

This creates two files driver\_points.json and team\_points.json.

A snippet of driver\_points.json looks like this

Text

Description automatically generated

And team\_points.json looks like this

Text, letter

Description automatically generated

The screenshots don’t show all the data but show how the data is stored and how it will be navigated when used later.

When I ran the code I noticed lots of teams that have been in the sport for a long time such as Ferrari and Williams came out as the highest points, so to counter this I decided to x10 all points in the most recent season. I did this by changing the points value just before it is saved.



## Week 6 – Server-User Login

Last week I finished the calculation and recording of team and driver points. So, this week I want to start the server-user login to a base level so that it can be incorporated into the GUI.

To start the code for the client-server login I need to:

* Create a connection between the client and server
* Send the login information to the server
* The server needs to take the data, hash the password compare them and accept or decline the client
* Then both the server and the client need to proceed through their code.

Before I created the client-server login I decided to move all the files that are to do with points calculation into a separate file to keep the project organised

So the project folder looks like this when it is run.



To do this I needed to change all references to other files such that they are placed and looked for inside the points folder. But this will help to keep my project tidy and easy to navigate.

To start the login page I need to create the frame of what the program would need to make use of in the backend.

I started by creating an SQL database for the login information and link it to the Python file



This creates a table called logins with 2 columns, username and hashpass to store the usernames and passwords and imports the hashing algorithm so that it can verify hashes.

I then created the class that handles each user that logs on.



The ‘Account’ class uses the username and finds the connected password hash if there is one, and the account variable states whether there is an account under this name which can be checked if vaid\_account() is run. Then if it is a valid account it can run check\_pass() which takes the given password and verifies it against the hash value.

To make use of this class a simple subroutine needs to be created.



I created a login subroutine which uses the account username and the password given and then checks if the username is valid by running check\_pass() with the given password which can then return a true or false, and the false has a reason attached.

I now need to create a subroutine and method to register an account into the database.



This subroutine uses the username to check if there is already an account and if there is not then it runs the create\_record() method.



This method inserts the username and password and also increments until it finds an open ID value. This is only possible because I changed the table initialiser to only accept unique values

This week I was able to complete the server side backend and database that will take, compare and register new accounts. This structure can be built on next week to start working on sockets and connecting to the client.

## Week 7 – Networking for login

Last week I created the database and the methods to communicate with it and the hashing algorithm, but I was not able to incorporate networking so that will be done this week.

To continue the code for the client-server login I need to:

* Create a connection between the client and server
* Determine where the user is logging in or registering.
* Take the data and send it to the server.
* Then the server will use last week’s code to send a result back.
* Then both the server and the client need to proceed through their code.

To start I created a host connection from the server.



This opens the server, currently to localhost for testing purposes, then loops and handles multiple clients by creating a thread for each client, then waits for the thread to finish and saves the database.

The handle\_client\_login subroutine splits up the request from the client and runs the login or register method depending on what is needed and sends back true or false. It returns true if the client logged in with correct the username and password.

I tested my program using a small script

Text

Description automatically generated

This piece of code connected and worked perfectly.

I believe some of the code will need to be moved into a main.py file so that it can be incorporated into the project properly and can easily send the team data needed

This week I was able to complete the server side login and test the client side compatibility

## Week 8 – Create User’s Team Database

Last week I completed the login page but it had no data to send to the client so this week I would like to set up and send and receive team data for each user’s team.

To do this I need to:

* Create a database
* Store each user’s team and 5 players
* After login send the data to the client
* Allow addition of new data via client in the future

I started by creating a sqlite3 database

This creates a database that stores the team and driver data of each user

Now I need to link the different files together, I started with the points recorder



This imports the required modules and then runs the required ones to make the final sorted points files

Now I need to link the login code, so I had to move and adapt some subroutines from network.py into main.py so that the team data could also be sent.

I moved the main() and handle\_client() to the main.py file so that they can easily integrate with the new database for teamdata. This code now will connect to a client, get a login or register request and check or append the database, and if the client logged in with the correct login info it will send the relevant teamdata back to the client.

Now that the different code files are linked so all the files are controlled by main.py. This is needed so that only one Python file needs to be started to launch the server

My file system looks like this, before running the code



After running the main.py file it creates these files



## Week 9 – Add/ Get Records From Team Database

Last week I created the teamData database and allowed users to get their data from it, but currently there is no way to create new records for the database, so I need to create code that will receive the team data from the client and append this data to the database.

To do this I need to:

* Find a way to receive the data from the user
* Create a record for the data in the teamData database with the same userID
* Test that the network connection works and the data is being saved
* Test the new data is being sent after login

First of all I need to find a way of receiving the data which needs to be added to the database. The two options I have are to create a new port and a new connection. or to use the current login connection and add a new code instead of login and register, this method would mean all connections would be done through one port which is easier to manage and understand.

First I added a new task so if the server receives the save\_team request and if the username and password are valid it saves the team and driver’s data with the same userID.



I created a client test that connects to the server and sends a request to save team data,

A screenshot of a computer

Description automatically generated



The first client test should return false as there is already a record with the username of test, the second should return True if the data has been saved to the database



These results show that the code is working as intended

Now there needs to be a way to send the team data to the client. I need to start by creating a subroutine that can get the data from the teamData table and return it.



I created this subroutine in main.py that takes in the userID, gets the related data and returns it.

Now I need to link it to the server connection.



Now after connecting to the server they can request the data if they provide the correct username and password.

I tested this new request from the server with a simple script



This worked and provided the correct data, but only if the username and password are correct.

## Week 10 – Login GUI

Last week I finished the code for the server, so I need to start working on the client. I want to start with the gui for the login as this should be the easiest to make.

To do this I need to:

* Install and set up qt software
* Create a .ui file for the login/ register page
* Link the ui file to Python code

To start I installed the Qt creator software which should allow me to start creating the .ui files for my gui.

Now using this new software I have to create these pages from my designs that can be used by the code.



I used the drag and drop interface to help when creating these pages which allowed a GUI to be made.



I used tabs so that the user can either login or register and used a list format to help keep the elements in line.

I created a text box just above the login and register buttons that can be used to display any errors if they occur.



So using these .ui files I now need to link them to some client code. First of all I created a quick piece of code that will just run and show the GUI how a user would see it but without any functionality.

Graphical user interface, text, application, email

Description automatically generated

Now I have the base code that can run the .ui file I can now start integrating the buttons and text boxes as inputs and then connect to the server.

## Week 11 – Login Functionality

Last week I created the GUI file and launched it in Python, so I need to work on adding use to the buttons and getting the info from the text inputs.

To do this I need to:

* Get the object names
* Link the widgets to the Python code
* Links the login and register buttons to methods
* Those methods need to connect to the server
* And verify the username and password

I started by creating the initialiser that would identify all the GUI objects, and connects the signals and slots to allow widget events to be handled by functions within the code.Text

Description automatically generated

This means that any method can call any object it needs and also if the login or register buttons are pressed it runs the corresponding method.

So I need to create the login\_button\_pressed and the register\_button\_pressed methods which are called when the buttons are pressed



This login\_button\_pressed connects to the server if possible and displays an error message if not,, then gets the username and password from the input fields. It sends a login request to the server which if the result is True then it prints login successful, closes the login GUI and proceeds through the code.

Now I need to create the regiser\_button\_pressed method

This method is run when the register button is pressed. It is very similar to the login\_button\_pressed method as it connects to the server and gets the username and password, but it then sends a register request to the server and if the result equals True then the register window will close and the code will proceed.

Now I need to create the connect\_to\_server function that connects to the server using an ip and port that are in the socket\_client list.



This list is stored in a file called connect.private, this is so that the ip and port can be changed in one file and it will affect any code that connects to the server.

The file just contains a list that is split across multiple lines.



Now the login functionality is completed for the server and client, but may need some touching up in the future.

I created some tests to test the code.

Login, is test numbers 4,5,6

Register, is test numbers 7,8,9

And register and login is test number 10

It completed all the tests which means I can now move on to other parts of the client code.

## Week 12 – Main Window

Last week I finished the login process, so I now need to move on and create the main window GUI and add some functionality.

To do this I need to:

* Create a .ui file for the main menu
* Create a class for the menu
* Link the widgets to the Python code
* Get the users team and display it
* Add some functionality into the buttons

To start off with I need to create the Qt .ui file



I have the design diagram to help show everything I would need and where to position the widgets. 

In Qt creator I created a basic layout which is not quite the same as the design spec but includes all the functionality required.

I created a small piece of code to get the .ui file and run it to show how It will be seen when it’s used.





Currently there are a lot of placeholders in the program, as it would need to request the data from the server to write it into the GUI.

Now I need to create some basic functionality. I will start with setting up the buttons and setting them to direct towards methods that will then need to be created.



I started by creating the how\_to\_play button functionality



I created a small how\_to\_play.ui file that can then be launched when the how\_to\_play button is pressed. I need to incorporate that into my code.

The HTP\_Dialogue\_Box class just loads the .ui file and sets the title. The ok button will close the dialogue box when pressed.

Graphical user interface, text, application

Description automatically generated

This is what the how to play dialogue box looks like. It is shown when the how to play button is pressed.

Now I need to create the “How values are calculated” button functionality.

I started by creating the .ui file in Qt Creator.

Text

Description automatically generatedI then linked the ui file and replaced the placeholder with this image

Which shows how the points are gained/ lost

Then I linked this class to the main program using a small method. 

Using the new code, when the code is run and the how values are calculated button is pressed, it shows this window.

This week I was able to create the main menu and add functionality to the exit, how to play and how values are calculated buttons. I need to now add functionality to the refresh, edit team and recent driver changes buttons, and I need to be able to change the data labels on the gui to show the user the team data.

I created some tests to make sure the features are working as intended, test numbers 11,12,13,14,15

## Week 13 – Edit Team Menu

Last week I started the main window GUI , so I now need to add functionality to the different buttons I want to start with the edit team button, which needs a new menu and functionality for the edit team page.

To do this I need to:

* Create a .ui file for the edit team menu
* Create a class for the menu
* Link the widgets to the Python code
* Get the point data from the server
* Allow the user to create a team
* Calculate the value of each driver
* Send the team data to the server

I started by creating the .ui file for the edit team page, with dropdown boxes to choose each player and the constructor and some labels that can be changed during runtime of the code.



After extracting the .ui file I needed to incorporate this into my code.



This class is used for the edit team page. It starts by loading the .ui file then assigns local variables, finds the combo boxes and buttons, then runs the set\_combo\_box\_data() method.

Now I need to create the set\_combo\_box\_data() method



This gets the current drivers and constructors from return\_current\_drivers\_and\_constructors()

Then loops through each combo box and through each driver and adds it to each combo box with their points using a function called return points, which I will need to create.



This function creates the needed directory if it does not exist. It then checks if the data file is older than a day, if so it will get fresh data by running the create\_driver\_file function. Otherwise, it will read the data and return it.



The create\_driver\_file function gets the data from the server and writes it into the correct file.



The return\_points function checks if the file is present and will create the file if not, otherwise the data is stored as [date of data,[drivers, constructors]], it reads the date and if it is not the same day it will fetch new data.



The create\_points\_file gets the data from the server and stores it in a data file.

Now I need to support the commands that are now being used



I added to the two new commands to the current requests, this means it is very well integrated.

When I first made the code for the client it did not store the data in files and would fetch it every time, this caused large wait times before the window would open, so I made it store a local copy that updates every day to reduce the wait time.

Now that the combo boxes have the correct data in them, I need the money calculation to constantly update to let the user know how much money they have left to build their team.



This method just loops, gets the user chosen data from each combo box then adds all the points together and sets all the labels with the correct data.

Then I used threading to run it in parallel so the user can still use the window.



Now I need to add functionality to the submit button.



It stops the update label loop then gets all the data from each combo box and sends it to the server along with the username and password and closes the window if the data is received.

Now the edit team window has functionality.

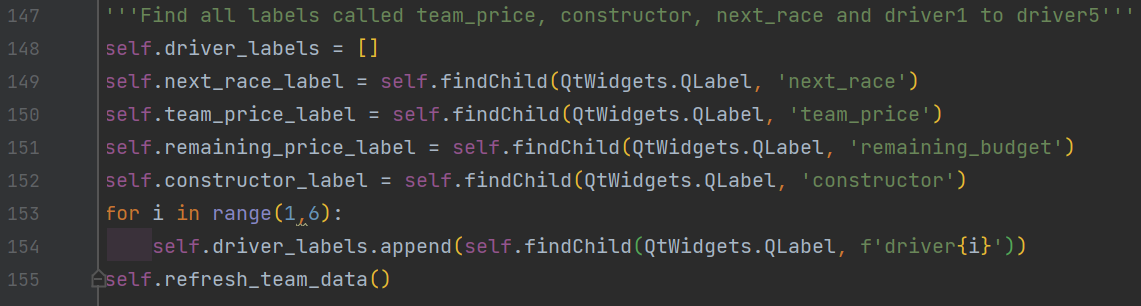
## Week 14 – Main Menu Label Edit

Last week I created the edit team window and added functionality. This week I need to edit the labels on the main menu screen so that it shows the user’s current team stats

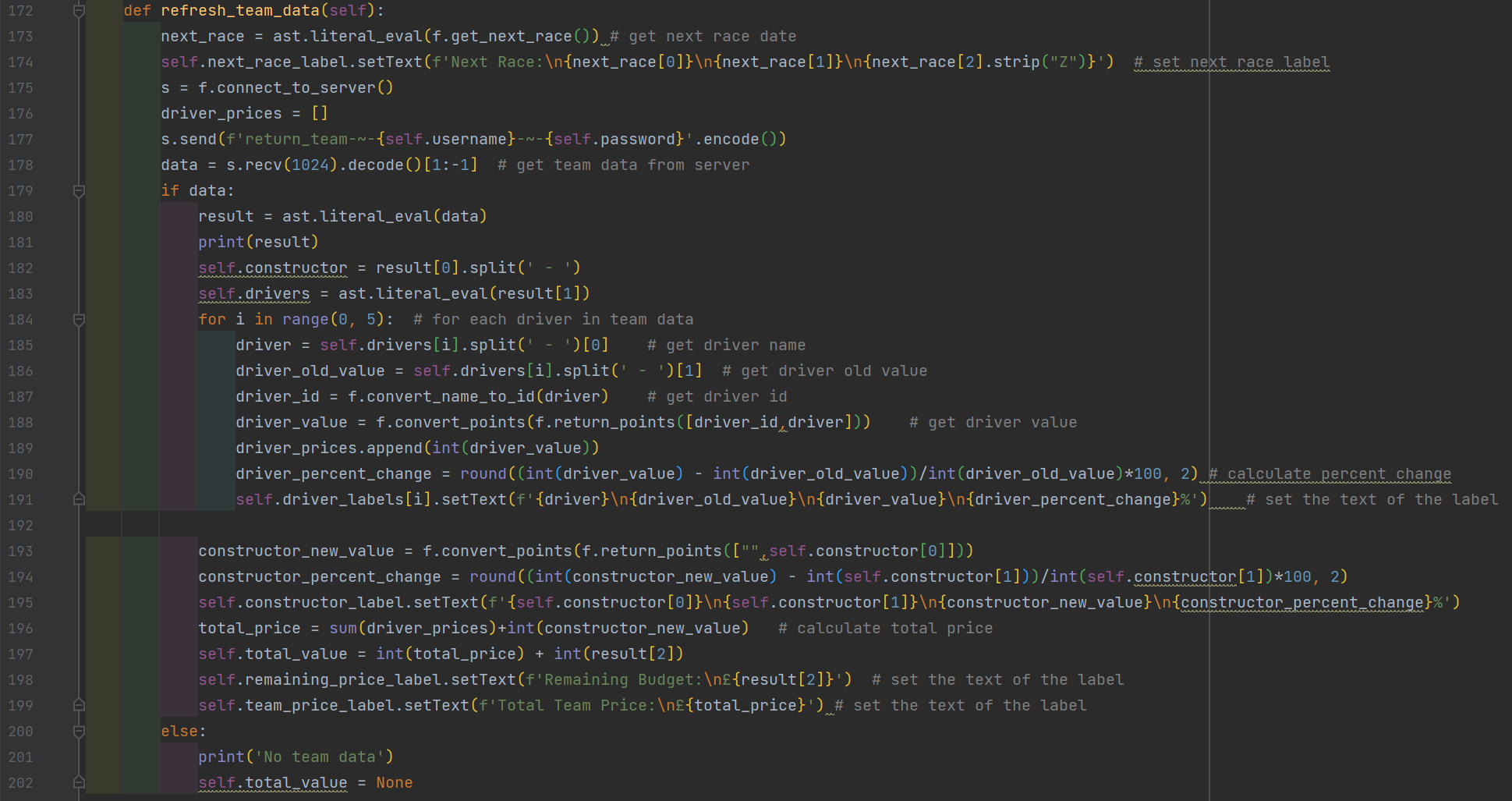
To do this I need to:

* Link labels to Python code
* Get the team data from the server
* Update the correct labels
* Link the refresh data button to the process
* Show the price at which the user bought the driver and the current price
* Show the percentage change of each driver in their team
* Colour the label accordingly

To start with I need to locate the labels and store them in the main menu class



So first all the labels are located and saved as private attributes so that they can be used throughout the class. It then calls the refresh\_team\_data() method which will update all the labels with the correct data.

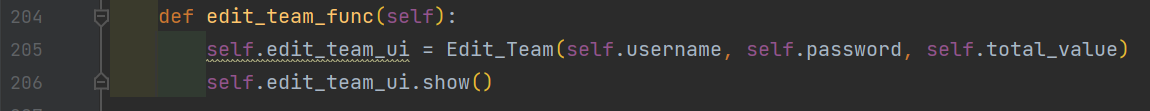


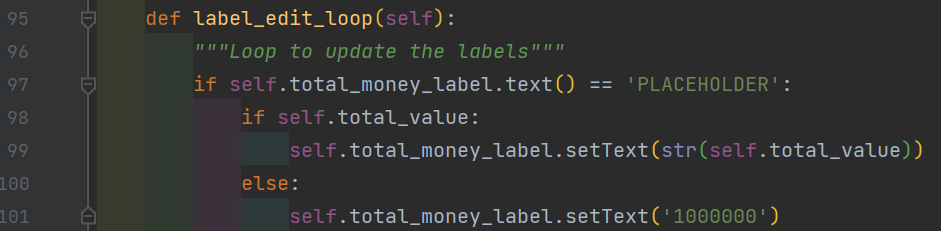
This code first runs the get next race function, then updates the next race label, with the given data.

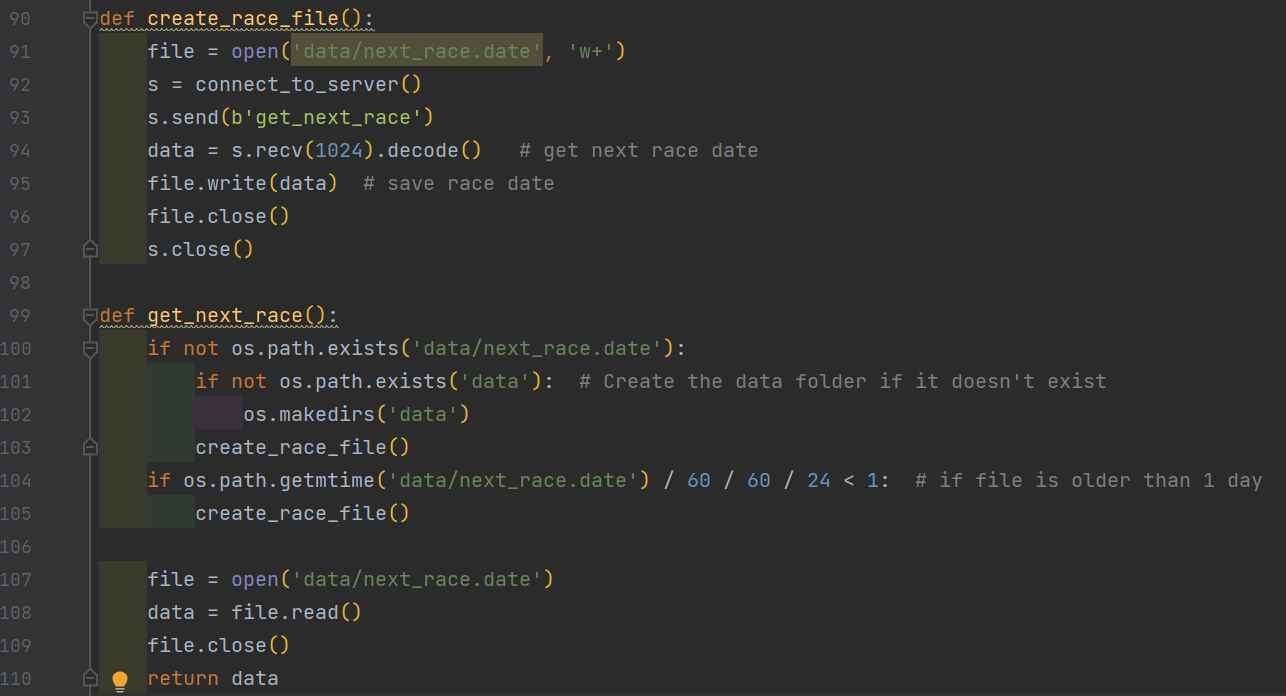
The code then asks the server to return the team, which if there is data will get each driver’s name, the value at which the user picked them, the current value and the percentage change between each price, and then updates the corresponding label.

The code gets the current value of their constructor, the old value, and the percentage change and adds them to the constructor label. It then calculates the total value of the team and gets the remaining budget from the server and adds them to their corresponding labels.

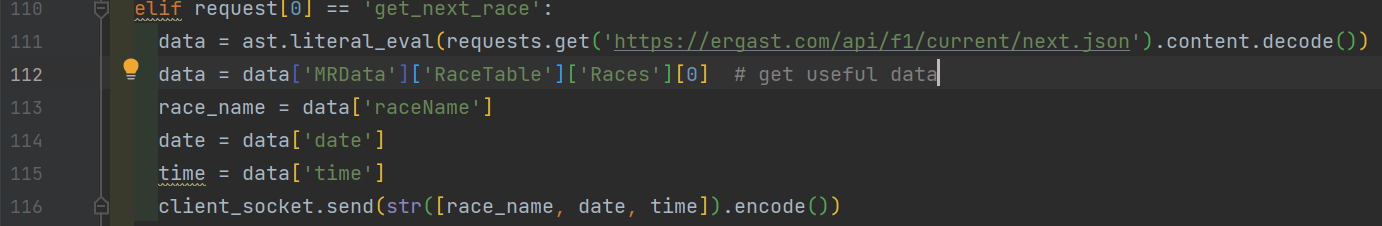
But if no data is present when it is requested then it will just print “no team data” and set the total\_value attribute to None.

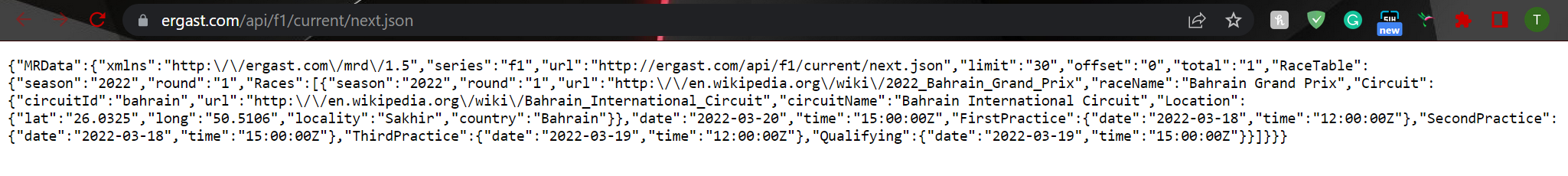


Which is done so that when parsing the data to the edit team window so that it can use that to calculate how much total value they can allocate to the user. So now the user will be able to gain or lose money depending on who they pick.

Now I need to create the get\_next\_race() function so the rest of my code can function. 

The get\_next\_race function is called at the start of the refresh data method, its purpose is just to return the date of the next race when requested. As this date is not going to change very often I made it check if the file was older than a day and if it was it would overwrite the file by requesting for the date again and saving it. To do this, I needed to create a new command on the server.



So now when a client requests the get next race command it gets the data from the api, which returns: This data is then processed to return the name of the next race, the date and time at which it will take place, and it then sends this data to the client.

## Week 15 – Error Handling

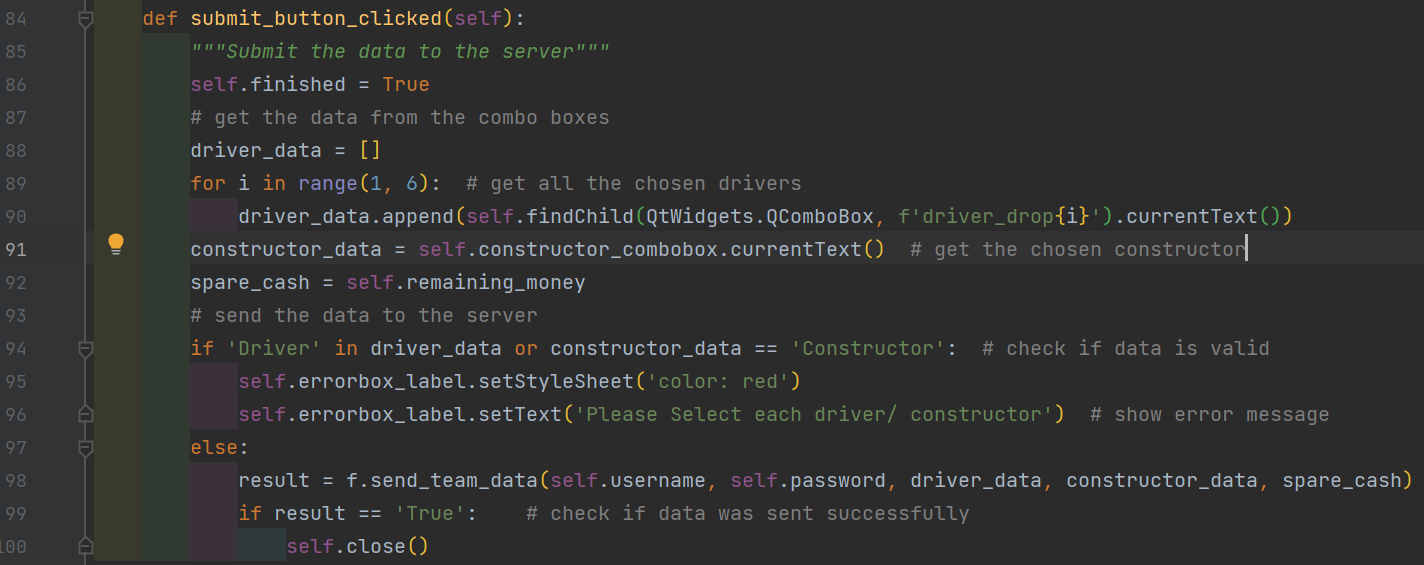
Last week I finished the main menu label edits and parsed and created support for holding the remaining money for each user. This fixed many issues and made my code closer to being finished, but also created a lot of places where the code can error, so this week I aim to prevent these errors or tell the user through the GUI that what they are doing is not supported.

To do this I need to:

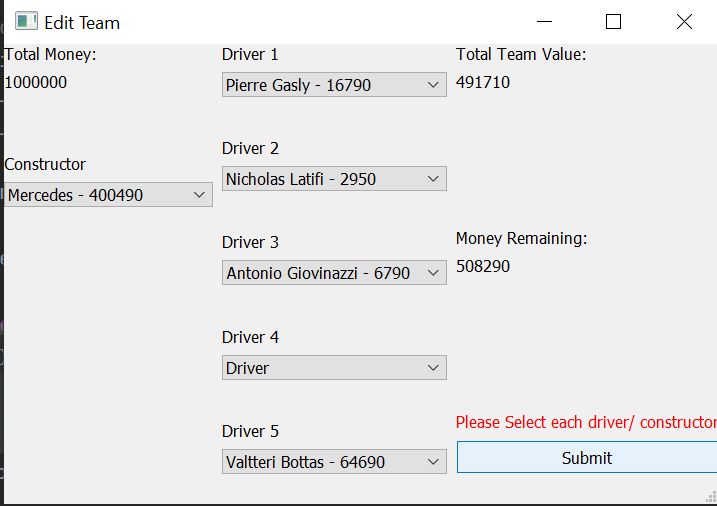
* Find all the places in which the code crashes or behaves as not expected
* Find a fix or workaround
* Implement the fix
* Test each fix

The first error I found was when editing your team if you did not choose a driver or constructor then it would crash whenever you request the data from the server.

This was caused by it trying to split the points and driver/ constructor name and would get an index error. I decided to stop this by just checking if each of the combo boxes have valid data in them when the submit button is pressed.



Now when the submit button is pressed, it will collect all the values from the combo box and if any of the boxes still have Driver or Constructor then it displays a red error message to tell the user to pick each driver and constructor. If all the drivers and constructors are chosen then proceed by sending the data to the server.



So now if you attempt to submit your incomplete team then it prompts with this red error message shown above.

I was testing my project and did not find any crashes or major bugs, so I decided to end this iteration of Error Handling early so that I can start on my client server chat.

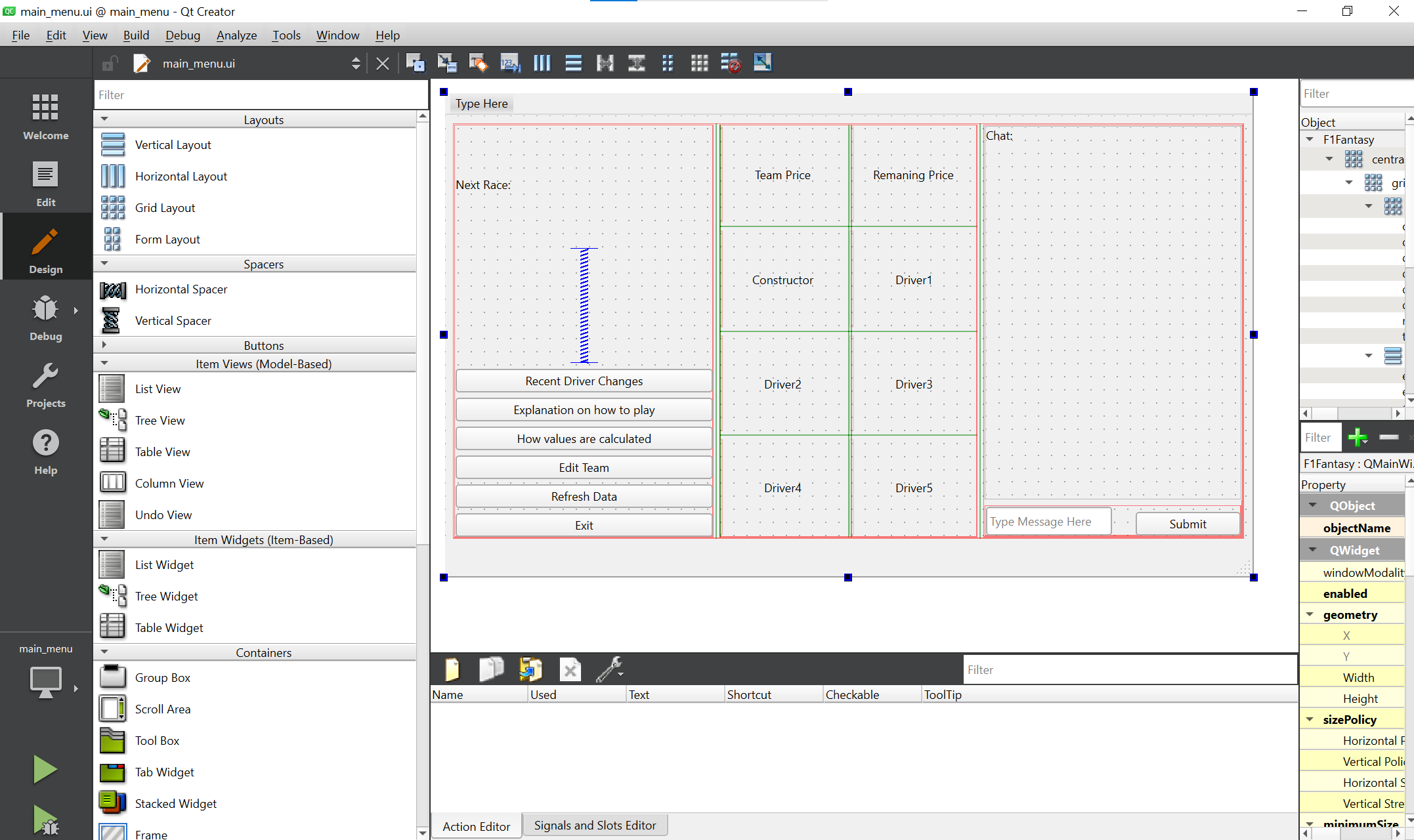
## Week 16 – Chat Room Development

Last week I fixed an error which would cause unexpected crashes, and as no more errors could be found at this stage, I decided to move on to creating the chat room functionality.

To do this I need to:

* Open a new port on the server for the chat
* The server needs to receive get or send requests
* Store send requests so that it can reply to get requests
* Client needs to send a user input to the server
* Client code needs to be linked to the GUI
* The GUI needs to show the chats

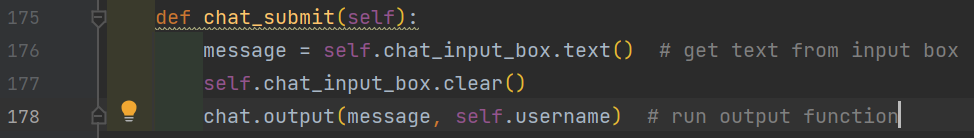
To start with I realised I needed to edit the GUI slightly on the main menu window to add a submit button.



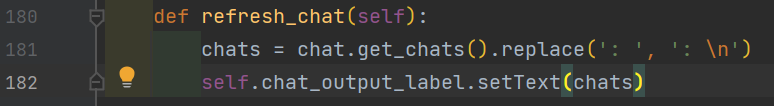
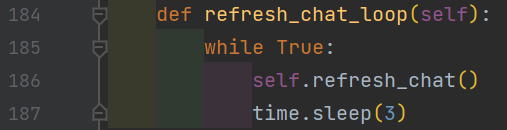
I added a submit button under the chat next to the input box, so that it can be used to send the messages.

I then had to find the chat elements in the code.

This code just finds all the buttons and labels required to run the chat, it then sets it so when the submit button is pressed, the chat\_submit method is called. It also starts a thread for the refresh chat loop so that it can run in parallel.



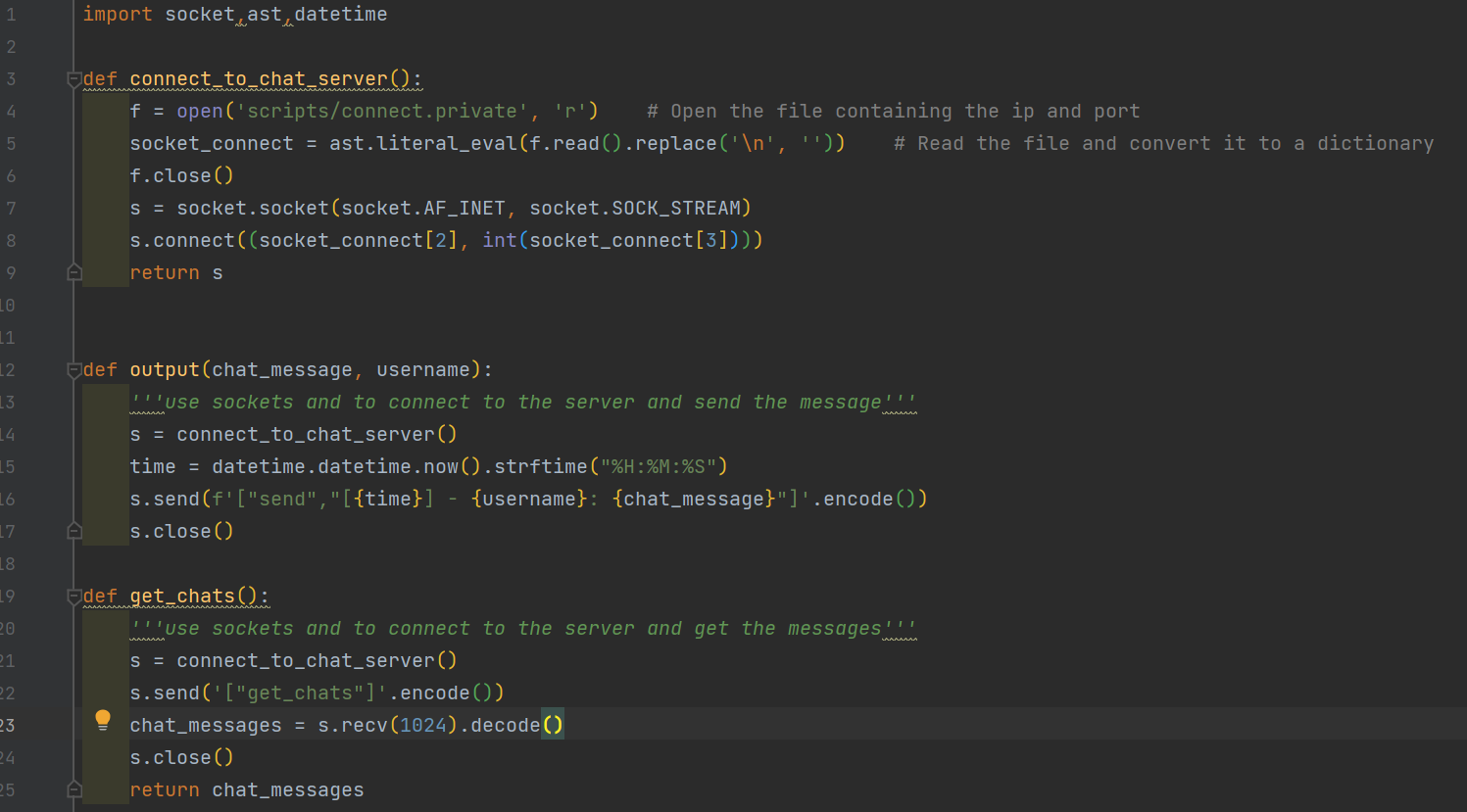
When the chat is submitted it runs this method, gets the message from the input box, clears it and passes the user’s username and the message to the output function.

When the refresh\_chat\_loop is run in parallel, it runs refresh\_chat every 3 seconds, and refresh\_chat gets the chats from the function get\_chats and sets it as the chat text.

I decided to write the needed chat functions in a separate Python file and import them in.



This imports the chat.py Python file and sets it as chat so I can identify when it is being used.

Chat.py:The connect\_to\_chat server function opens the file that includes the required ip and port, and returns the socket connection.

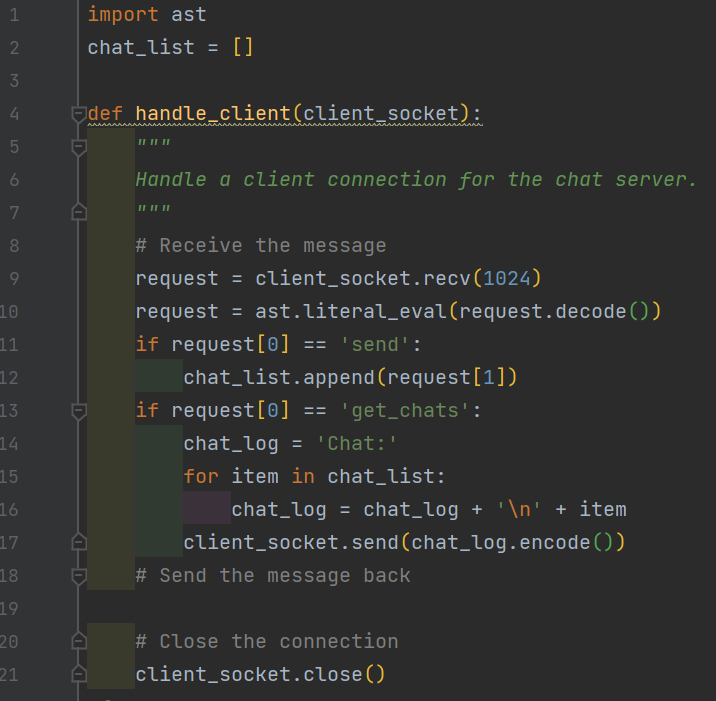
The output function outputs the messages to the server. It connects to the server and sends the request and the message. The message would look like [time] – username: message. this means you know when the message was sent and by whom.

The get\_chats function connects to the server and sends the “get\_chats” request and then receives the response from the server and returns it.

Now I need to code the server to support the new port and new requests.

I decided to make this in a new Python script to not clutter the main.py, so I created chat\_server.py.

chat\_server.py:



The handle\_client function takes in the client socket, receives the request and decodes it. It then checks what the request is. If the request is “send” the function saves the message to chat\_list, and if the request is “get\_chats” then it creates a string that starts with chat: and adds each message then sends that to the client. When it is finished it closes the connection.

Now I need it to run in parallel with the main server code.



The chat server function just binds to the required port and ip waits for any connections and if a client connects it creates a thread for each client that runs the handle client function.



So now when the server code is run it runs the chat server function as a thread so it can run in parallel.

I have now finished the chat code and it works, it works how you would expect, and runs pretty smoothly.

# Evaluation Testing

These tests where done after the last iteration of development to see if I had met all the required criteria.

I recorded me testing and running the program and using it as intended.



## Point Tests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Test | Input | Expected Output | Actual Output | Video/ Photo Reference |
| P1 | Driver Points | Check the points of driver ‘Sebastian Vettel’ | 5244 | 5244 |  |
| P2 | Drivers points | Check the points of driver ‘Daniel Riccardo’ | 2745 | 2745 |  |
| P3 | Constructor Points | Check the points of constructor ‘Aston Martin’ | 813 | 813 |  |

## Login Tests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Test | Input | Expected Output | Actual Output | Video/ Photo Reference |
| L1 | Login | Username = test  Password = test | True | True |  |
| L2 | Login | Username = test9  Password = test | False | False |  |
| L3 | Login | Username = test  Password = test9 | False | False |  |
| L4 | Register | Username = test  Password = test | False | False |  |
| L5 | Register | Username = test5  Password = test | True | True |  |
| L6 | Register, then login | Username = testing  Password = test | True, then True | True, then True |  |

## Main Menu Tests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Test | Input | Expected Output | Actual Output | Video/ Photo Reference |
| M1 | Main menu | Login | The main menu GUI to open | True |  |
| M2 | How to Play | Press how to play button | The How to play dialog box to open | True |  |
| M3 | How to Play Ok button | Press Ok Button | Close the dialog box | True |  |
| M4 | Points Value Calculation Box | Press how points values are calculated button | The Points value calculation dialog box to open | True |  |
| M5 | Points Value Calculation ok button | Press Ok Button | Close the dialog box | True |  |

## Edit Team Menu Tests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Test | Input | Expected Output | Actual Output | Video/ Photo Reference |
| E1 | Empty fields | Select none of the drivers or constructors and press submit | Error message displayed | Error message displayed |  |
| E2 | Half empty fields | Select half of the drivers and select the constructor and leave the their half, then press submit | Error message displayed | Error message displayed |  |
| E3 | no empty fields | Select data for all combobox’s and press submit | Window closes | Window closes |  |
|  |  |  |  |  |  |

## Points to Money Conversion Tests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Test | Input | Expected Output | Actual Output | Video/ Photo Reference |
| O1 | Money conversion for ‘Daniel Riccardo’ | 2745\*20+250=55150 | 55150 | 55150 |  |
| O2 | Money conversion for ‘Mercedes’ | 21492\*20+250=430090 | 430090 | 430090 |  |

## Next Race Tests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Test | Input | Expected Output | Actual Output | Video/ Photo Reference |
| N1 | Correct race | Check next race name and time on 15/03/2022 | Saudi Arabia, 27/03/2022 at 17:00 | Saudi Arabia, 27/03/2022 at 17:00 |  |
| N2 | Correct race | Check next race name and time on 03/04/2022 | Australia, 10/04/2022 at 5:00 | Australia, 10/04/2022 at 5:00 |  |
|  |  |  |  |  |  |

## Chatroom Tests

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No | Test | Input | Expected Output | Actual Output | Video/ Photo Reference |
| C1 | Sending and viewing a message | “Hello world” | Hello world to show in chat | Hello world was shown in chat |  |
| C2 | Sending special characters | “hi, #$%@+” | “hi, #$%@+” | “hi, #$%@+” |  |
| C3 | Viewing a message on a different client | “Hello world” sent from one client, then the other sends “hi” back | Messages on both clients | Both messages visible on both clients |  |

## Usability Testing

To test my project was easy to use and requirement that it was easy to navigate was successfully implemented, I asked a stake holder that had not previously used the system to:

* Create an account and log in
* Create a team
* Refresh their team data
* Send a message using the chat system

While they used the system I asked them to speak out their thoughts of the useability of the system, and how easy they found doing the actions where. I captured the feedback here:

* “The login screen is other programs, so I know how to navigate through it”
* “Very simple to create a new account”
* “I’m not sure how I would change my password if I needed to”
* “It was good to see how much money I had left as I added drivers to my team”
* “As there was no order to the list of drivers it was hard to find Daniel Riccardo”
* “Easy to tell how to refresh my team”
* “It was easy to know where to go for each of the features, because the button names made sense”
* “The chat looks normal and works well”
* “I noticed you cant press enter to send the message, but pressing the button works fine”
* “I would be nice if I could send some emoji’s”

Overall, the stakeholder liked my project and they mentioned how easy the GUI was to navigate. They noticed there was no ‘forgot password’ button they said creating their team was easy but found it slightly annoying to find the driver they wanted in the list as it is not ordered. They liked the chat and how it looked but noticed that if you press enter it does not send the message like many other chats do, and that they couldn’t use any emoji’s. Many of these issues can be solved easily in the future.

# Evaluation

## Stakeholder feedback

Following the directed usability testing I gave my project to my stakeholders and asked for any feedback. I listed their responses and an explanation attached.

James – “I wish the program menus looked a bit better, but I was able to navigate through it well”

The benefit of my code using .ui files for my GUI is that I can just switch out the file and change the GUI without any change to the code. This means in the future it will be very easy to create a nicer looking user interface.

Chris – “The amount each driver was worth seems a bit skewed, as there is a large difference in prices”

Due to the code being heavily bias on drivers that have been driving for longer, as it is all based on the amount of races. However this can be changed with tweaks to the points distribution, meaning you do not need to change any of the code.

Sam – “I actually like the fantasy f1, there are a couple bugs but I’m sure you would be able to sort them”

This is good as it shows the stakeholders are invested in the product and I will be able to fix and sort any issues with the code.

This feedback from my stakeholders can be used to improve the program in the future and shows that we meet the requirements set to begin with.

## Overall Evaluation

Throughout my project I have learnt lots of new skills such as how to create and implement a Python GUI and how to create and manage an SQL Database, and how to get data from an API.

## Maintainability and Future of program

I have made the code with lots of comments and spacing to make the code easy to follow by anyone with a decent understanding of Python. As well as commenting within the code I have also named my variables well to ensure it is easy to understand what each variable is intended for without having to search out its meaning within the code.

Throughout the development of the code I committed it regularly to GitHub, so if anyone wanted to know how it was coded, or how the code evolved over time they could check the commit history. GitHub also provides great tools to allow others to clone the code and either develop it themselves or make contributions back to my repository as pull requests.

The design of the code separates the client and the server components, which greatly simplifies what each aspect of the code deals with, making it both easier to understand, test and modify in the future. The client focuses on the Graphical user interface and communicating with the server. While the server handles the more complicated scoring logic, gathering data from other APIs, storing of the user’s data, and communication between users.

### Complexity

To help with maintainability I should record the Big O complexity of my code to see if it could be improved.

Server side points calculation is O(n2), this is due to nested loops used to get each race then each driver

### Dependencies and Dependency Management

Using external libraries and the Qt framework gave me a great improvement in development speed and allowed this project to be attempted, which would otherwise not be possible for a single person. However, it is important to remember that libraries and frameworks I have used will continue to change, and need to be updated to support bug and security fixes as well as new features. I could stick with the same versions of the modules forever, but that will miss out on any important fixes and older versions may not be supported on newer operating systems. That means that some effort will be needed in the future to update those libraries and handle any incompatibility between versions.

The code uses the following modules and libraries:

* Python Standard Library (v3.8)
  + requests
  + queue
  + datetime
  + os
  + socket
  + ast
  + time
  + threading
  + sqllite3
  + json
* PyQt5
  + QtGui
  + QtWidgets
  + uic
* argon2
  + PasswordHasher

As well as dependencies used within the code this project depends on the source of data in order to be able to gather race results and the next race information. Should this stop functioning then an alternative would need to be found, and adapter code written if the data formats were different. This code depends on the [Ergast Developer API](http://ergast.com/mrd/).

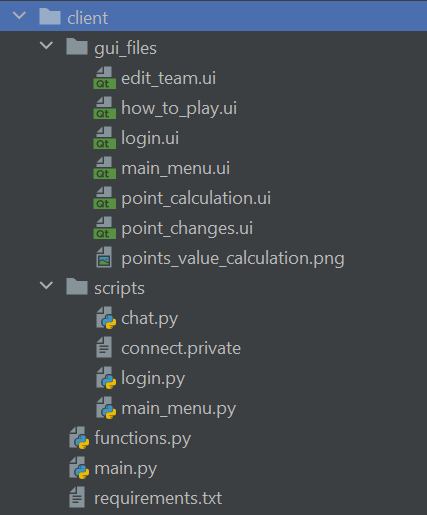
### Future of my program

In the future I would like to work on the GUI and make it more user friendly and more appealing to use. I have also been thinking I may convert the Python program into a website, this would mean it would be very easy to access for users and would help broaden my audience. I would also like to work on the points system to make it more balanced.

|  |  |  |
| --- | --- | --- |
| Future Improvement | Source | Comment |
| Improve balancing of the driver pricing | Stakeholder feedback | Currently the pricing being is biassed on drivers that have been driving for longer. This can be changed with tweaks to the points distribution, meaning you do not need to change any of the code.  I will need to trial a few different approaches to the points distribution to find one that provides better balance. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Final Code

## Client



### main.py

from scripts import login as lg  
from scripts import main\_menu as mm  
def main*()*:  
 result = lg.login*()* # get login result  
 if result:  
 print*(*'logged in'*)* mm.main*(*result*)* # call main menu  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main*()*

### functions.py

import datetime  
import os,socket, ast  
  
  
def connect\_to\_server*()*:  
 f = open*(*'scripts/connect.private', 'r'*)* # Open the file containing the ip and port  
 socket\_connect = ast.literal\_eval*(*f.read*()*.replace*(*'\n', ''*))* # Read the file and convert it to a dictionary  
 f.close*()* s = socket.socket*(*socket.AF\_INET, socket.SOCK\_STREAM*)* s.connect*((*socket\_connect*[*0*]*, int*(*socket\_connect*[*1*])))* return s  
  
  
def create\_driver\_file*()*:  
 # Get data from server and Create the file containing the drivers and constructors  
 s = connect\_to\_server*()* s.send*(*b'return\_current\_driver\_and\_constructors'*)* data = ast.literal\_eval*(*s.recv*(*10028*)*.decode*())* f = open*(*'data/current\_drivers\_and\_constructors.json', 'w+'*)* f.write*(*str*(*data*))* f.close*()*def return\_current\_drivers\_and\_constructors*()*:  
 if not os.path.exists*(*'data/current\_drivers\_and\_constructors.json'*)*:  
 if not os.path.exists*(*'data'*)*: # Create the data folder if it doesn't exist  
 os.makedirs*(*'data'*)* create\_driver\_file*()* if os.path.getmtime*(*'data/current\_drivers\_and\_constructors.json'*)*/60/60/24 < 1: # if the file is older than 1 day  
 create\_driver\_file*()* f = open*(*'data/current\_drivers\_and\_constructors.json', 'r'*)* data = f.read*()* # get data from file  
 if data == '[]' or data == None:  
 f.close*()* create\_driver\_file*()* f = open*(*'data/current\_drivers\_and\_constructors.json', 'r'*)* f.close*()* return ast.literal\_eval*(*data*)*def create\_points\_file*()*:  
 # Get data from server and create the file containing the point data  
 f = open*(*'data/points.json', 'w+'*)* s = connect\_to\_server*()* s.send*(*b'return\_point\_data'*)* point\_data = s.recv*(*100080*)*.decode*()* f.write*(*f'["*{*datetime.date.today*()}*",*{*point\_data*}*]'*)* f.close*()*def return\_points*(*driver*)*:  
 if not os.path.exists*(*'data/points.json'*)*:  
 if not os.path.exists*(*'data'*)*:  
 os.makedirs*(*'data'*)* create\_points\_file*()* f = open*(*'data/points.json'*)* data = ast.literal\_eval*(*f.read*())* if data*[*0*]* != str*(*datetime.date.today*())* or data*[*1*]* == '[]': # if the file is older than 1 day or empty  
 f.close*()* create\_points\_file*()* f.close*()* try:  
 return data*[*1*][*1*][*driver*[*0*]]* # try driver in driver list  
 except KeyError:  
 try:  
 return data*[*1*][*0*][*driver*[*1*]]* # try constructor in constructor list  
 except KeyError:  
 print*(*driver, 'Driver/ Constructor not in point data'*)*def send\_team\_data*(*username, password, driver\_data, constructor\_data, spare\_cash*)*:  
 s = connect\_to\_server*()* s.send*(*f'save\_team-~-*{*username*}*-~-*{*password*}*-~-*{*constructor\_data*}*-~-*{*driver\_data*}*-~-*{*spare\_cash*}*'.encode*())* result = s.recv*(*1024*)*.decode*()* return result  
  
def convert\_name\_to\_id*(*name*)*:  
 *'''use the current\_drivers\_and\_constructors.json file to convert the name[1] to the id[0]'''* data = return\_current\_drivers\_and\_constructors*()* for driver in ast.literal\_eval*(*data*[*0*])*:  
 if driver*[*1*]* == name:  
 return driver*[*0*]* for constructor in ast.literal\_eval*(*data*[*1*])*:  
 if constructor*[*1*]* == name:  
 return constructor*[*0*]*def convert\_points*(*points*)*:  
 return str*(*250 + int*(*points*)*\*20*)*def create\_race\_file*()*:  
 file = open*(*'data/next\_race.date', 'w+'*)* s = connect\_to\_server*()* s.send*(*b'get\_next\_race'*)* data = s.recv*(*1024*)*.decode*()* # get next race date  
 file.write*(*data*)* # save race date  
 file.close*()* s.close*()*def get\_next\_race*()*:  
 if not os.path.exists*(*'data/next\_race.date'*)*:  
 if not os.path.exists*(*'data'*)*: # Create the data folder if it doesn't exist  
 os.makedirs*(*'data'*)* create\_race\_file*()* if os.path.getmtime*(*'data/next\_race.date'*)* / 60 / 60 / 24 < 1: # if file is older than 1 day  
 create\_race\_file*()* file = open*(*'data/next\_race.date'*)* data = file.read*()* if data == "['N', '/', 'A']": # if no data saved, request again  
 create\_race\_file*()* file = open*(*'data/next\_race.date'*)* data = file.read*()* file.close*()* return data

### scripts/login.py

import ast  
import sys  
import functions as f  
from PyQt5 import QtWidgets, uic  
  
class LoginUser:  
 def \_\_init\_\_*(*self, username, password*)*:  
 self.username = username  
 self.password = password  
  
 def get\_username*(*self*)*:  
 return self.username  
  
 def get\_password*(*self*)*:  
 return self.password  
  
class Ui*(*QtWidgets.QMainWindow*)*:  
 def \_\_init\_\_*(*self*)*:  
 super*(*Ui, self*)*.\_\_init\_\_*()* # Call the inherited classes \_\_init\_\_ method  
 uic.loadUi*(*'gui\_files/login.ui', self*)* # Load the .ui file  
 self.show*()* # Show the GUI  
 self.logged\_in = False # Set logged\_in to False  
 self.logged\_in\_user = None  
  
 """Find all Login Widgets"""  
 self.login\_button = self.findChild*(*QtWidgets.QPushButton, 'login\_button'*)* self.login\_username = self.findChild*(*QtWidgets.QLineEdit, 'login\_username'*)* self.login\_password = self.findChild*(*QtWidgets.QLineEdit, 'login\_password'*)* self.login\_errorbox = self.findChild*(*QtWidgets.QLabel, 'login\_errorbox'*)* self.login\_errorbox.setStyleSheet*(*'color: red'*)* # Set the errorbox to red  
 self.login\_button.clicked.connect*(*self.login\_button\_pressed*)* # On click, call the login\_button\_pressed function  
  
 """Find all Register Widgets"""  
 self.register\_button = self.findChild*(*QtWidgets.QPushButton, 'register\_button'*)* self.register\_username = self.findChild*(*QtWidgets.QLineEdit, 'register\_username'*)* self.register\_password = self.findChild*(*QtWidgets.QLineEdit, 'register\_password'*)* self.register\_errorbox = self.findChild*(*QtWidgets.QLabel, 'register\_errorbox'*)* self.register\_errorbox.setStyleSheet*(*'color: red'*)* # Set the errorbox to red  
 self.register\_button.clicked.connect*(*self.register\_button\_pressed*)* # On click, call register\_button\_pressed  
  
 def login\_button\_pressed*(*self*)*:  
 *"""Check if server is online"""* try:  
 s = f.connect\_to\_server*()* except Exception:  
 self.login\_errorbox.setText*(*'Could not connect to server'*)* return  
  
 username = self.login\_username.text*()* # Get the username from the login\_username widget  
 password = self.login\_password.text*()* # Get the password from the login\_password widget  
 s.send*(*f'login-~-*{*username*}*-~-*{*password*}*'.encode*())* result = str*(*s.recv*(*1024*)*.decode*())* # Get the result from the server  
 if result == 'True':  
 print*(*'Login successful'*)* self.logged\_in = True  
 self.logged\_in\_user = LoginUser*(*username, password*)* self.close*()* # If Login successful close the login window  
 elif not ast.literal\_eval*(*result*)[*0*]*:  
 self.login\_errorbox.setText*(*'Username or password incorrect'*)* else:  
 self.login\_errorbox.setText*(*'Login failed'*)* # If Login failed show error message  
 s.close*()* def register\_button\_pressed*(*self*)*:  
 *"""Check if server is online"""* try:  
 s = f.connect\_to\_server*()* except Exception:  
 self.register\_errorbox.setText*(*'Could not connect to server'*)* return  
  
 username = self.register\_username.text*()* # Get the username from the register\_username widget  
 password = self.register\_password.text*()* # Get the password from the register\_password widget  
  
  
 if '-~-' in *(*username or password*)*:  
 self.register\_errorbox.setText*(*'-~- Cannot be used'*)* return  
   
 print*(*f'Register pressed: Username - *{*username*}*, Password - *{*password*}*'*)* s.send*(*f'register-~-*{*username*}*-~-*{*password*}*'.encode*())* result = s.recv*(*1024*)*.decode*()* # Get the result from the server  
 print*(*result*)* if str*(*result*)* == 'True': # If the register is True  
 print*(*'Register successful'*)* self.register\_errorbox.setStyleSheet*(*'color: green'*)* # Set the errorbox colour to green  
 self.register\_errorbox.setText*(*'Register successful'*)* # Show the register successful message  
 elif not ast.literal\_eval*(*result*)[*0*]*:  
 self.register\_errorbox.setText*(*'Username already exists'*)* # If the username already exists  
 else:  
 self.register\_errorbox.setText*(*'Register failed'*)* # If register is False, show error message  
  
 def get\_logged\_in*(*self*)*:  
 *"""Get the logged\_in value"""* return self.logged\_in  
  
 def get\_logged\_in\_user*(*self*)*:  
 *"""Get the logged\_in\_user value"""* return self.logged\_in\_user  
  
def login*()*:  
 app = QtWidgets.QApplication*(*sys.argv*)* # Create an instance of QtWidgets.QApplication  
 window = Ui*()* # Create an instance of our class  
 app.exec\_*()* # Start the application  
 if window.get\_logged\_in*()*: # Return the logged\_in value  
 return window.get\_logged\_in\_user*()* else:  
 print*(*'Login failed'*)*

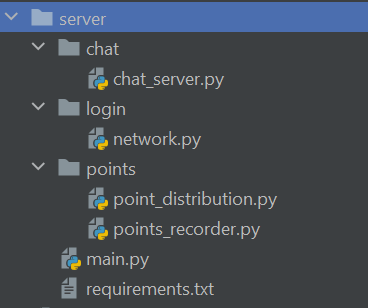
### scripts/main\_menu.py

from PyQt5 import QtWidgets, uic  
from PyQt5.QtGui import QIcon, QPixmap  
import sys  
import ast  
import time  
import threading  
import functions as f  
from scripts.login import LoginUser  
import scripts.chat as chat  
  
class Recent\_Point\_Changes*(*QtWidgets.QDialog*)*:  
 *"""Display the point changes dialog box"""* def \_\_init\_\_*(*self*)*:  
 super*(*Recent\_Point\_Changes, self*)*.\_\_init\_\_*()* uic.loadUi*(*'gui\_files/point\_changes.ui', self*)* self.setWindowTitle*(*'Recent Driver and Constructor Point Changes'*)* # set title  
 self.show*()* self.ok\_button = self.findChild*(*QtWidgets.QPushButton, 'ok\_button'*)* self.ok\_button.clicked.connect*(*self.close*)* # close dialogue box when ok is clicked  
  
class HTP\_Dialogue\_Box*(*QtWidgets.QDialog*)*:  
 *"""Display the how to play dialog box"""* def \_\_init\_\_*(*self*)*:  
 super*(*HTP\_Dialogue\_Box, self*)*.\_\_init\_\_*()* uic.loadUi*(*'gui\_files/how\_to\_play.ui', self*)* self.setWindowTitle*(*'How to Play'*)* # set title  
 self.show*()* self.ok\_button = self.findChild*(*QtWidgets.QPushButton, 'ok\_button'*)* self.ok\_button.clicked.connect*(*self.close*)* # close dialogue box when ok is clicked  
  
class Point\_Calculation\_Dialogue\_Box*(*QtWidgets.QDialog*)*:  
 *"""Display the point calculation dialogue box"""* def \_\_init\_\_*(*self*)*:  
 super*(*Point\_Calculation\_Dialogue\_Box, self*)*.\_\_init\_\_*()* uic.loadUi*(*'gui\_files/point\_calculation.ui', self*)* self.setWindowTitle*(*'Point Calculation'*)* # set title  
 label = self.findChild*(*QtWidgets.QLabel, 'point\_calculation\_pixelmap'*)* pixmap = QPixmap*(*'gui\_files/points\_value\_calculation.png'*)* label.setPixmap*(*pixmap*)* self.show*()* self.ok\_button = self.findChild*(*QtWidgets.QPushButton, 'ok\_button'*)* self.ok\_button.clicked.connect*(*self.close*)* # close dialogue box when ok is clicked  
  
class Edit\_Team*(*QtWidgets.QMainWindow*)*:  
 def \_\_init\_\_*(*self, username, password, total\_value*)*:  
 super*(*Edit\_Team, self*)*.\_\_init\_\_*()* uic.loadUi*(*'gui\_files/edit\_team.ui', self*)* self.username = username  
 self.password = password  
 self.total\_value = total\_value  
 self.finished = False  
 self.setWindowTitle*(*'Edit Team'*)* # set title  
 '''find combo boxes and buttons'''  
 self.constructor\_combobox = self.findChild*(*QtWidgets.QComboBox, 'constructor\_drop'*)* self.submit\_button = self.findChild*(*QtWidgets.QPushButton, 'submit\_button'*)* self.total\_team\_value\_label = self.findChild*(*QtWidgets.QLabel, 'total\_team\_value'*)* self.remaining\_money\_label = self.findChild*(*QtWidgets.QLabel, 'remaining\_money'*)* self.total\_money\_label = self.findChild*(*QtWidgets.QLabel, 'total\_money'*)* self.errorbox\_label = self.findChild*(*QtWidgets.QLabel, 'errorbox'*)* self.set\_combo\_box\_data*()* # set driver and constructor combobox items  
 self.submit\_button.clicked.connect*(*self.submit\_button\_clicked*)* threading.Thread*(*target=self.label\_edit\_loop*)*.start*()* self.show*()* def set\_combo\_box\_data*(*self*)*:  
 *"""Set the data for the combo box"""* data = f.return\_current\_drivers\_and\_constructors*()* # get data from server in form [[driver1,driver2],[team1,team2]]  
 drivers = data*[*0*]* constructors = data*[*1*]* # loop through the driver combo boxes  
 for i in range*(*1, 6*)*:  
 self.driver\_combo\_box = self.findChild*(*QtWidgets.QComboBox, f'driver\_drop*{*i*}*'*)* for driver in ast.literal\_eval*(*drivers*)*:  
 self.driver\_combo\_box.addItem*(*f'*{*driver*[*1*]}* - *{*f.convert\_points*(*f.return\_points*(*driver*))}*'*)* # add each driver and their points to the combo box  
 for constructor in ast.literal\_eval*(*constructors*)*:  
 self.constructor\_combobox.addItem*(*f'*{*constructor*[*1*]}* - *{*f.convert\_points*(*f.return\_points*(*constructor*))}*'*)* def submit\_button\_clicked*(*self*)*:  
 *"""Submit the data to the server"""* self.finished = True  
 # get the data from the combo boxes  
 driver\_data = *[]* for i in range*(*1, 6*)*: # get all the chosen drivers  
 driver\_data.append*(*self.findChild*(*QtWidgets.QComboBox, f'driver\_drop*{*i*}*'*)*.currentText*())* constructor\_data = self.constructor\_combobox.currentText*()* # get the chosen constructor  
 spare\_cash = self.remaining\_money  
 # send the data to the server  
 if 'Driver' in driver\_data or constructor\_data == 'Constructor': # check if data is valid  
 self.errorbox\_label.setStyleSheet*(*'color: red'*)* self.errorbox\_label.setText*(*'Please Select each driver/ constructor'*)* # show error message  
 else:  
 result = f.send\_team\_data*(*self.username, self.password, driver\_data, constructor\_data, spare\_cash*)* if result == 'True': # check if data was sent successfully  
 self.close*()* def label\_edit\_loop*(*self*)*:  
 *"""Loop to update the labels"""* if self.total\_money\_label.text*()* == 'PLACEHOLDER':  
 if self.total\_value:  
 self.total\_money\_label.setText*(*str*(*self.total\_value*))* else:  
 self.total\_money\_label.setText*(*'1000000'*)* while self.finished == False:  
 time.sleep*(*0.2*)* driver\_data = *[]* for i in range*(*1, 6*)*: # for each combo box  
 try: # if there is data to collect  
 driver\_data.append*(*int*(*self.findChild*(*QtWidgets.QComboBox, f'driver\_drop*{*i*}*'*)*.currentText*()*.split*(*' - '*)[*1*]))* except IndexError:  
 driver\_data.append*(*0*)* try: # if there is data to collect  
 constructor\_data = int*(*self.constructor\_combobox.currentText*()*.split*(*' - '*)[*1*])* except IndexError:  
 constructor\_data = 0  
 total = sum*(*driver\_data*)*+constructor\_data  
 self.remaining\_money = int*(*self.total\_money\_label.text*())* - total  
 self.total\_team\_value\_label.setText*(*str*(*total*))* # set label text  
 self.remaining\_money\_label.setText*(*str*(*self.remaining\_money*))* # set label text  
  
 def closeEvent*(*self, event*)*:  
 self.finished = True  
  
  
  
class Main\_Menu\_Ui*(*QtWidgets.QMainWindow*)*:  
 def \_\_init\_\_*(*self, temp\_username, temp\_password*)*:  
 super*(*Main\_Menu\_Ui, self*)*.\_\_init\_\_*()* # Call the inherited classes \_\_init\_\_ method  
 uic.loadUi*(*'gui\_files/main\_menu.ui', self*)* # Load the .ui file  
 self.show*()* # Show the GUI  
 self.username = temp\_username  
 self.password = temp\_password  
 self.setWindowTitle*(*'Main Menu'*)* """Find all buttons"""  
 self.edit\_team\_button = self.findChild*(*QtWidgets.QPushButton, 'edit\_team'*)* self.refresh\_button = self.findChild*(*QtWidgets.QPushButton, 'refresh'*)* self.how\_to\_play\_button = self.findChild*(*QtWidgets.QPushButton, 'explain\_htp'*)* # explain how to play  
 self.recent\_driver\_changes\_button = self.findChild*(*QtWidgets.QPushButton, 'recent\_driver\_changes'*)* self.how\_val\_calc\_button = self.findChild*(*QtWidgets.QPushButton, 'how\_vac'*)* # how values are calculated  
 self.exit\_button = self.findChild*(*QtWidgets.QPushButton, 'exit'*)* """bind buttons to functions"""  
 self.refresh\_button.clicked.connect*(*self.refresh\_team\_data*)* self.exit\_button.clicked.connect*(*self.sign\_out*)* self.edit\_team\_button.clicked.connect*(*self.edit\_team\_func*)* self.how\_to\_play\_button.clicked.connect*(*self.how\_to\_play*)* self.recent\_driver\_changes\_button.clicked.connect*(*self.recent\_point\_changes*)* self.how\_val\_calc\_button.clicked.connect*(*self.how\_val\_calc*)* '''Find all labels called team\_price, constructor, next\_race and driver1 to driver5'''  
 self.driver\_labels = *[]* self.next\_race\_label = self.findChild*(*QtWidgets.QLabel, 'next\_race'*)* self.team\_price\_label = self.findChild*(*QtWidgets.QLabel, 'team\_price'*)* self.remaining\_price\_label = self.findChild*(*QtWidgets.QLabel, 'remaining\_budget'*)* self.constructor\_label = self.findChild*(*QtWidgets.QLabel, 'constructor'*)* for i in range*(*1,6*)*:  
 self.driver\_labels.append*(*self.findChild*(*QtWidgets.QLabel, f'driver*{*i*}*'*))* self.refresh\_team\_data*()* '''chat attributes'''  
 self.chat\_submit\_button = self.findChild*(*QtWidgets.QPushButton, 'chat\_submit\_button'*)* self.chat\_input\_box = self.findChild*(*QtWidgets.QLineEdit, 'chat\_input'*)* self.chat\_output\_label = self.findChild*(*QtWidgets.QLabel, 'chat\_output'*)* self.chat\_submit\_button.clicked.connect*(*self.chat\_submit*)* threading.Thread*(*target=self.refresh\_chat\_loop*)*.start*()* def chat\_submit*(*self*)*:  
 message = self.chat\_input\_box.text*()* # get text from input box  
 self.chat\_input\_box.clear*()* chat.output*(*message, self.username*)* # run output function  
  
 def refresh\_chat*(*self*)*:  
 chats = chat.get\_chats*()*.replace*(*': ', ': \n'*)* self.chat\_output\_label.setText*(*chats*)* def refresh\_chat\_loop*(*self*)*:  
 # refresh chat every 3 seconds  
 while True:  
 self.refresh\_chat*()* time.sleep*(*3*)* def recent\_point\_changes*(*self*)*:  
 self.recent\_points\_change = Recent\_Point\_Changes*()* self.recent\_points\_change.show*()* # show the window  
  
 def how\_to\_play*(*self*)*:  
 self.htp\_dialogue\_box = HTP\_Dialogue\_Box*()* self.htp\_dialogue\_box.show*()* # show dialogue box  
  
 def how\_val\_calc*(*self*)*:  
 self.point\_calculation\_dialogue\_box = Point\_Calculation\_Dialogue\_Box*()* self.point\_calculation\_dialogue\_box.show*()* # show dialogue box  
  
 def refresh\_team\_data*(*self*)*:  
 next\_race = ast.literal\_eval*(*f.get\_next\_race*())* # get next race date  
 self.next\_race\_label.setText*(*f'Next Race:\n*{*next\_race*[*0*]}*\n*{*next\_race*[*1*]}*\n*{*next\_race*[*2*]*.strip*(*"Z"*)}*'*)* # set next race label  
 s = f.connect\_to\_server*()* driver\_prices = *[]* s.send*(*f'return\_team-~-*{*self.username*}*-~-*{*self.password*}*'.encode*())* data = s.recv*(*1024*)*.decode*()[*1:-1*]* # get team data from server  
 if data: # if the user has team data  
 result = ast.literal\_eval*(*data*)* print*(*result*)* self.constructor = result*[*0*]*.split*(*' - '*)* self.drivers = ast.literal\_eval*(*result*[*1*])* for i in range*(*0, 5*)*: # for each driver in team data  
 driver = self.drivers*[*i*]*.split*(*' - '*)[*0*]* # get driver name  
 driver\_old\_value = self.drivers*[*i*]*.split*(*' - '*)[*1*]* # get driver old value  
 driver\_id = f.convert\_name\_to\_id*(*driver*)* # get driver id  
 driver\_value = f.convert\_points*(*f.return\_points*([*driver\_id,driver*]))* # get driver value  
 driver\_prices.append*(*int*(*driver\_value*))* driver\_percent\_change = round*((*int*(*driver\_value*)* - int*(*driver\_old\_value*))*/int*(*driver\_old\_value*)*\*100, 2*)* # calculate percent change  
 self.driver\_labels*[*i*]*.setText*(*f'*{*driver*}*\n*{*driver\_old\_value*}*\n*{*driver\_value*}*\n*{*driver\_percent\_change*}*%'*)* # set the text of the label  
  
 constructor\_new\_value = f.convert\_points*(*f.return\_points*([*"",self.constructor*[*0*]]))* constructor\_percent\_change = round*((*int*(*constructor\_new\_value*)* - int*(*self.constructor*[*1*]))*/int*(*self.constructor*[*1*])*\*100, 2*)* self.constructor\_label.setText*(*f'*{*self.constructor*[*0*]}*\n*{*self.constructor*[*1*]}*\n*{*constructor\_new\_value*}*\n*{*constructor\_percent\_change*}*%'*)* total\_price = sum*(*driver\_prices*)*+int*(*constructor\_new\_value*)* # calculate total price  
 self.total\_value = int*(*total\_price*)* + int*(*result*[*2*])* self.remaining\_price\_label.setText*(*f'Remaining Budget:\n£*{*result*[*2*]}*'*)* # set the text of the label  
 self.team\_price\_label.setText*(*f'Total Team Price:\n£*{*total\_price*}*'*)* # set the text of the label  
 else:  
 print*(*'No team data'*)* self.total\_value = None  
  
 def edit\_team\_func*(*self*)*:  
 self.edit\_team\_ui = Edit\_Team*(*self.username, self.password, self.total\_value*)* self.edit\_team\_ui.show*()* def sign\_out*(*self*)*:  
 self.close*()* self.destroy*()* sys.exit*()*def main*(*login\_user: LoginUser*)*:  
 app = QtWidgets.QApplication*(*sys.argv*)* # Create an instance of QtWidgets.QApplication  
 window = Main\_Menu\_Ui*(*login\_user.get\_username*()*, login\_user.get\_password*())* # Create an instance of our class  
 app.exec\_*()* # Start the application

### scripts/chat.py

import socket,ast,datetime  
  
def connect\_to\_chat\_server*()*:  
 f = open*(*'scripts/connect.private', 'r'*)* # Open the file containing the ip and port  
 socket\_connect = ast.literal\_eval*(*f.read*()*.replace*(*'\n', ''*))* # Read the file and convert it to a dictionary  
 f.close*()* s = socket.socket*(*socket.AF\_INET, socket.SOCK\_STREAM*)* s.connect*((*socket\_connect*[*2*]*, int*(*socket\_connect*[*3*])))* return s  
  
  
def output*(*chat\_message, username*)*:  
 *'''use sockets and to connect to the server and send the message'''* s = connect\_to\_chat\_server*()* time = datetime.datetime.now*()*.strftime*(*"%H:%M:%S"*)* s.send*(*f'["send","[*{*time*}*] - *{*username*}*: *{*chat\_message*}*"]'.encode*())* s.close*()*def get\_chats*()*:  
 *'''use sockets and to connect to the server and get the messages'''* s = connect\_to\_chat\_server*()* s.send*(*'["get\_chats"]'.encode*())* chat\_messages = s.recv*(*1024*)*.decode*()* s.close*()* return chat\_messages

## Server



### main.py

*"""Create a database for the team data for each user"""*import sqlite3  
import socket  
import threading  
import chat.chat\_server as chat  
  
  
sqliteConnection = sqlite3.connect*(*'teamData.db', check\_same\_thread=False*)*dataCursor = sqliteConnection.cursor*()*dataCursor.execute*(*"""  
CREATE TABLE IF NOT EXISTS   
teamData (userID int NOT NULL,team string,drivers,cash, UNIQUE(userID), PRIMARY KEY (userID));  
"""*)* #Creates table in the format [userID,team,drivers]  
  
  
def create\_driver\_points*()*:  
 from points import points\_recorder as pr  
 pr.get\_drivers*()* pr.get\_race\_data*()* pr.split\_driver\_points*()*from login import network as net  
  
  
def main*()*:  
 *"""Create server connection"""* s = socket.socket*(*socket.AF\_INET, socket.SOCK\_STREAM*)* s.bind*((*'localhost', 9999*))* s.listen*(*5*)* print*(*'Server started'*)* """Create a thread for each client"""  
 while True:  
 client\_socket, address = s.accept*()* print*(*f'Connection from *{*address*}* has been established'*)* client = threading.Thread*(*target=handle\_client, args=*(*client\_socket,*))* # create a thread for each client  
 client.start*()* print*(*f'Thread for *{*address*}* has been created'*)* client.join*()* print*(*f'Thread for *{*address*}* has been joined'*)* sqliteConnection.commit*()* # save changes to the database  
  
  
def handle\_client*(*client\_socket*)*:  
 user = net.handle\_client\_login*(*client\_socket*)* sqliteConnection.commit*()*def save\_team*(*userID, constructor, drivers, spare\_cash*)*:  
 *"""Save the team data to the database"""* """IF DATA EXISTS, UPDATE IT"""  
 dataCursor.execute*(*f'INSERT OR REPLACE INTO teamData (userID, team, drivers, cash) VALUES (*{*userID*}*, "*{*constructor*}*", "*{*drivers*}*", *{*spare\_cash*}*)'*)* sqliteConnection.commit*()* return True  
  
  
def return\_team*(*userID*)*:  
 *"""Return the team data from the database"""* """IF DATA EXISTS, RETURN IT"""  
 data = dataCursor.execute*(*f'SELECT team, drivers, cash FROM teamData WHERE userID = *{*userID*}*'*)*.fetchall*()* return data  
  
def chat\_server*()*:  
 *"""Create server connection"""* s = socket.socket*(*socket.AF\_INET, socket.SOCK\_STREAM*)* s.bind*((*'localhost', 9988*))* s.listen*(*5*)* print*(*'Chat Server started'*)* """Create a thread for each client"""  
 while True:  
 client\_socket, address = s.accept*()* client = threading.Thread*(*target=chat.handle\_client, args=*(*client\_socket,*))* # create a thread for each client  
 client.start*()* client.join*()* print*(*f'Chat Connection from *{*address*}* has been established, started, and joined'*)*if \_\_name\_\_ == '\_\_main\_\_':  
 try:  
 create\_driver\_points*()* pass  
 except SyntaxError:  
 print*(*'API Offline'*)* '''Start chat server'''  
 threading.Thread*(*target=chat\_server*)*.start*()* # create a thread for the chat server  
 '''Start main server'''  
 main*()* sqliteConnection.commit*()* # save changes to the database

### login/network.py

import sqlite3  
import time  
import ast, requests  
from argon2 import PasswordHasher  
import socket  
import threading, json  
from main import save\_team, return\_team  
  
ph = PasswordHasher*()*sqliteConnection = sqlite3.connect*(*'login/logins.db', check\_same\_thread=False*)*cursor = sqliteConnection.cursor*()*cursor.execute*(*"CREATE TABLE IF NOT EXISTS logins (userID ,username string,hashpass string, UNIQUE(userID), PRIMARY KEY (userID));"*)*#cursor.execute(f'INSERT INTO logins VALUES (1,"test", "{ph.hash("teststring")}");')  
print*(*cursor.execute*(*"SELECT \* FROM logins"*)*.fetchall*())*class Account:  
 def \_\_init\_\_*(*self,username*)*:  
 self.username = username  
 try:  
 if str*(*cursor.execute*(*f'SELECT username FROM logins WHERE username="*{*self.username*}*"'*)*.fetchall*()[*0*])*.split*(*"'"*)[*1*]* != None: # if username exists  
 self.account = True  
 self.password\_hash = str*(*cursor.execute*(*f'SELECT hashpass FROM logins WHERE username="*{*username*}*"'*)*.fetchall*()[*0*])*.split*(*"'"*)[*1*]* except IndexError:  
 self.account = False  
  
 def valid\_account*(*self*)*:  
 return self.account  
  
 def check\_pass*(*self, given\_pass*)*:  
 try:  
 if ph.verify*(*self.password\_hash, given\_pass*)*:  
 return True  
 except Exception:  
 return *[*False,'Invalid Password'*]* def create\_record*(*self, password*)*:  
 id = 0  
 while True:  
 try:  
 cursor.execute*(*f'INSERT INTO logins (userID, username, hashpass) VALUES(*{*id*}*,"*{*self.username*}*","*{*password*}*")'*)* print*(*'Account Created'*)* break  
 except Exception:  
 id = id + 1  
  
  
def login*(*account\_username, password*)*:   
 user = Account*(*account\_username*)* if user.valid\_account*()*:  
 result = user.check\_pass*(*password*)* else:  
 result = *[*False,'Invalid Username'*]* return result  
  
def register*(*username, password*)*:  
 user = Account*(*username*)* if user.valid\_account*()* != True:  
 user.create\_record*(*ph.hash*(*password*))* return True  
 else:  
 return *[*False, 'Username Occupied'*]*def handle\_client\_login*(*client\_socket*)*:  
 *"""create a new thread for each client and determine if the client is a new user or returning user so they can login or register"""* request = client\_socket.recv*(*1024*)*.decode*()* request = request.split*(*'-~-'*)* print*(*request*)* if request*[*0*]* == 'register':  
 """request in form [register, username, password]"""  
 result = register*(*request*[*1*]*, request*[*2*])* sqliteConnection.commit*()* # commit changes to database  
 client\_socket.send*(*str*(*result*)*.encode*())* elif request*[*0*]* == 'login':  
 """request in form [login, username, password]"""  
 result = login*(*request*[*1*]*, request*[*2*])* print*(*result*)* client\_socket.send*((*str*(*result*))*.encode*())* if result: # if login is successful  
 return True  
 elif request*[*0*]* == 'save\_team':  
 """request in form [save\_team, username, password, team\_data, driver\_data, spare\_cash]"""  
 if login*(*request*[*1*]*, request*[*2*])*: # if login is successful  
 userID = cursor.execute*(*f'SELECT userID FROM logins WHERE username = "*{*request*[*1*]}*"'*)*.fetchall*()[*0*][*0*]* result = save\_team*(*userID, request*[*3*]*, request*[*4*]*, request*[*5*])* # save team  
 client\_socket.send*(*str*(*result*)*.encode*())* elif request*[*0*]* == 'return\_team':  
 """request in form [return\_team, username, password]"""  
 if (login\_result := login(request[1], request[2])): # if login is successful  
 userID = cursor.execute(f'SELECT userID FROM logins WHERE username = "{request[1]}"').fetchall()[0][0]  
 result = return\_team(userID) # return team  
 else:  
 result = login\_result  
 client\_socket.send*(*str*(*result*)*.encode*())* elif request*[*0*]* == 'return\_point\_data':  
 # get point data from files and send to client  
 team\_points = open*(*'points/team\_points.json'*)* driver\_points = open*(*'points/driver\_points.json'*)* team\_points = json.load*(*team\_points*)* driver\_points = json.load*(*driver\_points*)* client\_socket.send*(*json.dumps*([*team\_points, driver\_points*])*.encode*())* elif request*[*0*]* == 'return\_current\_driver\_and\_constructors':  
 # get current driver and constructor data from files and send to client  
 current\_drivers = open*(*'points/current\_drivers.txt', 'r'*)* current\_constructors = open*(*'points/current\_constructors.txt', 'r'*)* current\_drivers = current\_drivers.read*()* current\_constructors = current\_constructors.read*()* client\_socket.send*(*str*([*current\_drivers, current\_constructors*])*.encode*())* elif request*[*0*]* == 'get\_next\_race':  
 try:  
 data = ast.literal\_eval*(*requests.get*(*'https://ergast.com/api/f1/current/next.json'*)*.content.decode*())* data = data*[*'MRData'*][*'RaceTable'*][*'Races'*][*0*]* # get useful data  
 race\_name = data*[*'raceName'*]* date = data*[*'date'*]* time = data*[*'time'*]* except:  
 race\_name, date, time = 'N/A'  
  
 client\_socket.send*(*str*([*race\_name, date, time*])*.encode*())* else:  
 client\_socket.send*(*b'Invalid Request'*)* return False  
  
  
def main():  
 s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 s.bind(('localhost', 9999))  
 s.listen(5)  
 """create a thread for each client"""  
 while True:  
 client\_socket, address = s.accept()  
 print(f'Connection from {address} has been established')  
 client = threading.Thread(target=handle\_client, args=(client\_socket,))  
 client.start()  
 print(f'Thread for {address} has been created')  
 client.join()  
 sqliteConnection.commit()

### points/points\_recorder.py

import requests #import needed modules  
import ast  
from threading import Thread, Lock  
import queue  
from datetime import date  
global directory  
directory = 'points/'  
lock = Lock*()*def get\_drivers*()*:  
 year = date.today*()*.year  
 driver\_data = requests.get*(*f'https://ergast.com/api/f1/*{*year*}*/drivers.json'*)* constructor\_data = requests.get*(*f'https://ergast.com/api/f1/*{*year*}*/constructors.json'*)* constructors = ast.literal\_eval*(*constructor\_data.content.decode*())[*'MRData'*][*'ConstructorTable'*][*'Constructors'*]* drivers = ast.literal\_eval*(*driver\_data.content.decode*())[*'MRData'*][*'DriverTable'*][*'Drivers'*]* if drivers == *[]* or constructors == *[]*:  
 driver\_data = requests.get*(*f'https://ergast.com/api/f1/*{*year-1*}*/drivers.json'*)* constructor\_data = requests.get*(*f'https://ergast.com/api/f1/*{*year-1*}*/constructors.json'*)* constructors = ast.literal\_eval*(*constructor\_data.content.decode*())[*'MRData'*][*'ConstructorTable'*][*'Constructors'*]* drivers = ast.literal\_eval*(*driver\_data.content.decode*())[*'MRData'*][*'DriverTable'*][*'Drivers'*]* detail\_driver\_list = *[]* detail\_constructor\_list = *[]* for driver in drivers:  
 driver\_id = driver*[*'driverId'*]* full\_name = f"*{*driver*[*'givenName'*]} {*driver*[*'familyName'*]}*"  
 detail\_driver\_list.append*([*driver\_id,full\_name*])*#,nationality,DoB,number])  
 for constructor in constructors:  
 constructor\_id = constructor*[*'constructorId'*]* name = constructor*[*'name'*]* detail\_constructor\_list.append*([*constructor\_id, name*])* driver\_file = open*(*f'*{*directory*}*current\_drivers.txt', 'w+'*)* driver\_file.write*(*str*(*detail\_driver\_list*))* driver\_file.close*()* constructor\_file = open*(*f'*{*directory*}*current\_constructors.txt', 'w+'*)* constructor\_file.write*(*str*(*detail\_constructor\_list*))* constructor\_file.close*()*def save\_to\_file*(*queue*)*:  
 file = open*(*f'*{*directory*}*raw\_points.csv', 'w+'*)* file.write*(*'year,round,race name,team,team points,driver1,driver1 points,driver2,driver2 points\n'*)* #Write titles of data  
 while finished != True or queue.empty*()* != True:  
 """Loop until queue is empty and finished == True"""  
 file.write*(*queue.get*())* file.close*()*def get\_race\_data*()*:  
 global q,finished  
 q = queue.Queue*()* finished = False  
 """use threading to run each year in parallel"""  
 threads=*[]* for year in range*(*1950,date.today*()*.year+1*)*:  
 threads.append*(*Thread*(*target=year\_loop, args=*(*year,*)))* for thread in threads:  
 thread.start*()* for thread in threads: #End all threads  
 thread.join*()* finished = True  
 save\_to\_file*(*q*)* #save the queue to a file  
  
  
def year\_loop*(*year*)*:  
 #for year in range(1950,date.today().year+1): #Cycle through all years of f1 from 1950 to the current year  
 total\_races\_in\_year = ast.literal\_eval*(*requests.get*(*f'http://ergast.com/api/f1/*{*year*}*.json'*)*.content.decode*())[*'MRData'*][*'total'*]* #Get all rounds in the year  
 for race in range*(*1,int*(*total\_races\_in\_year*)*+1*)*: #Cycle through all the rounds in the year  
 race\_data = ast.literal\_eval*(*requests.get*(*f'http://ergast.com/api/f1/*{*year*}*/*{*race*}*/results.json'*)*.content.decode*())* #Get the race data  
 try:  
 assign\_driver\_points*(*race\_data*)* #,quali\_data)  
 except IndexError as a:  
 print*(*f"*{*a*} {*year*}*"*)*def assign\_driver\_points*(*rData*)*: #use race data to assign points to the driver  
 team\_dict = *{}* year = rData*[*'MRData'*][*'RaceTable'*][*'Races'*][*0*][*'season'*]* #Get race year  
 round = rData*[*'MRData'*][*'RaceTable'*][*'Races'*][*0*][*'round'*]* #Get race number  
 rName = rData*[*'MRData'*][*'RaceTable'*][*'Races'*][*0*][*'raceName'*]* #Get race name  
 try:  
 for driver in rData*[*'MRData'*][*'RaceTable'*][*'Races'*][*0*][*'Results'*]*:  
 constructor = driver*[*'Constructor'*][*'name'*]* #Get drivers team  
 driver\_id = driver*[*'Driver'*][*'driverId'*]* #Get drivers name  
 finish\_position = driver*[*'position'*]* #Get finish pos  
 grid\_position = driver*[*'grid'*]* #Get starting position  
 try: fastest\_lap\_position = driver*[*'FastestLap'*][*'rank'*]* #Get fastest lap rank  
 except KeyError: fastest\_lap\_position = 22  
 if constructor not in team\_dict: #Combines all drivers of the same teams in a dictionary  
 team\_dict*[*constructor*]* = *[]* team\_dict*[*constructor*]*.append*({*'driver':driver\_id,'fastPosition':fastest\_lap\_position,'gPosition':grid\_position,'fPosition':finish\_position*})* #make a dictionary which contains all driver data needed and groups drivers by team  
 for team in team\_dict:  
 driver1 = team\_dict*[*team*][*0*]* try: driver2 = team\_dict*[*team*][*1*]* except IndexError: driver2 = *{*'driver': 'n/a', 'fastPosition': 22, 'gPosition': '22', 'fPosition': '22'*}* #If no second driver exists, use filler data  
 points = get\_points*(*driver1,driver2*)* #get the driver and constructors points  
 save\_points*([*driver1*[*'driver'*]*,points*[*'driver1'*]]*,*[*driver2*[*'driver'*]*,points*[*'driver2'*]]*,*[*team,points*[*'constructor'*]]*,year,round,rName*)* except KeyError as a:  
 print*(*a*)*def save\_points*(*driver1,driver2,team,year,round,rName*)*: # save the points to the queue  
 if int*(*year*)* == date.today*()*.year: # If it is this year, increase all points 10x  
 team*[*1*]* = int*(*team*[*1*])*\*10  
 driver1*[*1*]* = int*(*driver1*[*1*])*\*10  
 driver2*[*1*]* = int*(*driver2*[*1*])*\*10  
 lock.acquire*()* # Lock the queue  
 q.put*(*f"*{*year*}*,*{*round*}*,*{*rName*}*,*{*team*[*0*]}*,*{*team*[*1*]}*,*{*driver1*[*0*]}*,*{*driver1*[*1*]}*,*{*driver2*[*0*]}*,*{*driver2*[*1*]}*\n"*)* #Add the data to the queue  
 lock.release*()* # Unlock the queue  
  
  
def get\_points*(*driver1\_data,driver2\_data*)*: #Get all data required to calculate points for both drivers  
 drivers\_data = *[[*int*(*driver1\_data*[*'fPosition'*])*,int*(*driver1\_data*[*'gPosition'*])*,int*(*driver1\_data*[*'fastPosition'*])*,int*(*driver2\_data*[*'fPosition'*])]*,  
 *[*int*(*driver2\_data*[*'fPosition'*])*,int*(*driver2\_data*[*'gPosition'*])*,int*(*driver2\_data*[*'fastPosition'*])*,int*(*driver1\_data*[*'fPosition'*])]]* #Creates a list so that all the data can be inputed into the loop  
 driver\_points = *[]* team\_points = 0  
 from points import point\_distribution as pd  
 for i in range*(*0,2*)*: #loop twice to get both drivers points  
 fPos = drivers\_data*[*i*][*0*]* #Extract all data from the list  
 gPos = drivers\_data*[*i*][*1*]* fLapPos = drivers\_data*[*i*][*2*]* teammatePos = drivers\_data*[*i*][*3*]* points=0  
 """Race Points"""  
 try:   
 points = points + pd.assign\_points*[*'DriverPoints'*][*'Race'*][*'Results'*][*str*(*fPos*)]* #add points to driver for finishing finishing top 10 or minus 25 for not finishing  
 team\_points = team\_points + pd.assign\_points*[*'Constructor'*][*'Race'*][*'Results'*][*str*(*fPos*)]* #Add points to team  
 except KeyError:   
 points = 0  
 points = points + *((*int*(*gPos*)* - int*(*fPos*))*\* pd.assign\_points*[*'DriverPoints'*][*'Race'*][*'PGFG'*])* #add points to driver for position gained from grid  
 team\_points = team\_points + *((*int*(*gPos*)* - int*(*fPos*))* \* pd.assign\_points*[*'Constructor'*][*'Race'*][*'PGFG'*])* if fLapPos == 1:   
 points = points + pd.assign\_points*[*'DriverPoints'*][*'Race'*][*'fLap'*]* #Add points to driver for achiving fastest lap  
 team\_points = team\_points + pd.assign\_points*[*'Constructor'*][*'Race'*][*'fLap'*]* #Add points to team for fastest lap   
 if fPos > teammatePos:   
 points = points + pd.assign\_points*[*'DriverPoints'*][*'Race'*][*'FAT'*]* #add points for finishing above teammate  
 elif fPos < teammatePos:   
 points = points + pd.assign\_points*[*'DriverPoints'*][*'Race'*][*'FBT'*]* #add points for finishing below teammate  
 try:team\_points = team\_points + pd.assign\_points*[*'Constructor'*][*'Race'*][*'Results'*][*str*(*fPos*)]* #Add points to team for result in race  
 except KeyError:   
 pass  
 """Quallifying Points"""  
 try:   
 points = points + pd.assign\_points*[*'DriverPoints'*][*'Qualifying'*][*'Results'*][*gPos*]* #Add driver points for qually pos  
 team\_points = team\_points + pd.assign\_points*[*'Constructor'*][*'Qualifying'*][*'Results'*][*gPos*]* #Add team points for qually pos  
 except KeyError:   
 pass  
 if gPos <= 15:  
 points = points + pd.assign\_points*[*'DriverPoints'*][*'Qualifying'*][*'RQ2'*]* #Add points for qually positions above 15  
 team\_points = team\_points + pd.assign\_points*[*'Constructor'*][*'Qualifying'*][*'RQ2'*]* #Add team points for qually positions above 15  
 if gPos <= 10:  
 points = points + pd.assign\_points*[*'DriverPoints'*][*'Qualifying'*][*'RQ3'*]* #Add points for qually positions above 10  
 team\_points = team\_points + pd.assign\_points*[*'Constructor'*][*'Qualifying'*][*'RQ3'*]* #Add team points for qually positions above 10  
 driver\_points.append*(*points*)* return *{*'constructor':team\_points,'driver1':driver\_points*[*0*]*,'driver2':driver\_points*[*1*]}*def split\_driver\_points*()*:  
 data = open*(*f'*{*directory*}*raw\_points.csv','r'*)*.readlines*()[*1:*]* teams = *{}* drivers = *{}* for line in data:  
 """Convert the file data back into useful variables"""  
 line = line.split*(*','*)* team = line*[*3*]* team\_points = line*[*4*]* driver1 = line*[*5*]* driver1\_points = line*[*6*]* driver2 = line*[*7*]* driver2\_points = line*[*8*]* """take all points for each driver and append them to a list in a dictionary for that driver"""  
 variable\_set = *[[*driver1,driver1\_points,drivers*]*,*[*driver2,driver2\_points,drivers*]*,*[*team,team\_points,teams*]]* for driver in variable\_set: # iterate through each driver  
 if driver*[*0*]* in driver*[*2*]*:  
 driver*[*2*][*driver*[*0*]]*.append*(*int*(*driver*[*1*]))* else:  
 driver*[*2*][*driver*[*0*]]* = *[*int*(*driver*[*1*])]* """Add the points in the list together and replace the list with a total"""  
 for team in teams:  
 teams*[*team*]*=sum*(*teams*[*team*])* for driver in drivers:  
 drivers*[*driver*]*=sum*(*drivers*[*driver*])* """Sort each dictionary so its easier to read"""  
 drivers\_sorted=dict*(*sorted*(*drivers.items*()*,key= lambda x:x*[*1*]))* teams\_sorted=dict*(*sorted*(*teams.items*()*,key= lambda x:x*[*1*]))* """save the points to a json file"""  
 save\_final\_points*(*drivers\_sorted,'driver\_points.json'*)* save\_final\_points*(*teams\_sorted,'team\_points.json'*)* print*(*'Team and Driver points Compleated'*)*def save\_final\_points*(*points,file*)*:  
 file = open*(*directory+file,'w+'*)* file.write*(*str*(*points*)*.replace*(*"'",'"'*))* file.close  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 import time  
 get\_drivers*()* start = time.time*()* get\_race\_data*()* print*(*f"Runtime of the program is *{*time.time*()* - start*}*"*)* split\_driver\_points*()*

### points/points\_distribution.py

#Points given for tasks  
assign\_points=*{* 'DriverPoints':*{* 'Race':*{* 'Results':*{* '1':25,  
 '2':18,  
 '3':15,  
 '4':12,  
 '5':10,  
 '6':8,  
 '7':6,  
 '8':4,  
 '9':2,  
 '10':1,  
 'F':-25, #F,D,W,R,N is DNF  
 'D':-25,  
 'W':-25,  
 'R':-25,  
 'N':-25  
 *}*,  
 'fLap':1, #Fastest Lap  
 'PGFG':1, #Points per Position gained From grid  
 'PLFG':-1, #Points per Position lost From grid  
 'FAT':3, #Finished above teammate  
 'FBT':-3, #Finished below teammate  
 *}*,  
 'Qualifying':*{* 'Results':*{* '1':3,  
 '2':2,  
 '3':1,  
 *}*,  
 'RQ2':5, #Reached Q2  
 'RQ3':5, #Reached Q3  
 'QBT':-3, #Quallified Below Teammate  
 'QAT':3 #Quallified Above Teammate  
 *}  
 }*,  
 'Constructor':*{* 'Race':*{* 'Results':*{* '1':25+3,  
 '2':18+3,  
 '3':15+3,  
 '4':12+3,  
 '5':10+3,  
 '6':8+3,  
 '7':6+3,  
 '8':4+3,  
 '9':2+3,  
 '10':1+3,  
 'F':-25, #F,D,W,R,N is DNF  
 'D':-25,  
 'W':-25,  
 'R':-25,  
 'N':-25  
 *}*,  
 'fLap':1, #Fastest Lap  
 'PGFG':1, #Points per Position gained From grid  
 'PLFG':-1, #Points per Position lost From grid  
 *}*,  
 'Qualifying':*{* 'Results':*{* '1':3,  
 '2':2,  
 '3':1,  
 *}*,  
 'RQ2':5, #DriverReached Q2  
 'RQ3':5, #Driver Reached Q3  
 'BQ3':3, #Both drivers reach Q3  
 *}  
 }  
 }*

### chat/chat\_server.py

import ast  
chat\_list = *[]*def handle\_client*(*client\_socket*)*:  
 *"""  
 Handle a client connection for the chat server.  
 """* # Receive the message  
 request = client\_socket.recv*(*1024*)* request = ast.literal\_eval*(*request.decode*())* if request*[*0*]* == 'send':  
 chat\_list.append*(*request*[*1*])* if request*[*0*]* == 'get\_chats':  
 chat\_log = 'Chat:'  
 for item in chat\_list:  
 chat\_log = chat\_log + '\n' + item  
 client\_socket.send*(*chat\_log.encode*())* # Send the message back  
  
 # Close the connection  
 client\_socket.close*()*