MuscleMax

A solution for tracking fitness and nutrition

Name: Armaan Khaitan

Candidate number: 7385

Centre name: Greenford High School

Centre number: 12432

Contents

ANALYSIS

Problem Description

My application is a fitness tracking and progress app to help people keep track of their different workouts, suggest suitable ones, and create a plan for helping them achieve a goal. I have a friend who's planning to open his own gym that thinks that a desktop app would be particularly useful to have for his customers in his gym.
It is also a place where they can enter their food for the day which will be split into ingredients and calories calculated and a chart generated displaying their split of different food groups.

My client asked me to write a program that could record the user's profile and record peoples' workouts, suggesting workouts to beginners and allowing more experienced people the option to edit their workouts and display and save their progress. Also, my client asked me to add a feature that could take their current meal and split it into ingredients, with the calories collated and the different food groups displayed in a chart format.Background:
Many people find it difficult to keep track of their progress in the gym and at home when

exercising. They miss having a clear display with all the in-built plans and processing done for them, allowing them to focus on their workouts. On Google Play store and Apple App Store there were different apps to measure progress in gym and home workouts and track one's diet too. However, no app could be found for free that could meet these two criteria fully within the same app. Often the principal problem people find on their fitness journey is recording their progress and consistency. My computer application will help keep track of their workouts and nutrition and show their progress to encourage them to break their records.

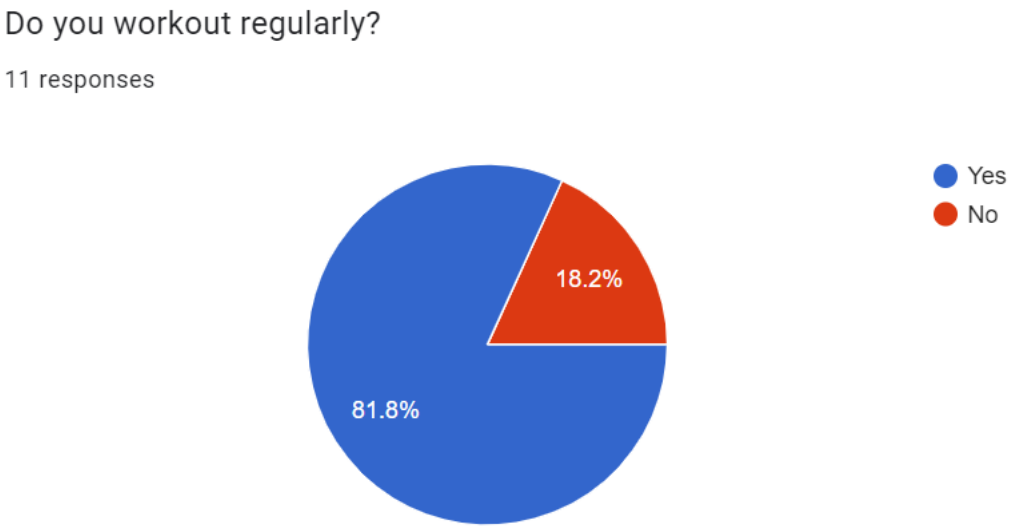
End User Investigation

I formed these questions to gauge a sense of my user's requirements and how well I can fulfil these requirements with the time, resources and skills I have. For the options of workouts and nutrition I did some research and added some of the topmost popular workouts tracking and nutrition tracking apps as part of the multi choice questions.

For example, the "Sworkit Fitness & Workout App" is ranked #1 fitness app by ACSM – American College of Sports. It has over 135,000 5-star reviews and an average rating of 4.7 on Google Play Store.

Questionnaire:

1) Do you work out regularly?

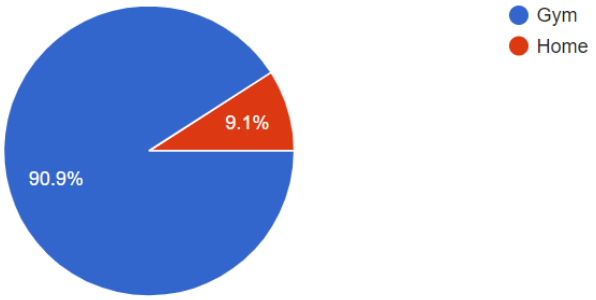


2) Where do you workout? If you workout in both the gym and home, select whichever one you workout in most of the time. If you workout elsewhere, please specify in other.

Where do you workout? If you workout in both the gym and home select whichever one you workout in most of the time. If you workout elsewhere please specify in other

Copy

11 responses

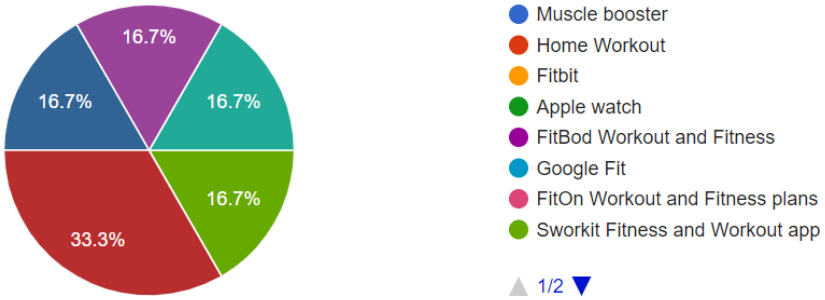


3) If you record your workout, please select which one and write in other if not listed.

If you record your workout please select which one and write in other if not listed

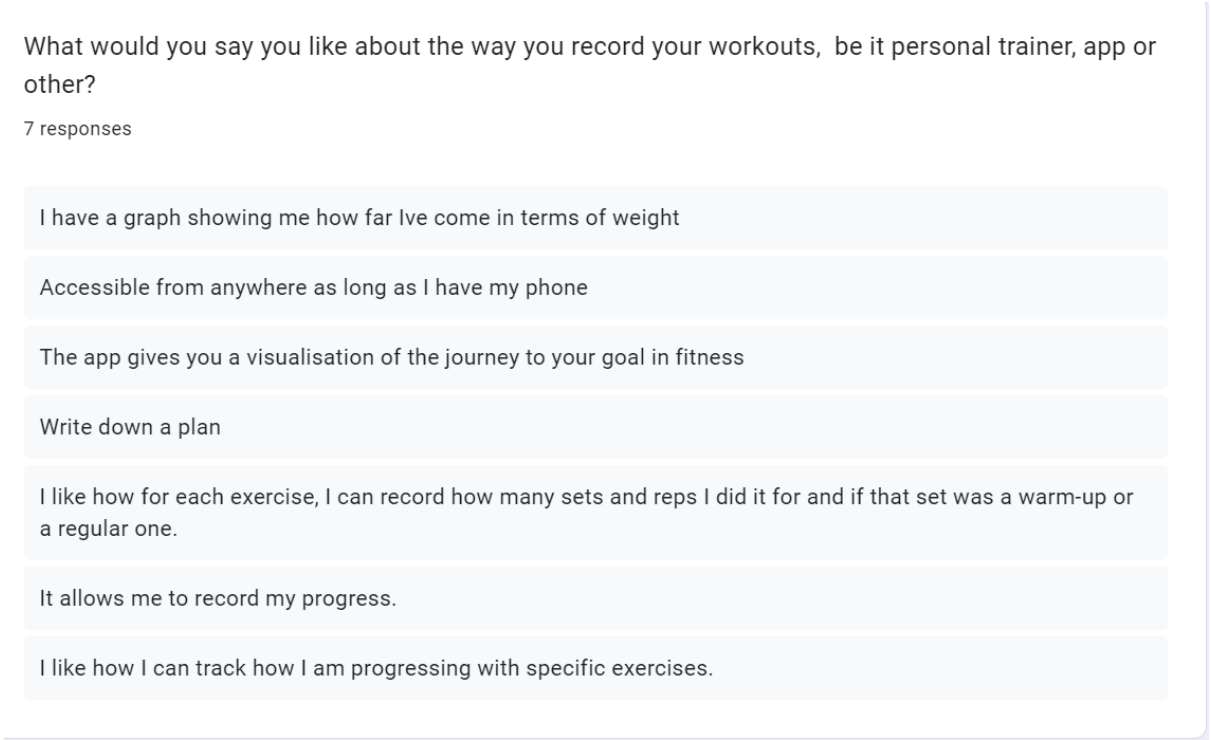
Copy

6 responses



- Personal trainer
- Strong app - the strong app selection was here in your last Google form but u removed it
- Notes app
- Hevy - Gym Log Workout Tracker

4) What would you say you like about the way you record your workouts, be it personal trainer, app or other?



5) What would you say is missing from the way you record your workouts, be it personal trainer, app or other?

What would you say is missing from the way you record your workouts, be it personal trainer, app or other?

6 responses

Nothing tbh just copy the strong app I don't go gym that often for the last year but I remember it's a good app

I don't know tbh it does what it needs to do

Recording weight

I want to be able to log my nutrition and my workouts on the same app.

It would be good to see approximately how many calories I spend in a workout

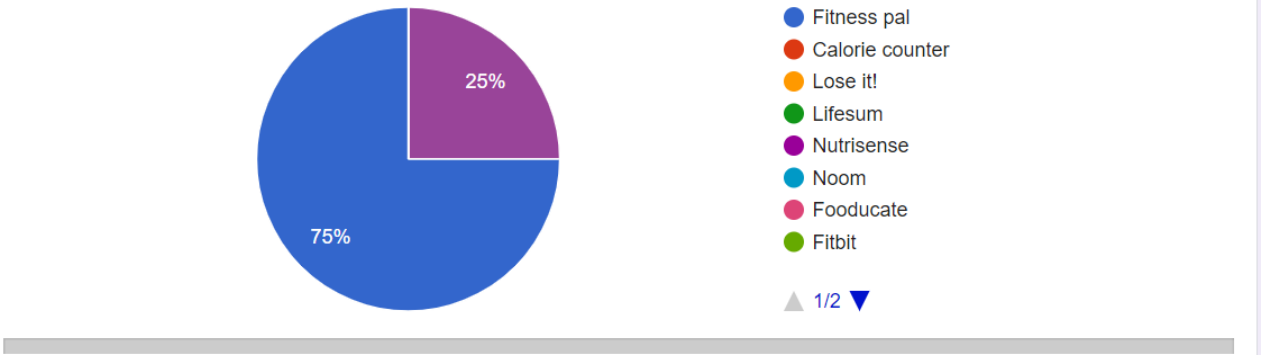
An estimate of how many calories I spend doing these exercises.

6) Food: If you record your nutrition, please select which one and write in other if not listed:

Food: If you record your nutrition please select which one and write in other if not listed

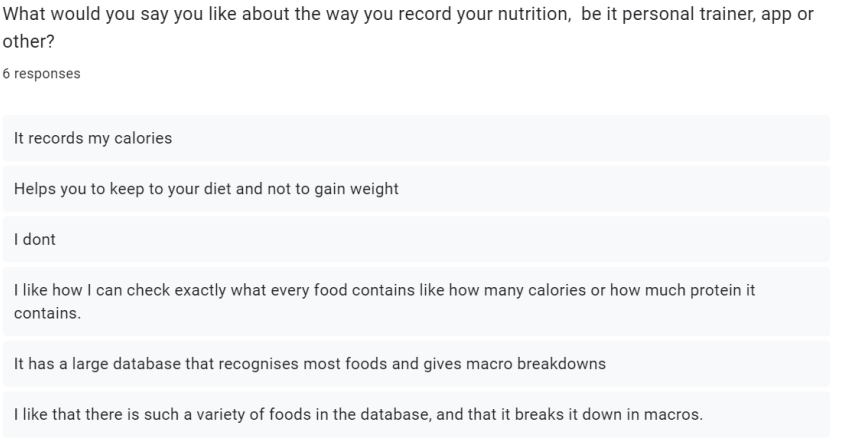
Copy

4 responses

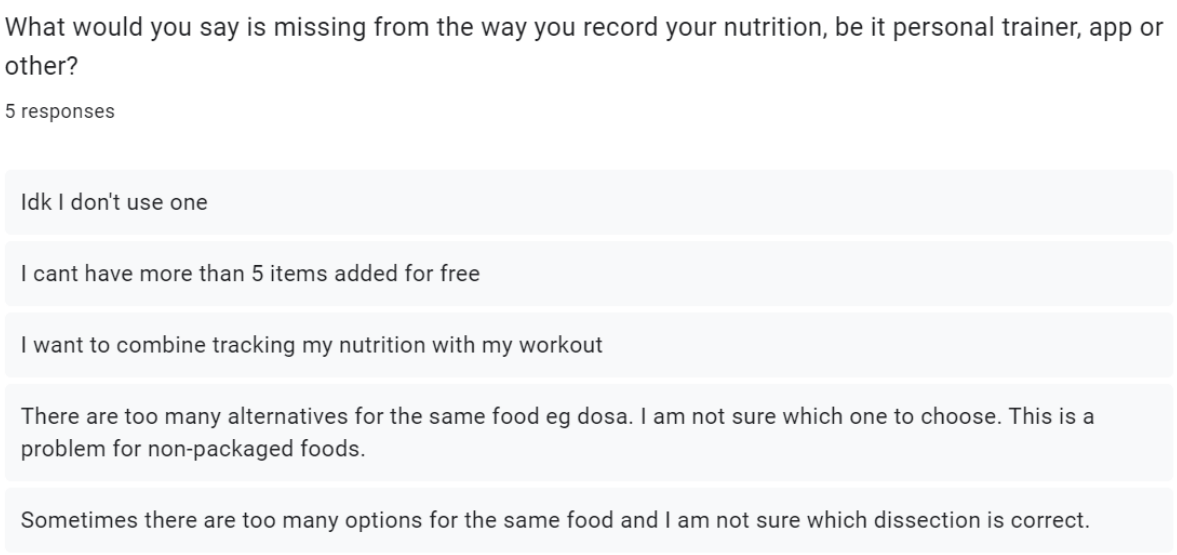


Apple watch
Personal trainer
Nutracheck

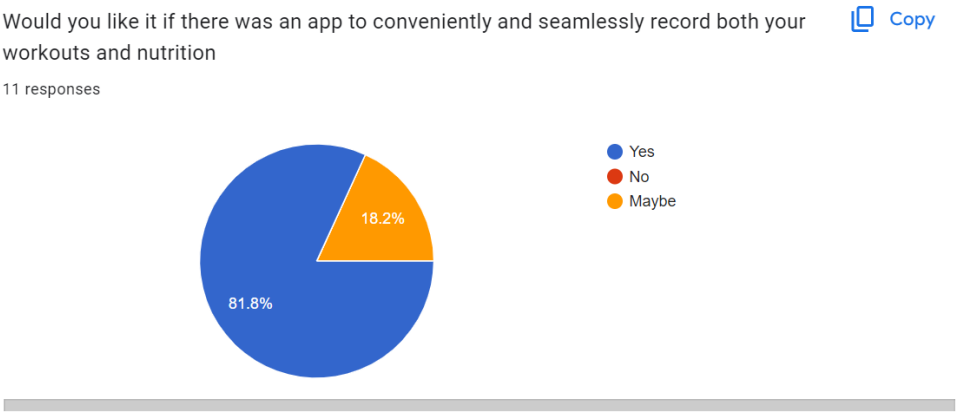
7) What would you say you like about the way you record your nutrition, be it personal trainer, app or other?



8) What would you say is missing from the way you record your nutrition, be it personal trainer, app or other?



Would you like it if there was an app to conveniently and seamlessly record both your workouts and nutrition?



Firstly, all the people who work out regularly have said they think my app would be useful. There was a lot of variances in the type of workouts used ranging from Sworkit Fitness to Personal trainer. 75% of people said they use Fitness pal as a nutrition tracking app with the other 25% using Nutrisense. This goes to show how popular and culturally diverse Fitness pal is. It is also said to be the number one health and fitness tracking app in the world – just for nutrition, it's capabilities are limited to just food.

Subject Research

I have researched 3 applications to explore what potential applications are already available and understand their strengths and weaknesses to mold my application accordingly:
Fitness Pal – It is the second most and popular fitness app.
Home Workout – It was the number one most downloaded fitness app in 2021 – 28 million times.
Muscle Booster – It was the 5th most downloaded fitness app in 2021 – 18.5 million times.

Fitness pal

Fitness pal records the calories in meals. The user chooses how many calories they want to eat per day based on several questions that find their fitness goals.
It allows the user to pick between four meals: breakfast, lunch, dinner, snack
Once they have picked the meal, they can search through a database of food that will tell

them about the calories for the food items, accounting for the number of portions eaten. The total calories are then subtracted from the user's daily calorie goal.

Pros	Cons
Effectively tracks calories consumed	Cannot track workouts for which a separate app is needed, which is inconvenient.
Displays the calories in a pie chart	User cannot manually enter calories if not in search bank
	Cannot track or record workouts

Home workout

This app has pre-suggested workouts and training plans that the user can select. However, they are only for exercise at home and not for the gym. Also, the user cannot add their own workouts or edit the existing workouts. It is useful in having a set plan for beginners to start exercising and stick to a regime. However, for someone seriously interested in fitness, it is not useful because the options are too limited and restrictive.

Pros	Cons
Offers a training plan	Does not record calories or offer diet plans, which is very inconvenient
Displays what muscle/area is being worked	As the name suggests, it only records home workouts and so is useless for people who also go to the gym

Muscle booster

This app asks the user questions to gauge their level of fitness.

In my opinion the questions are slightly tailored to the gym so people who work out at home with no equipment may be judged as amateurs even if they're not - however this point may be unavoidable in my code too.

It has a quite nice display of different muscle groups the user can train, with the option to select single or multiple muscles to train at one time; this is something that I might incorporate into my program.

The layout of the workouts with a separate frame appearing when the workout is clicked is also nice, which I may choose to adopt.

It also splits the workouts into morning workouts and evening, lengthy workouts in a calendar format that I also would possibly adopt using the "calendar" module of python. This app is one of the best apps I've seen so far for recording workouts.

Description of current system or similar current products/games

The problems with the current systems

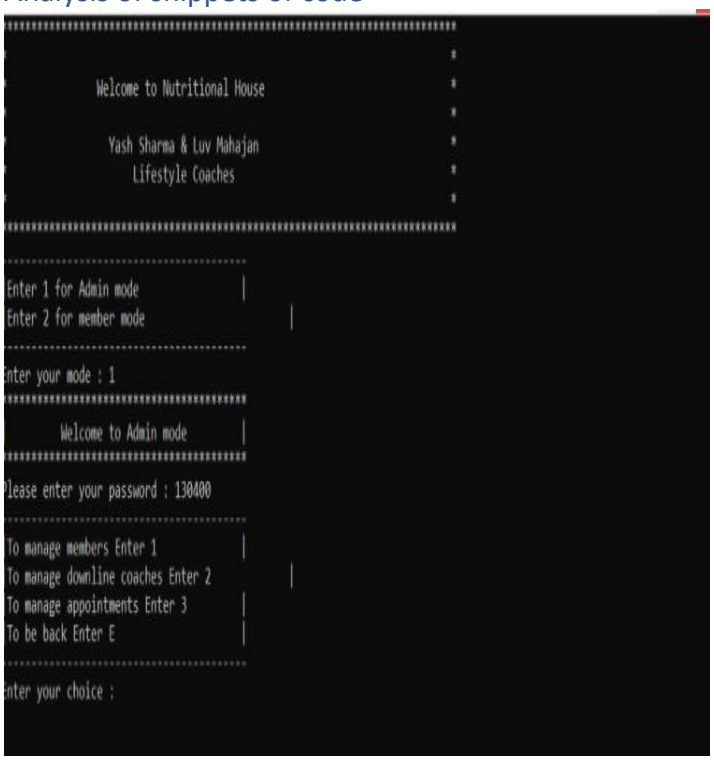
They don't record both calorie analysis and workout analysis.

Most apps need manual entering to create entire workouts.

Many apps are tailored too much to one side of working out whether that be gym, home, aerobics, calisthenics (a form of strength training consisting of a variety of movements that exercise large muscle groups. These exercises are often performed rhythmically and with minimal equipment).

This restricts variance in workouts as a combination of these is usually the most optimal workout to do. My application might possibly solve this problem.

Analysis of snippets of code

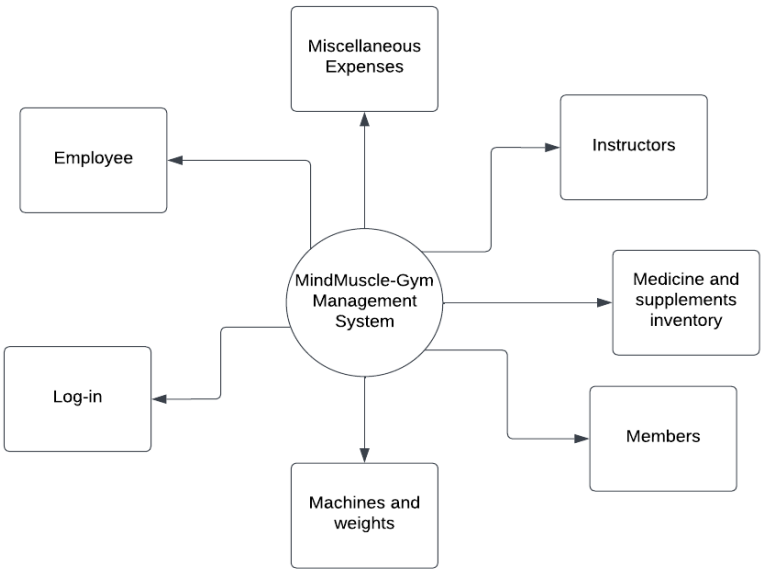


1. The idea of admin mode and user mode is an interesting concept I may choose to implement. The user enters a password that allows them access to either the owner of the app status or just a guest status and according to distinctive features will be unlocked.

2. Instead of the "downline coaches" and "appointments", as seen in

the snippet, I will have workouts, split, calories, charts, etc. and use a user interface that will most likely be tkinter, but turtle and pygame are highly unlikely but available options

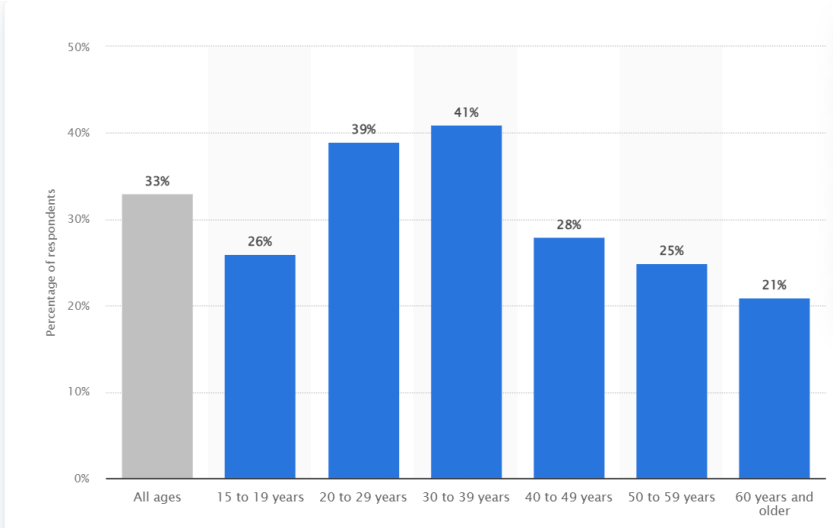
Data flow diagram of an existing system



This diagram represents the flow of data split into different sections representing distinct parts of an existing application. The structure seemed intuitive and clever, and I plan to use some of the structure from here, for example the log-in, the members, the machines and weights. Obviously, this application is built for slightly differing purposes than mine so some parts of this diagram and not relevant for me.

Additional user requirements/limitations

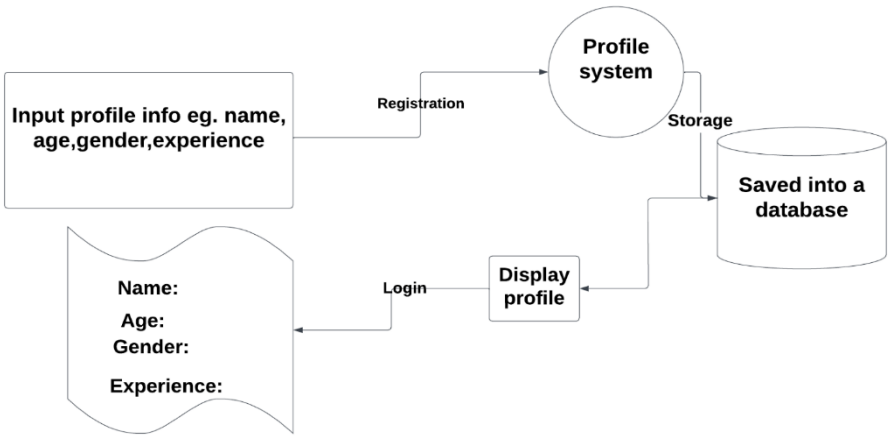
My app doesn't particularly have a specific scope, able to be used by anyone who pleases, ranging from beginners to very advanced and seasoned people. Tracking and recording are helpful at all experience levels.



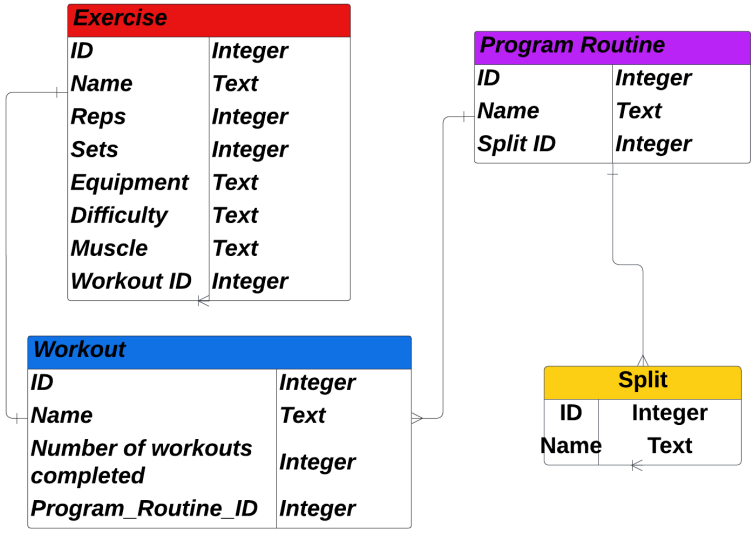
This diagram shows that the main demographic of people who use fitness apps are 30-39 years of age. However, for all age groups more than 20% use fitness apps. This is why my application is not targeted towards a specific age group. This link says that coronavirus saw a drastic increase in fitness apps being used, for example Strava, an application I recently downloaded, which has been recommended to me by a friend. I quite like the idea of a login and profile page for the user. The login and signup process will be short. Either one should only take a few seconds maximum to complete. The client will only need to enter their details in designated and very self – explanatory boxes, then press a button Login / SignUp depending on which page their on. The login or signup page will be pre – decided by the code. After the client enters their details and presses the button, the rest will be done by the code, and they will be able to access the application. A status of experience will be decided in the beginning based on questions which will be updated from beginner, to intermediate to advance to display their fitness journey too. Access to the source code will be restricted to avoid unintentional or malicious tampering or avoid people editing their workouts to make it seem like they're much more advanced than they are. To help me understand my end user requirements I conducted an interview with students in my school, asking questions about the current system to gain an understanding of the needs of my target users

My systems' designs

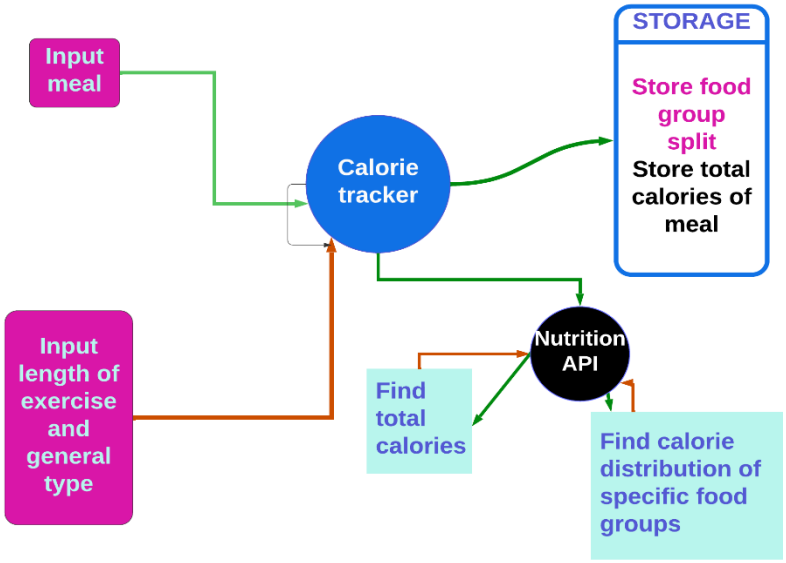
Context diagram (Data Sources and Destinations)



ERD diagram



Data flow diagram for my system



Database overview - data dictionary for essential entities

Name	Data type	Characters	Status
Weight	Integer	2	None
Reps	Integer	2	None
Name	Alphanumeric	10-20	Not null
Exercise	Text		
Exercise_ID	Integer	1-2	Primary Key
Equipment	Alphanumeric Text	5-10	None
Difficulty	Alphanumeric Text	5-15	None
Day	Alphanumeric Text	7	None

Specific objectives of the proposed system

Based on my research and concept diagrams I have summarised the functionalities of my application into the following objectives:

1. A login page will load when the app is started. This will include a sign in page and a registration page. The registration page with collect name, age, gender, password while the sign in page will allow the user to enter their username and password to access the system
2. There will be a set of questions to decide the experience level based on the user's answer to sthe questions
3. A page will load showing the user's profile of experience, age, gender, name with some colour and a picture if they want, else a default will be provided. The user interface will load the profile of the user within a few seconds / immediately of logging in or signing up. All other functionality of the application will also respond within a few seconds / immediately of clicking on any feature of the application.
4. There will be a separate page for 6 different pre-made workouts but I could possibly build in the functionality of being able to change the number of pre-made workouts. Each workout will have editable and removable options and a dropdown menu of all the workouts available. There will also be a choice to add exercises if not available.
5. A dynamic workout saved database and workout history will be in place. The database will update instantly as soon as the user implements any of the update functionalities
6. There will be a separate page that appears when a certain workout is picked showing the exercises to be completed with the weight, reps, sets and equipment, muscle and difficulty level all there. It will be a checkbox beside each one and a master checkbox for every workout that will increase the number of workouts completed when clicked.
7. When a certain exercise is pressed there will be an image and the relevant information there
8. There will be a button to view a graph showing the progress of workouts info: reps against set or reps against weight etc...
9. There will be a separate tab for calories.
10. There will be an entry box to either search up the meal in a databank of meals or manually enter the calories.
11. There will be a circle showing how many calories have been eaten and are left. There will be a clickable button to show a pie chart of the split of calories into different food group categories with different colours representing each category.
12. There will be a history of nutrition showing the meals each day.

Limitations of the proposed system

Limited time:

I have about 2.5 months to complete the technical solution to my problem, which means although I will try my best it could possibly not be the absolute best I could produce because of the time limit

My programming skill

I am trying to implement GUIs, APIs, databases and website scraping. I have not used APIs, databases or website scraping many times before so it will prove challenging when programming it. This means I could produce a relatively more simplistic version of what could be achieved with these tools.

Lack of powerful processing power

Because I am programming on a cloud-based platform called Replit, or the IDLE shell of python, I have limited processing power and therefore need to optimize my code to run smoothly and efficiently.

Type of application:

The application is a computer app meaning it can only be used on laptops and PCs, not mobile phones. A mobile or web application could have been other options, but I have chosen to create computer app. This could be used by users on their own computers or the gym computer

Proposed solution analysis

Overview / breakdown of problems

After analyzing practical solutions (Fitness pal, Home Workout and Muscle Booster), I have chosen to create my own solution in the programming language of python:

I have chosen to write in python because I have been using and coding with python for approximately 3-4 years now, so time will not be wasted learning a new programming language like JavaScript, Java, Rust, or C++.

Python is a mainly object-oriented dominant language, not as much as Java but still mainly written in object orientation.

It supports many database types like MySQL, SQLite3, Oracle, Sybase, PostgreSQL etc……

This therefore means that I can create a relational database with tables for my workouts, calories and recording current and history of workouts.

Moreover, I can then use classes to call this data that has been stored in secondary storage and link a GUI with them to interact with the user and be able to edit and update this

database.

Python

```
print("What is your name?")
user = input()
print("Hello",user)
print("Please write all of your answers to these questions in lower case
letters")
print("Do you like pizza?")
user = input()
if user == "yes":
  print("Me too!")
  print("Pizza is my favourite food")
else:
  print("Your Strange")
  print("How do you not like pizza!")
  print("What do you like then?")
  user = input()
  if user == "pizza":
    print("But I thought you didnt like pizza")
    print("Anyway,")
  else:
    print("I like that too!")
print("How old are you?")
user = input()
print("So your",user)
print("Am I correct?")
user = input()
if user == "yes":
  print("Ok I was just checking")
```

The object orientation of python means I can create functions that I can assign buttons, frames, and check buttons to.

Python is a widely used programming language nowadays alongside JavaScript and java which means there are many official websites with free help I can use like, “Stack Overflow”, “GitHub”, and other discussion forums like “Student Room”

Python also has an intuitive interface with easy to add breakpoints and an inbuilt debugger that shows the status of the variables and functions at the point you stop it. This allows for easier and effective debugging by going through each line slowly and checking where error occurs.

Advantages of using python	Disadvantages of using python
Easy to learn and understand syntax and “grammar”	Slow speed
Vast collection of varied libraries	High memory usage
Free, open-source and a vast community	Garbage collection leads to potential memory losses which could lead to

	unintended eradication of widgets that weren't meant to be destroyed.
Interpreted language which is helpful due to the inevitable plethora of errors and bugging during creation	Multi-threading enables you to write in a way where multiple activities can continue concurrently in the same program. Python doesn't do this very well although it does support basic threading.
It is the second highest paid computer	The data types of variables in Python can
language which means that skill in this language especially for a project of this size is useful	change suddenly, as it is a dynamically typed language. A variable holding a string may hold an integer later, and this can lead to runtime errors.
Used in data science, machine learning, and server-side web development	Code can't be blocked off like in jupyter notebooks and VS studio code reducing the readability of it,
Highly scalable as it is suitable for building large-scale web applications.	The presentation and analysis of massive data volumes by a Python app,
	Third-party libraries must be used to compensate for the lack of ability of python. [This won't affect my project but is an overall disadvantage of python].

Implementation methods

Tkinter

A python module used to create GUI projects

Tkinter is a graphical user interface that allows users to interact with computers through icons like labels, buttons, entry boxes etc.....

It's simple and easy to code, interact and learn with

It uses labels to display text

It has many secondary functions which allow for direct typing into GUI window, like checkboxes, buttons, entry boxes, radio buttons, frames, text widgets

It helps supply the visual representation of my code to a layperson person using my program.

It is one of the most used UIs so therefore makes sense that it is easy to use and interact with

It has three geometry managers: place (), pack (), grid () which are powerful and easy to use.

Pygame

More specifically a game programming module used primarily for graphics and mouse binding events like arrow keys

It is a free and open-source cross-platform library for the creation of complex projects using Python

It has many different modules like music, images, shapes, text, buttons, keys, sound, videos and more.

OOP classes identified

Create database

Insert into database

Display Workouts

Display Profile

Display specific exercises

Display database in a comprehensible view and allow editability of the view and database where coder allowed

Display meals, and caloric split

Add workouts

Add increment of weights and reps to status table and history of workouts take

Add manual suggestion of workouts

Add split of calories from the meal inputted using an API

Add each workout checked with checkbox to history of workouts table

Show the pie chart for the calories and the line graph or something for the progression of weight and reps and sets [including home workouts] throughout your journey in working out.

Data sources

Modules

API - application programming interface

A software that allows communication between two servers/clients acting as the connection between the two seeming to the programmer as a flawless direct connection

Exercise API

Supplies access to a comprehensive list of hundreds of exercises targeting every major muscle group

Supplies the name, type, muscle, biceps, equipment, difficulty, instructions of each exercise

I can therefore extricate the necessary exercises I require irrelevant information from each one and load them into a relational database I create that I can then manipulate and use to suit my needs

Nutrition API

Extracts nutrition information from text using natural language processing.

From food blogs to menus to recipes, it can read any text and calculate the corresponding nutrition data.

An intelligent feature of this API is custom portioning: if your text specifies quantities of individual food items or ingredients, the algorithm will automatically scale the nutrition data in the result accordingly.

This will allow me to split the user's inputted meals into separate categories of calories, enabling me to display both their total amalgamation of calories and their split into:

Total fat

Total saturated fat

Total protein

Total sodium

Total potassium

Total cholesterol

Total carbohydrates

Total fiber

Total sugar

Internet scraping

Some information that I need that would be inefficient to manually enter

However, if it cannot be found on the APIs I have might need to come from the internet.

I could use several python libraries to seamlessly extract information from certain websites

Then manipulate my code to gather the requisite information I need.

Beautiful soup

A python library for pulling data out of HTML files

It supplies simple ways of searching, and editing the parse tree

The parse is a visual representation of the syntactic structure of a piece of source code, as produced by a thing called a parser

Parsers are used when there is a need to represent source code abstractly as a data structure/ in a way that is more interpretable to the human brain)

It is all about visualizing the structure of the syntax of the website

It shows the hierarchy of the elements in the code and the relationships between them.

I could use this to pull data from the HTML source code of websites or the homepage of the google browser

Requests

It is a python module used to send a protocol for fetching resources such as HTML documents [HTTP requests] across a browser.

I can use this in tandem with beautiful soup and selenium to pull data from the internet for my use like researching workouts and calories in the case of this application.

Sqlite3

This could possibly be one of the most useful modules.

Sqlite3 is a module that allows me to create, insert into, update, remove and select information to and from a custom database of my creation.

This module is so useful because it allows me to save data on there that will stay there even after the program stops running and can be recalled and updated when the program requires

This drastically increases efficiency and decreases processing time.

Very quick and easy to implement

Lacks user management and security features

Not easily scalable

Is not suitable for big databases

Cannot be customized

MySQL

MySQL is also easy and quick to implement

It cannot be used on large database sizes and slows down in such cases

Superior SQL databases are Oracle, SQL server, PostgreSQL.

However, they could possibly require many complicated downloads and complications that I might not want to waste time on

NumPy

A python library primarily used for working with arrays, linear algebra, matrices, complex mathematical concepts like Fourier transformations and calculus

I may need to use this module and my knowledge of product moment correlation coefficient, linear regression, and averages to calculate the values for the charts of progression of weights and fluctuation in amount of certain categorical intake in meals for example:

I could use it to calculate how saturated fat intake has increased/decreased

I could use it to calculate how protein intake has increased/decreased

NumPy arrays are faster and more compact than Python lists:

NumPy uses much less memory to store data and it supplies a mechanism of specifying the data type

Some capabilities of the NumPy module include:

String manipulation

Powerful n-dimensional arrays

Image manipulation capabilities

Easy to use.

OS

This module is a way for programmers to interact with the operating system.

This module allows them to create new files, folders, handling and changing existing or new directories, transferring files into different directories, checking current path or current directory and other things like this…

This could allow me to save sound files to the current folder that can be listened to by the user, perhaps an audio of the instructions of an exercise.

Pathlib

It is thought that managing the files and folders is very inconvenient in os

Perhaps “Pathlib” could be a better choice, another module in python used for interacting with the operating system and handling file management

Potential data structures which may be used

Stack

Perhaps a stack will be used for the history of workouts part and progression of weights and reps

The last item checked will be the one displayed at the top, so in essence the first one displayed.

A database will be used to store the workouts

Perhaps a queue can be used to store the calories each day:

Displaying the first set of meals you had on the day you started the application and the last one being the one you entered just before closing the application.

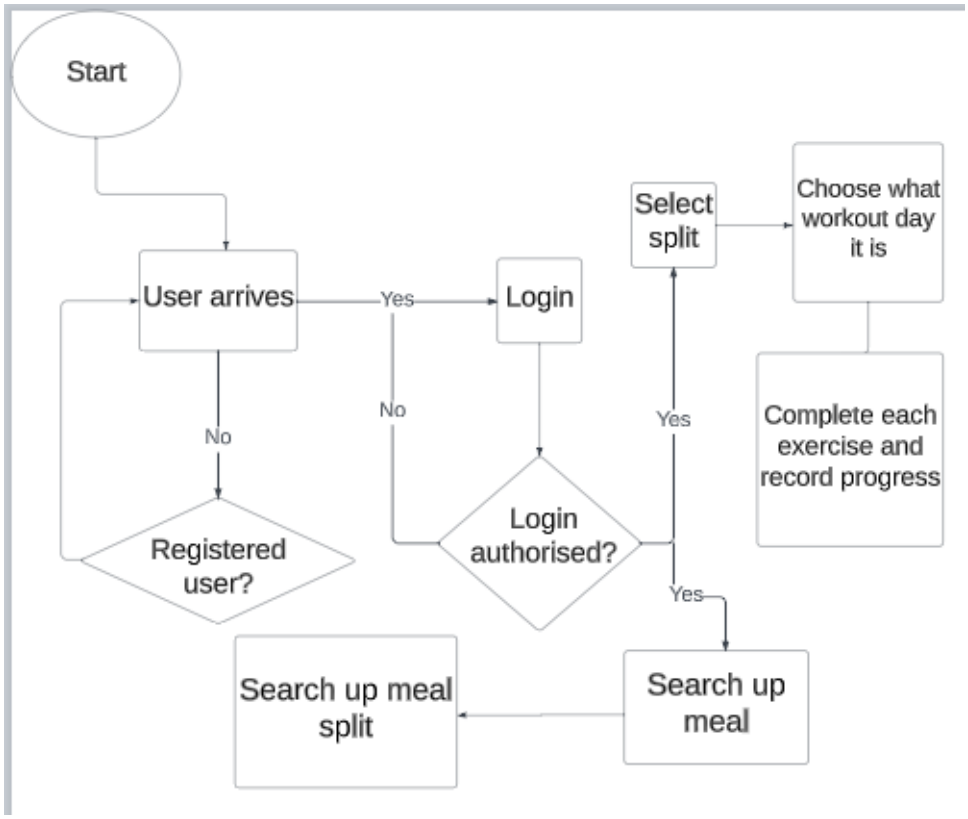
Perhaps an array can be used to store the user’s profile and preferences which can then be transferred to a user’s profile that can be called from and displayed.

DESIGN

This design section is the structures and ideas I've have decided to implement based on research conducted in the Analysis section:

Outline system design

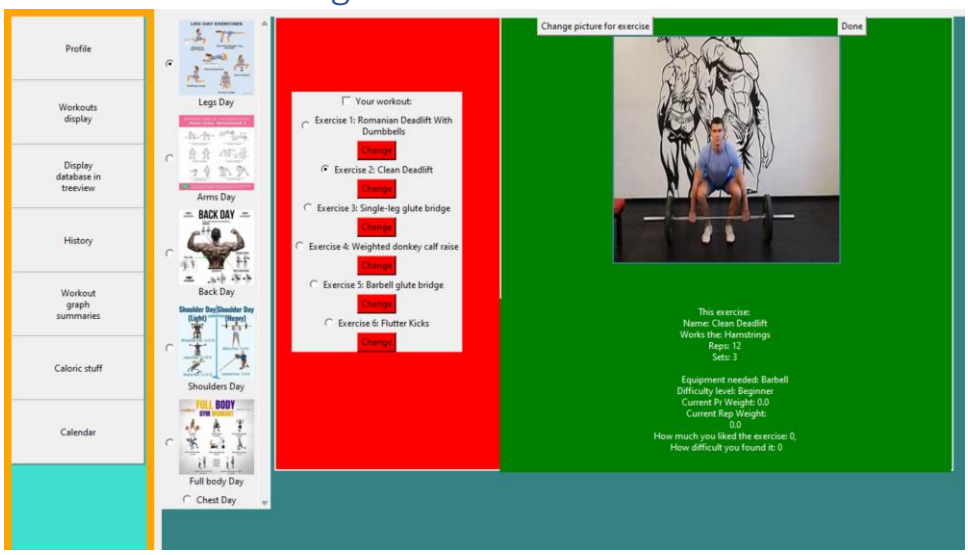
System flowcharts



The user runs the program.
They are taken straight to the login page if they are already a registered user and to the sign in page if they are not.
In the registration they enter their details and are then taken into the rest of the program
In the signing in, the user is given 3 opportunities to enter the right details. If they get it wrong thrice they are told to reset their password.
After correct login they are sent to the page that asks them questions to decide their level of experience. Then they enter the app which shows them their workouts and allows them to view information on a specific exercise
They can view all possible exercises in a separate tab, which also allows them to edit the values, which is where they can update their progress
Then there is the calorie tab where they can search up a meal and it will show them the calorie split
Then the calendar tab shows them a summary of their workouts on any day and their max lift and exercise name
The graph tab shows them a horizontal bar graph of their different weights and reps and

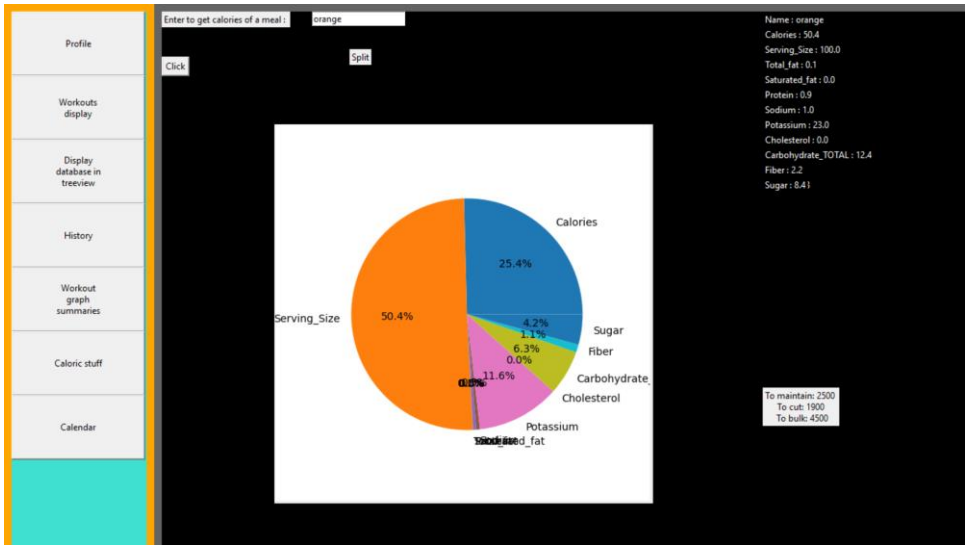
exercises against one another. Since there are multiple graphs available, the user can pick which one they want to view

User Interface Design



This one shows the different workout days possible and a randomly selected exercises generated when pressed. When the exercises are pressed, they show information about it and an image which can be changed and a button which will take them to another tab to update their weights/reps/difficulty etc....

This is where all the updating happens, of all the possible fields that the user is available to update. For the fields that the user can't update, the widgets are created but just not placed onto the screen. So, in essence those fields are invisible so can't be edited by the user.



This show calories split and a graph

The UI has been optimised for ease of use of the user. The first tab for example allows for the switch between workout day, specific exercise, change of specific exercise and change of specific exercises picture.

Hardware specification

A modern dual – core processor or better
4GB of RAM or better

At least 50 MB of free storage space or more

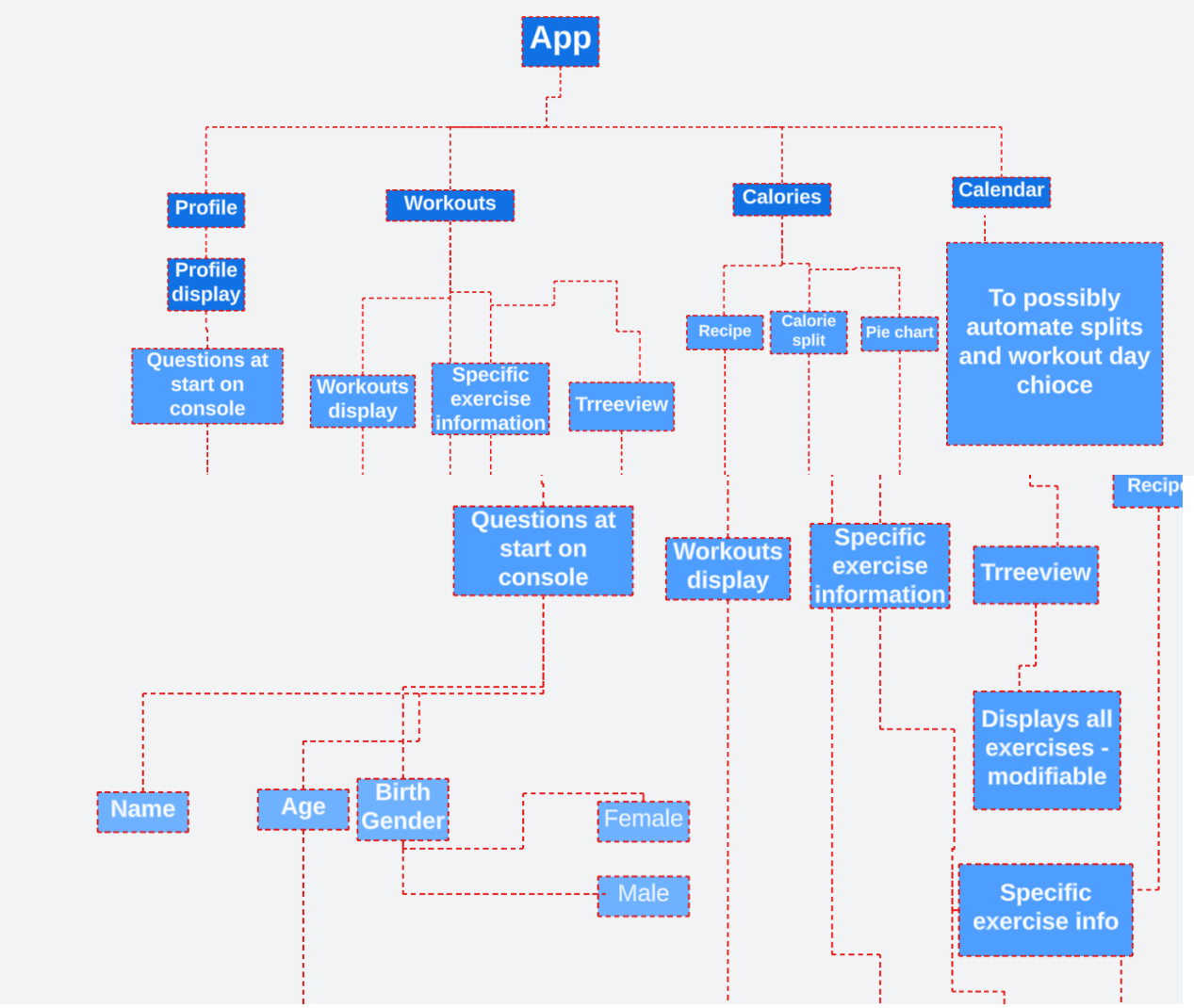
Default integrated graphics (such as integrated GPU or Intel UHD or Iris or baseline AMD Radeon) or better

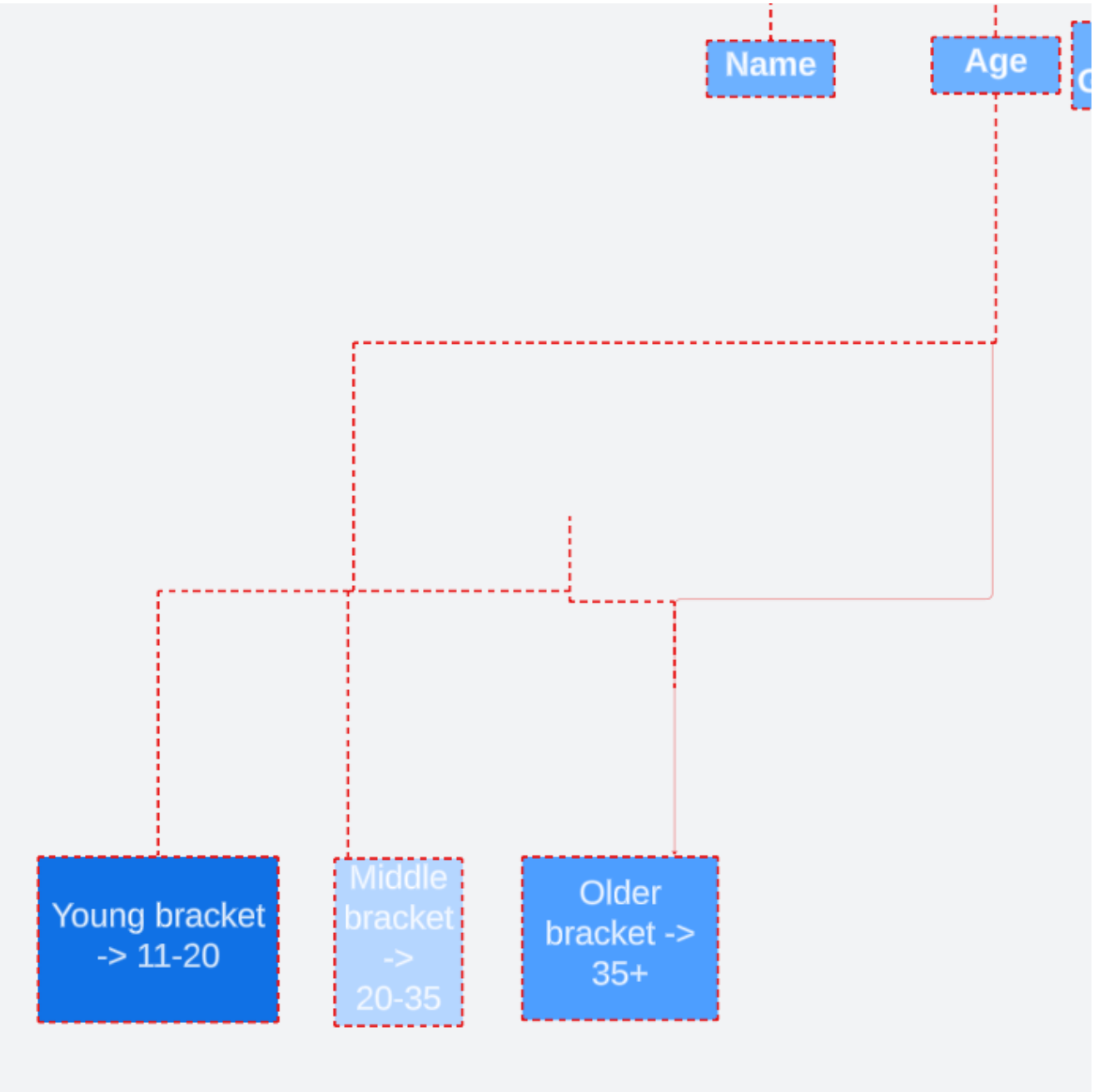
<u>Extra but not essential specifications:</u>

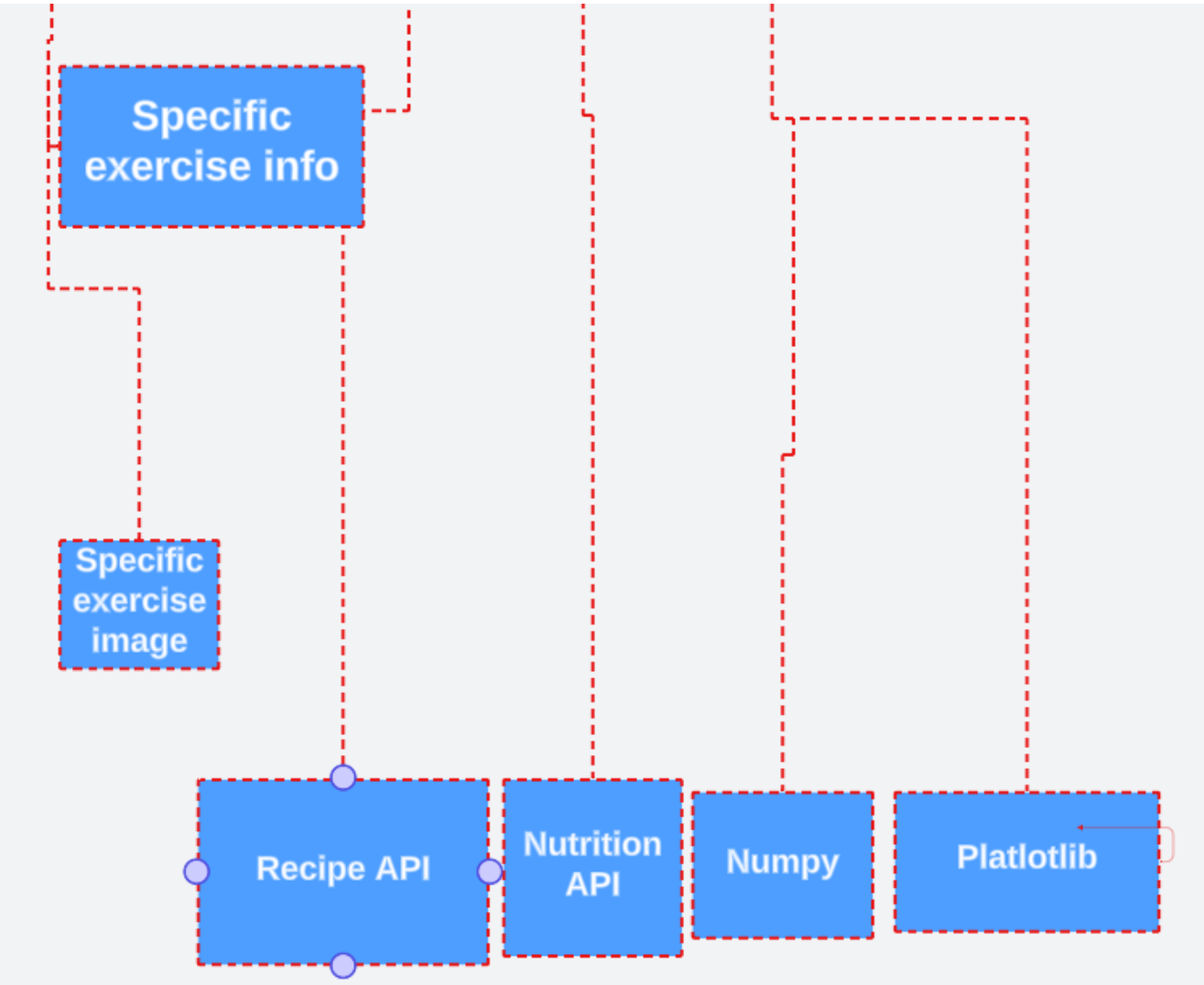
Intel Core i5 (12th or 13th-gen) or AMD Ryzen 5 (3000 or 5000 series)

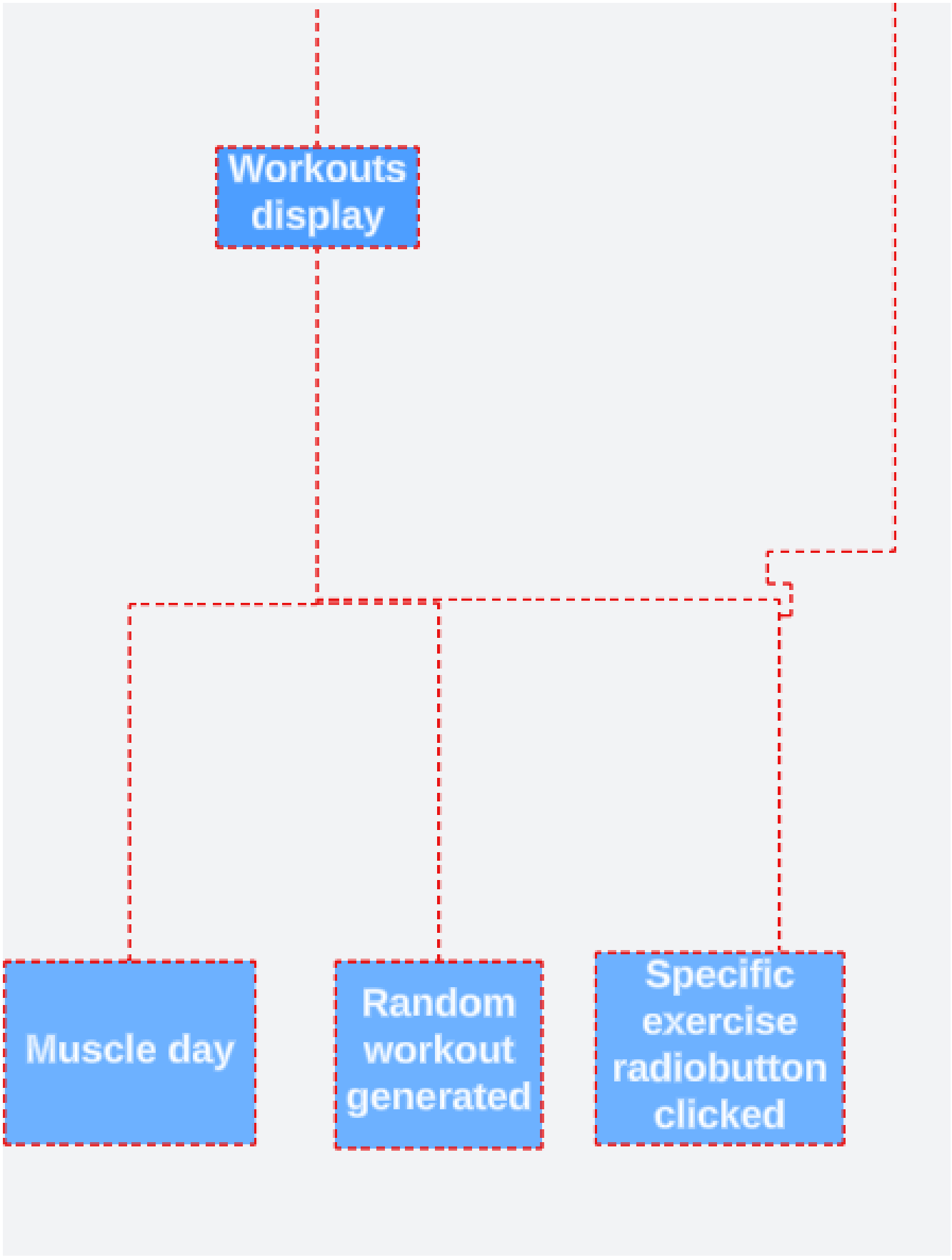
512GB or larger NVMe SSD drive

Program Structure




Name
Age
Young bracket -> 11-20
Middle bracket -> 20-35
Older bracket -> 35+


Specific exercise info
Specific exercise image
Recipe API
Nutrition API
Numpy
Platlotlib


Workouts display
Muscle day
Random workout generated
Specific exercise radiobutton clicked

Design Data Dictionary

Data dictionary for all data in my program later in design as part of the "normalised database tables"

Name	Data type	Characters	Status
Weight	**Integer**	**2**	**None**
Reps	**Integer**	**2**	**None**
Name	**Alphanumeric**	**10-20**	**Not null**
Exercise	**Text**		
Exercise_ID	**Integer**	**1-2**	**Primary Key**
Equipment	**Alphanumeric Text**	**5-10**	**None**
Difficulty	**Alphanumeric Text**	**5-15**	**None**
Day	**Alphanumeric Text**	**7**	**None**

Object diagrams and class definitions

1. A function for switching between different frames in my tkinter GUI
2. A function for creating the buttons that switches between the frames. It's a dynamic function that uses a dictionary to create buttons based on an array of names at the start of the code that defines all the buttons' names
3. A function called "main" that iterates through a certain folder on local storage looking for a certain file and returns that file path
4. Two functions for scraping images from the internet and saving them to local storage
5. A class to display random auto generated workouts in Tkinter GUI which is dynamic so can be edited based on the user's requirements
6. A class to create a dynamic Treeview display of all the exercises available that can be modified and changed based on the user's needs
7. A class for creating GIFs in Tkinter. I use this throughout multiple parts of my code so a separate class for it is a good idea
8. A class for displaying specific exercise information and a check button to mark when the exercise is done which automatically sends the user to a page where they can record their exercise
9. A class for connecting to my google sheets and manipulating it through my tkinter code
10. A class to represent the calendar display

11. A class to represented caloric manipulation, inputting a meal and getting the graph split and food-calorie split. The calculations are accurate upto the point of the API I am using. I might implement the fitness pal database but since it may take too long that may be outside the scope of the project.

Data Structures

- Stack:
    - History of workouts – first entry put in is the last one to be displayed
- Sqlite3 database
    - To save all the workouts, exercises and be updated
- Dictionary
    - To hold the entry and label widgets in the treeview frame of the GUI
    - To hold the buttons to switch between the frames in the GUI
- Queue
    - The array that holds the information for the profile – the first information is the first one displayed [left-most or uppermost depending on positioning of information on the page]

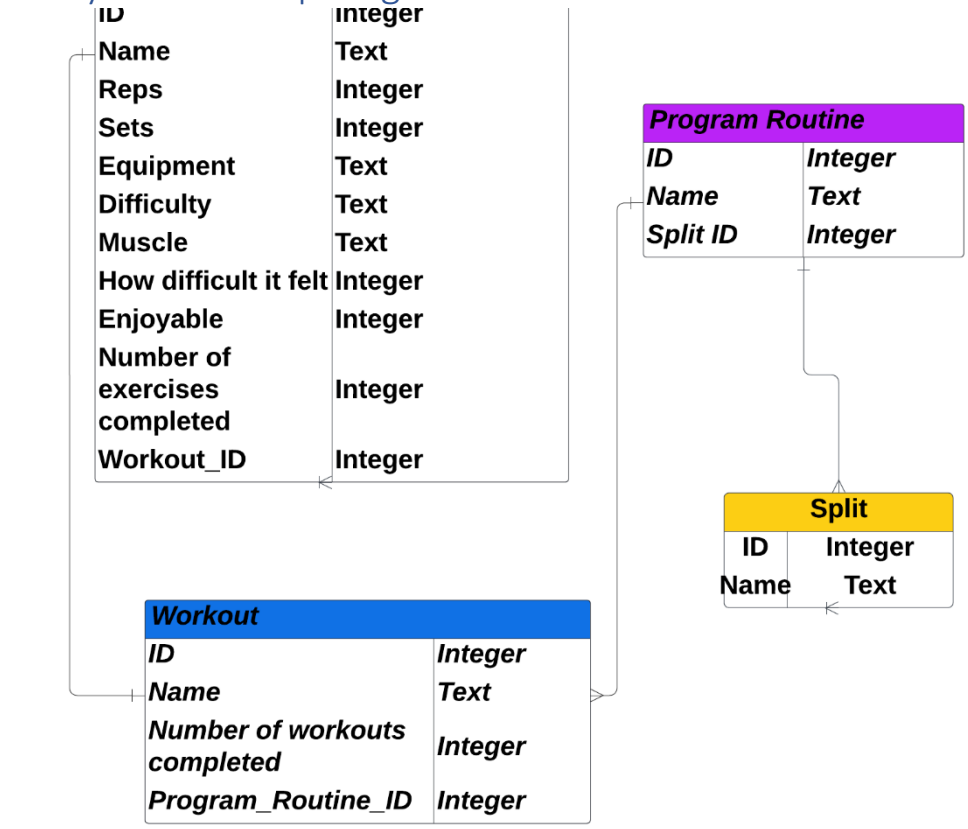
File Organisation

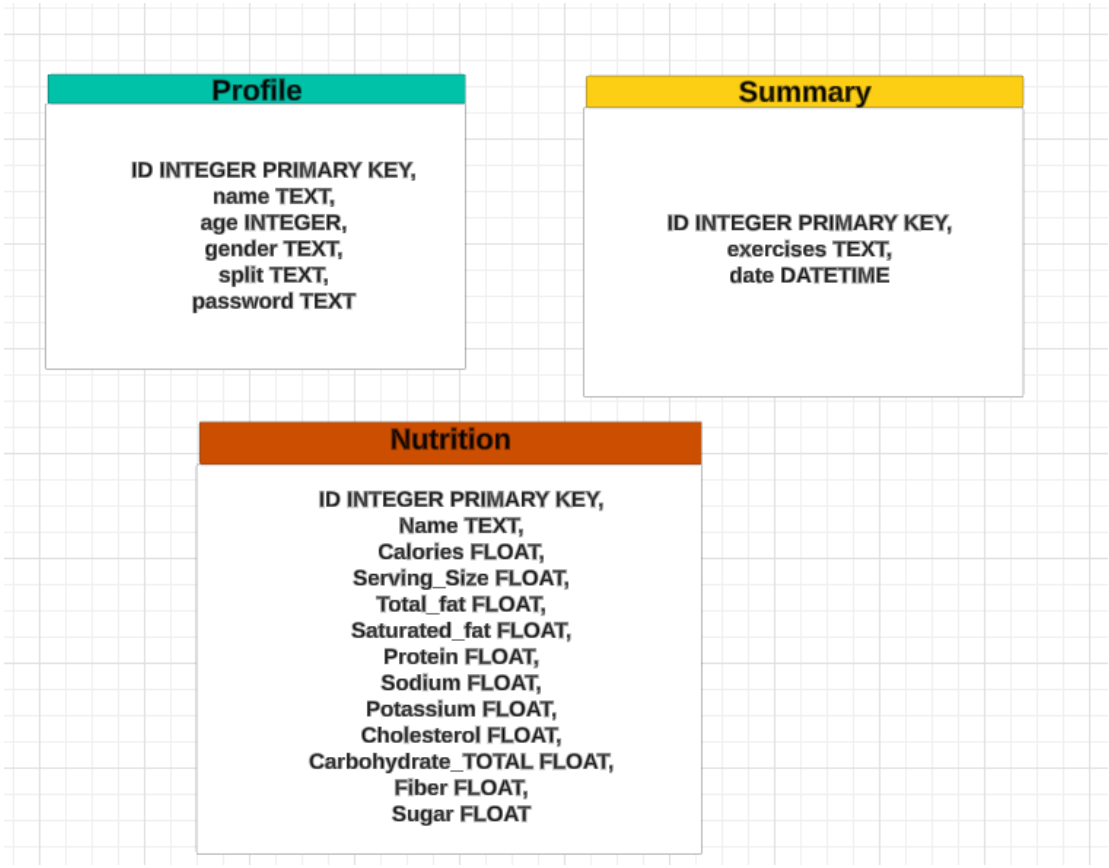
- There is a nea.json file to link my code to my google sheets.
- There is a folder of images and gifs scraped and saved locally to the disk

APIs used

1. Nutrition API
2. Exercise API
3. Calories burned API

Entity-relationship diagram





Normalised database tables

Attribute Name	Primary key? Foreign key?
Exercise_ID	Primary key
Name	Normal attribute
Reps	Normal attribute
Sets	Normal attribute
Equipment	Normal attribute
Difficulty	Normal attribute
Muscle	Normal attribute
How difficult it felt	Normal attribute
Enjoyable	Normal attribute
Number_of_exercises_completed	Normal attribute
Workout_ID	Foreign key
Workout_ID	Primary key
Name	Normal attribute
Number_of_workouts_completed	Normal attribute
Program_Routine_ID	Foreign key
Program_Routine_ID	Primary key
Name	Normal attribute
Split_ID	Foreign key
Split_ID	Primary key
Name	Normal attribute

Attribute Name	Sample data
Exercise_ID	1
Name	Pullups
Reps	10
Sets	3
Equipment	None
Difficulty	5
Muscle	Back
How difficult it felt	6
Enjoyable	7
Number_of_exercises_completed	3
Workout_ID	3
Workout_ID	5

Name	Glutes
Number_of_workouts_completed	7
Program_Routine_ID	4
Program_Routine_ID	2
Name	Legs
Split_ID	8
Split_ID	9
Name	5-day split

Attribute Name	Data type [Sqlite3]
Exercise_ID	INTEGER
Name	TEXT
Reps	INTEGER
Sets	INTEGER
Equipment	TEXT
Difficulty	INTEGER
Muscle	TEXT
How difficult it felt	INTEGER
Enjoyable	INTEGER
Number_of_exercises_completed	INTEGER
Workout_ID	INTEGER
Workout_ID	INTEGER
Name	TEXT
Number_of_workouts_completed	INTEGER
Program_Routine_ID	INTEGER
Program_Routine_ID	INTEGER
Name	TEXT
Split_ID	INTEGER
Split_ID	INTEGER
Name	TEXT

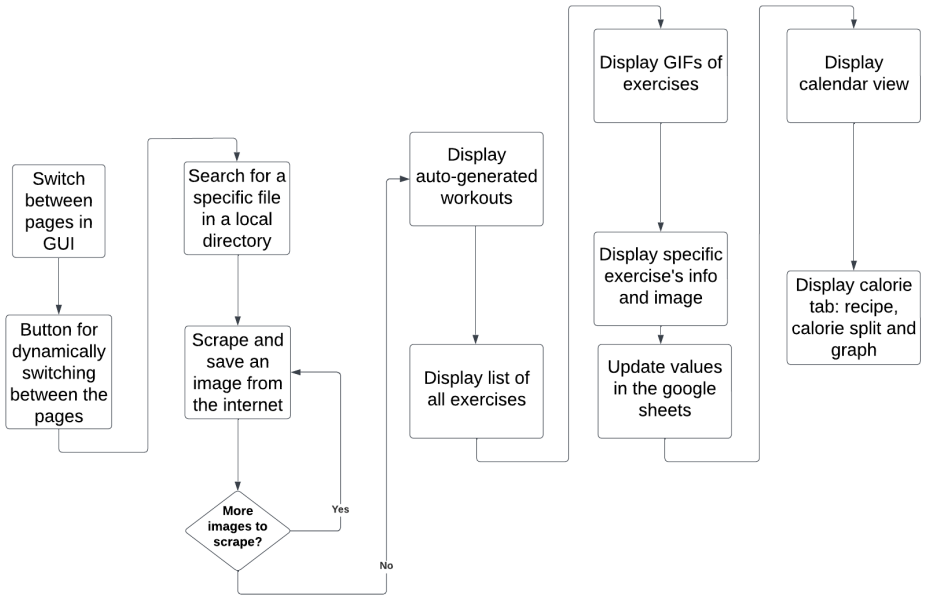
Attribute Name	Data type [Python]
Exercise_ID	Integer
Name	String
Reps	Integer
Sets	Integer
Equipment	String

Difficulty	Integer
Muscle	String
How difficult it felt	Integer
Enjoyable	Integer
Number_of_exercises_completed	Integer
Workout_ID	Integer
Workout_ID	Integer
Name	String
Number_of_workouts_completed	Integer
Program_Routine_ID	Integer
Program_Routine_ID	Integer
Name	String
Split_ID	Integer
Split_ID	Integer
Name	String

Attribute Name	Size or range
Exercise_ID	Max 4 letters
Name	Max 30 characters
Reps	Max 4 letters
Sets	Max 4 letters
Equipment	Max 30 characters
Difficulty	Max 4 letters
Muscle	Max 30 characters
How difficult it felt	Max 4 letters
Enjoyable	Max 4 letters
Number_of_exercises_completed	Max 4 letters
Workout_ID	Max 4 letters
Workout_ID	Max 4 letters
Name	Max 30 characters
Number_of_workouts_completed	Max 4 letters
Program_Routine_ID	Max 4 letters
Program_Routine_ID	Max 4 letters

Name	Max 30 characters
Split_ID	Max 4 letters
Split_ID	Max 4 letters
Name	Max 30 characters

Algorithms



This flowchart is the representation of the approximate solution to the dynamic workout objectives

Top-down description of modules

import sqlite3
I need sqlite3 to create and utilise a database
import os
I need os to manipulate files on my local storage and write programs that can adapt and carry out processing on any user's local storage
import string
import re
I need re and string to carry out generalised string character recognition
import babel
from babel import numbers
import math
import numpy as np
I need babel, numpy and math for more complex maths operation like resizing image arrays

```
import tkinter as tk
```
I need tkinter to create a GUI
```
from tkinter import ttk,filedialog as fd,messagebox
```
I need filedialog and messagebox to allow user to access local storage folders through the code
```
from PIL import Image,ImageTk, Image, ImageSequence
```
I need PIL to make Images and GIFs in my GUI
```
import requests
```
I need requests to utilise my API information and to carry out internet scraping
```
from bs4 import BeautifulSoup
from urllib.parse import urljoin
import urllib.request
import threading
import gspread
```
I need urllib.parse, urllib.request, threading and gspread and bs4 to carry out internet scraping

```
import random
from collections import Counter
```
I need random to generate random numbers and Counter to carry out string frequency occurrence

```
import time

import imageio.v2 as imageio
```
I need imageio.v2 to manipulate images
```
from datetime import datetime
```
I need datetime to extract and manipulate dates
```
from oauth2client.service_account import ServiceAccountCredentials
from googleapiclient.http import MediaIoBaseDownload
from googleapiclient.discovery import build
```
I need these to access and update my google sheets

```
import tkcalendar
from tkcalendar import *
```
I need tkcalendar to create and use a calendar in my GUI
```
import subprocess
```
I need subprocess to dynamically install modules
```
import ast
```
I need ast to produce the literal array from a string that hold an array

Detailed test data

Attribute Name	**Valid data example(s)**	**Invalid data example(s)**	**Boundary data example(s)**
Exercise_ID	3	-6, True, “Hello”,17.5	0
Name	“Lateral pulldown”	17, True	"True” or “False”
Reps	3	-6, True, “Hello”,17.5	0
Sets	3	-6, True, “Hello”,17.5	0

Equipment	Barbell	True,17, -5,17.8	"True" or "False"
Difficulty	Beginner	True,17, -5,17.8	"True" or "False"
Muscle	Glutes	True,17, -5,17.8	"True" or "False"
How difficult it felt	3	-6, True, "Hello",17.5	0
Enjoyable	3	-6, True, "Hello",17.5	0
Number_of_exercises_completed	3	-6, True, "Hello",17.5	0
Workout_ID	3	-6, True, "Hello",17.5	0
Workout_ID	3	-6, True, "Hello",17.5	0
Name			
Number_of_workouts_completed	3	-6, True, "Hello",17.5	0
Program_Routine_ID	3	-6, True, "Hello",17.5	0
Program_Routine_ID	3	-6, True, "Hello",17.5	0
Name	"Lateral pulldown"	17, True	"True" or "False"
Split_ID	3	-6, True, "Hello",17.5	0
Split_ID	3	-6, True, "Hello",17.5	0
Name	"Lateral pulldown"	17, True	"True" or "False"

TECHNICAL SOLUTION:

Basics file

```
import sqlite3
import requests
import os
import subprocess
import gspread
import ast
from oauth2client.service_account import ServiceAccountCredentials
from googleapiclient.http import MediaIoBaseDownload
from googleapiclient.discovery import build
import time
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from tkinter import filedialog
def find_folder(parent_folder_name,target_folder_name):
    if parent_folder_name == None:

        # Check the OneDrive Desktop path
        parent_folder_path = os.getcwd()
        #print(f"Checking {parent_folder_path_onedrive}")
        if os.path.exists(parent_folder_path) and os.path.isdir(parent_folder_path):
            for item in os.listdir(parent_folder_path):
                item_path = os.path.join(parent_folder_path, item)

                if os.path.isdir(item_path) and item == target_folder_name:
                    return item_path

        return None
    else:


        # Check the OneDrive Desktop path
        parent_folder_path_onedrive = os.path.join(os.getcwd(), parent_folder_name)
        #print(f"Checking {parent_folder_path_onedrive}")
        if os.path.exists(parent_folder_path_onedrive) and os.path.isdir(parent_folder_path_onedrive):
            for item in os.listdir(parent_folder_path_onedrive):
                item_path = os.path.join(parent_folder_path_onedrive, item)
                if os.path.isdir(item_path) and item == target_folder_name:
                    return item_path
        else:
            return None

def find_file(file_name):

    parent_folder_path = os.getcwd()
    if os.path.exists(parent_folder_path) and os.path.isdir(parent_folder_path):
        for item in os.listdir(parent_folder_path):
            item_path = os.path.join(parent_folder_path, item)
            if item == file_name:
                return item_path
```

```
    else:
        return None

project_pictures_folder = find_folder("Code","Project pictures")
base_path = rf'{os.getcwd()}/Project pictures/Images'
def main(search):

  # List all files and directories in the specified directory
  global contents
  contents = os.listdir(base_path)
  # Iterate over the contents and do something with each item (file or directory)
  for filename in contents:
    if filename == search:
      file_path = os.path.join(base_path, filename)
      return file_path
    else:
      pass


Database_path = str(find_file('My database'))
connection = sqlite3.connect(Database_path)
c = connection.cursor()
```

These 2 screenshots above are the “basic.py” file. They serve 3 functions:
To import the necessary imports that are needed throughout the code and are also not GUI imports
A function that returns the file path for a specific folder on any user's computer
These takes 2 inputs:
Name of the parent folder name
Name of target folder name

It uses the method in the OS module called “.getcwd(~)”. This method returns the beginning file path of the user's current working directory enabling the file path to adapt to any computer

It then joins the parent folder name to the getcwd () file path to create the file path to the parent folder name on any computer.

It then checks if this file path exists and whether this file path is a folder. Then it iterates through all the files and folders in that path until it finds the one that matches the required folder name. Next it joins this folder name to the file path to the parent folder name and returns this file path. It also checks whether there is a valid parent folder name input – if there isn’t it simply uses the getcwd () file path as the parent folder file path

A function that returns the file path for a specific file on any user's computer
This one take 1 input -> Name of target file name

It uses the method in the os module called “. getcwd ()”. This method returns the beginning file path of the user's local working directory enabling the file path to adapt to any computer

It then joins the parent folder name to the getcwd () file path to create the file path to the parent folder name on any computer.

It then checks if this parent folder file path exists and whether this file path is a folder, then it iterates through all the files and folders in that path until it finds the one that matches the required file name. Lastly it joins this file name to the file path to the parent folder name and returns this file path

It uses the find_file function to get the file path to the database file and then initialises the connection to the database

Login system:

```
from basics import *
trials = 0
def reset():
  name = input("Enter your name: ")
  password = input("You've forgetten your old password - Enter your new password: ")
  try:
    print("Please wait while your password is reset...")
    c.execute(f'''
      UPDATE Profile
      SET password = "{password}"
      WHERE name = "{name}"5
      ''')
    connection.commit()
  except Exception as e:
    print(e,"That name isn't valid. Enter the correct name for tohis account")
    reset()

def login(choice,trials):
  global name
  # Sign up
  if choice == 1:
    print("Sign up: ")
    print()
    cur = []
    name = input("Name: ")
    age = int(input("Age: "))
    gender = input("Gender: ")
    password = input("Password: ")
    c.execute(f'''
    INSERT INTO Profile (name,age,gender,split,password,PATH) VALUES ("{name}",{age},"{gender}","Five Day Split","{password}","{find_file("SignIn.jpg")}")
    ''')
    connection.commit()
  # Login
  if choice == 2:
    print("Login")
    print()
    name = input("Enter your name: ")
    password = input("Enter your password: ")
    database_info_of_users = c.execute(f'''SELECT * FROM Profile WHERE name = "{name}" and password = "{password}"''').fetchall()
    if trials <= 1:
      if database_info_of_users:
        print("Login successful")
      else:
        print("Login unsuccesssful")
        trials += 1
        login(2,trials)
    else:
      print("Login incorrect too many times, reset your password")
      reset()
```

This is my login and sign-up system.

Firstly, it checks whether there are any entries in the database
If there are entries, then it goes to the login part by entering 2 for the value of choice.
If there aren't entries, then it goes to the sign-up part by entering 1 for the value of choice.

*The algorithm **recurses** based on valid or invalid inputs*

Constants file

```
from basics import *
import json
import sqlite3
import os
import re
import babel
from babel import numbers
import tkinter as tk
from tkinter import ttk,filedialog as fd,messagebox

import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin
import urllib.request

from PIL import Image,ImageTk, Image, ImageSequence
import math
import random
from collections import Counter


import threading
import gspread
import time

import imageio.v2 as imageio
import numpy as np
from datetime import datetime, timedelta

from oauth2client.service_account import ServiceAccountCredentials
from googleapiclient.http import MediaIoBaseDownload
from googleapiclient.discovery import build

import string

import tkcalendar
from tkcalendar import *

import mplcursors
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt
# Formatting
root = tk.Tk()

root.attributes('-fullscreen', True)
H = 1000
W=1000
geo = f"{W}x{H}"
root.geometry(geo)
```

```
root.geometry(geo)
root.title("Gym")
root['background']='white'

# Constants
add = 0
border = 0
tabs = {}
Buttons = {}
frames = []
file_extension = '.jpg'

texts = ["Profile","Workouts display","Display database in treeview","Workout graph summaries","Caloric stuff","Calendar"]
tab_strings = [f'tab{i+1}' for i in range(len(texts))]
colours = [f'#{random.randint(0,255):02X}{random.randint(0,255):02X}{random.randint(0,255):02X}' for i in range(len(texts))]


project_pictures_folder = find_folder(None,"Project pictures")

base_path = rf"{project_pictures_folder}\Images"
def tab_search(search):
  index = texts.index(search)
  value = tab_strings[index]
  return tabs[value]


def main(search):
  print(search)
  global directory_path
  directory_path = f'{base_path}'
  # List all files and directories in the specified directory
  global contents
  contents = os.listdir(directory_path)
  # Iterate over the contents and do something with each item (file or directory)
  for filename in contents:
    if filename == search:
      file_path = os.path.join(directory_path, filename)
      return file_path
    else:
      pass
```

This imports all required modules for all the GUIs functionality [including the imports from the basics file]
This initialises the dynamic addition of the tabs in the GUI
This stores the tab number like “tab1” in an array of strings and the corresponding colours in another array and the names of the tabs in another array
The tab_search function if an extremely useful function that will return the tab number based on the name of the tab entered so tabs can be easily added and removed, and their colours easily altered.
The main function can iterate through the pictures that are stored in a folder and return the file path of any one of them needed based on the inputted name
There are separate files for constants and basics because this file initialises all the GUI functionality required, and “basics” has only the bar imports and modules required throughout the code.
This is so that the functions that don’t require GUI initialisation and imports can run more efficiently. As seen, the constants file imports the basics file meaning the constants file contains ALL the necessary imports and functions required.

GUI basics file

```
from constants import *
def Frame(frame):
  if frame == tab_search("Display database in treeview"):
    #frame.config(width=W+600)
    for i in frames:
      if i != frame:
        i.config(width=W+1000,height=H+300)
  elif frame == tab_search("Workouts display"):
    #frame.config(width=W+150,height=H)
    for i in frames:
      if i != frame:
        i.config(width=W+1000,height=H+300)
  elif frame == tab_search("Caloric stuff"):
    frame.config(background="black")
    #frame.config(width=W+300,height=H)
    for i in frames:
      if i != frame:
        i.config(width=W+1000,height=H+300)
  else:
    for i in frames:
      i.config(width=W+1000,height=H+300)
  frame.tkraise()

options_frame = tk.Frame(root,width=W-400,height=H+300,highlightbackground='orange',highlightthickness=10,background="turquoise")
options_frame.place(x=0,y=0)
for index,val in enumerate(tab_strings):
  tabs[val] = tk.Frame(root,height=H+300,width=W+1000,highlightthickness=10,background=colours[index])



for i in tabs.keys():
  frames.append(tabs[i])
  tabs[i].place(x=W-400,y=0)
  tabs[i].pack_propagate(False)
def Button(first_half_of_name,second_half_of_name,full_name):
  number_of_buttons = len(tab_strings)
  button_gap_distance = H // (number_of_buttons)
  button_gap_distance = int(button_gap_distance)
  number = re.findall(r'\d+', first_half_of_name)
  number = int(number[0])
  Buttons[full_name] = tk.Button(options_frame,text=texts[number-1],height=5,width=24,command=lambda:Frame(tabs[first_half_of_name]),wraplength=75)
  coordinate = (number-1) * (button_gap_distance)
  Buttons[full_name].place(x=0,y=coordinate)
for i in tabs.keys():
  second_part = "_button"
  combined = i + second_part
  Button(i,second_part,combined)
```

This file handles the appearance of the basics of the GUI
It creates the frame where the buttons to switch between tabs sit
The next line loops through the tab_strings array and dynamically adds to the empty dictionary tabs.
It assigns each string in the tab_strings array to a Frame in tkinter and adds it to the tabs dictionary
It uses the enumerate function to return both the value and index at each point in the tab_strings so that the colours array can be iterated through simultaneously as well
Then by iterating through the tab names, the tk.Frames are appended to the frames array and the frames are all placed in the same place one on top of another. The "pack. Propagate (False)" means that the tab is stopped from auto changing its dimensions based on the widgets inside it
After that it iterates through the tab names again and a name is formed for each frame to represent the button for that frame. For example, the frame "Workouts display" would have the name "Workouts display_button". The "_button" string is added to the name of the key
Then the Button function is called with 3 parameters given:
The name of the tab/ the first part of the button name
The second part of the button name i.e... "_button"
The full combined name of them added together
First the place of each button is calculated by dividing up the height of the option frame by the number of buttons, which is represented by the length of the tab_strings array, or texts array or colours array. It doesn't really matter which one because they all have the same length – because the colours and tab_strings arrays are dynamically created from the text array

The re.findall(r'\d+',first_half_of_name) finds all the numbers in the string. It returns it as a string in an array so the line number = int (number [0]) manipulates it into a number
The next line assigns the full button name string to a Button in tkinter and adds it to the Buttons empty dictionary

The Buttons are placed within the options_frame window and their texts are taken from the pre-defined texts array in the constants file. "Number –1" is used to make sure the right element is accessed because indexes in python start from 0.
The coordinates increase because of the number value so that the buttons are placed neatly one beneath each other
The coordinates are calculated and kept in the coordinates variable which is then used in the y axis of the place () geometry manager.

Profile

```
from constants import *
from GIF import GIF
class Profile():
  def __init__(self):
    self.file = ""
    self.button = tk.Button(tab_search("Profile"))
    self.name = c.execute('''SELECT name FROM Profile''').fetchone()[0]

    def change(file):
      self.file=file
      img = Image.open(file)
      img = img.resize((400, 400))
      self.photo = ImageTk.PhotoImage(img)
      tk.Label(tab_search("Profile"),image=self.photo).place(x=400,y=100)
      c.execute('''UPDATE Profile SET PATH = ? WHERE name = ?''',(file,self.name))
      connection.commit()


    def dialog():
      FILETYPES = [("JPEG files", "*.jpg"),["GIF files","*.gif"]]
      filename = filedialog.askopenfilename(initialdir = os.path.expanduser("~"),title="Select a file",filetypes=FILETYPES)
      if not filename:
        return
      change(filename)

    x = c.execute('''SELECT PATH FROM Profile WHERE name = ?''',(self.name,)).fetchone()
    img_path = x[0]
    change(img_path)

    tk.Button(tab_search("Profile"),text="Change profile picture",command=dialog).pack()
    table_name = "Profile"
    names = c.execute(f'''PRAGMA table_info({table_name})''').fetchall()
    info = c.execute(f'''SELECT name,age,gender,split FROM Profile WHERE name = "{self.name}"''').fetchall()
    x = 50
    y = 50
    names.remove(names[0])
    for i in range(len(info[0])):
      if names[i][1] != "ID":
        tk.Label(tab_search("Profile"),text=f'{names[i][1].capitalize()}  : {info[0][i]}',font=("Helvetica",18,"bold"),foreground="red").place(x=
        y += 100
```

The self.name value is obtained by searching the database
The x variable stores the file path of the profile picture of the user, which is queried from the database
A button to change the profile picture is created
The names variable stores the column names
The info variable stores the user's information: name, age, gender and split
The user's information is iterated through and displayed on the Profile tab
The dialog method uses filedialog to open a dialog box in tkinter so the user can choose the file they want from their computer as a profile picture. The dialog box only shows jpeg and gif files so compatibility with tkinter is already dealt with since the user cannot pick files that aren't compatible with tkinter.
When a valid file it picked and open is pressed the change method is run.
This renders the selected picture to the profile tab and updates the file path in the database so that next time the user loads the application, there chosen profile picture is displayed.

Calendar

```
from constants import *
class Calendar_Display():
  def __init__(self,d):
    self.event_id = 0
    self.date = d
    self.exercises = ast.literal_eval(c.execute(f'''SELECT exercises FROM Summary WHERE date = '{self.date.strftime('%Y-%m-%d')}' ''').fetchall()[0][0])

    self.l = tk.Label(tab_search("Calendar"),text = "Pick the exercise you want to view: 1 - 6: ")
    self.e = tk.Entry(tab_search("Calendar"))
    self.b = tk.Button(tab_search("Calendar"),text="Click to show")


    self.l2 = tk.Label(tab_search("Calendar"))
    self.e2 = tk.Entry(tab_search("Calendar"))
    self.b2 = tk.Button(tab_search("Calendar"),text="Click to decide")

    self.y  = 250
    self.x = 600
    ''' Different ways of formatting date'''
    # dd/mm/YY
    self.formatted2 = datetime.strptime(self.date.strftime("%m/%d/%y"), "%m/%d/%y").date()

    self.day = self.date.strftime('%d')
    self.month = self.date.strftime('%m')
    self.year = self.date.strftime('%Y')
    self.display_Calendar_widget()
    self.revert()
```

This part picture initialises all the attributes of the calendar class:
The exercises attribute finds the absolute list from a string that holds a list
The labels, entries and buttons for the choice of summary are initialised here
The positioning of the labels, entries and buttons are initialised here in the self.x and self.y attributes
The self. formatted2 attribute manipulates the datetime input into something useable in the code
Three methods are instantiated here: Get,add_event and revert

```
 def display_Calendar_widget(self):
   self.calendar = Calendar(
     tab_search("Calendar"),
     selectmode="day",
     month=int(self.month),
     day=int(self.day),
     year=int(self.year),
     selectbackground = "red",
     selectforeground = "white"
     )
   self.calendar.place(x=0,y=0,width=W+300)
   self.calendar.bind("<<CalendarSelected>>", self.handle_date_click)
   self.add_event()
 def delete_event(self):
   date = self.date
   if date is not None:
     events_on_date = self.calendar.calevent_get(date)

     for event in events_on_date:
       self.calendar.calevent_delete(event)
```

The display_Calendar_widget creates and places the tkcalendar widget down In the calendar tab, hence the name of the method. It also binds the choice of any event on the tkcalendar widget to the function self.handle_date_click
The delete_event method removes all events on a given date

```
  def handle_date_click(self, event):
    for i in tab_search("Calendar").winfo_children():
      if i in (self.e,self.e2):
        i.delete(0, 'end')
      elif i in (self.l,self.l2):
        i.destroy()

    def summary_part1():
      val = int(self.e.get())-1
      specific_exercise_instance = Specific_exercise_class()
      specific_exercise_instance.show(self.exercises[val],"Calendar",None,200,self.x-600,400)
      self.l = tk.Label(tab_search("Calendar"),text = "Pick the exercise you want to view: 1 - 6: ")
      self.e = tk.Entry(tab_search("Calendar"))
      self.b = tk.Button(tab_search("Calendar"),text="Click to show")
      self.l2 = tk.Label(tab_search("Calendar"))
      self.e2 = tk.Entry(tab_search("Calendar"))
      self.b2 = tk.Button(tab_search("Calendar"),text="Click to decide")
      self.display_Calendar_widget()
    def showing_on_calendar_tab():
      choice = int(self.e2.get())-1
      select = c.execute(f'''SELECT exercises FROM Summary WHERE date = '{formatted_date}' ''').fetchall()
      self.exercises = [exercise for sublist in select for exercise in sublist]
      self.exercises = ast.literal_eval(self.exercises[choice])

      self.l = tk.Label(tab_search("Calendar"),text = "Pick the exercise you want to view: 1 - 6: ")

      self.e = tk.Entry(tab_search("Calendar"))
      self.b = tk.Button(tab_search("Calendar"),text="Click to show")
      self.l.place(x=self.x,y=self.y)
      self.e.place(x=self.x,y=self.y+50)
      self.b.config(command=summary_part1)
      self.b.place(x=self.x,y=self.y+100)


    from Workout import Specific_exercise_class
    date_string = self.calendar.get_date()
    date_object = datetime.strptime(date_string, "%m/%d/%y").date()
    formatted_date = date_object.strftime('%Y-%m-%d')
    select = c.execute(f'''SELECT exercises FROM Summary WHERE date = '{formatted_date}' ''').fetchall()
    number_of_entries_on_specific_date = len(select)


    self.l2 = tk.Label(tab_search("Calendar"))
    self.e2 = tk.Entry(tab_search("Calendar"))
self.b2 = tk.Button(tab_search("Calendar"),text="Click to decide")

self.l2.config(text = f"Pick the workout you want to view: 1 - {number_of_entries_on_specific_date}: ")
self.l2.place(x=self.x, y=self.y + 200)
self.e2.place(x=self.x, y=self.y + 225)
self.b2.config(command=showing_on_calendar_tab)
self.b2.place(x=self.x, y=self.y + 250)
```

This method handles what happens when any event on the tkcalendar is Pressed. First the entry boxes are cleared, and labels destroyed to make sure, only new data is considered.

The date that is being clicked on is extracted and formatted to a form that can be used.

It is changed into a date object from a datetime object

It is changed from month [2 digit]/day [2 digit] /year [2 digit] to a year [4 digit]/months [2 digit]/ day [2 digit] format

Then all exercises with this date is searched for in the database

The number of records found is saved to variable “number_of_entries_on_specific_date”
Then the l2, e2 and b2 are re-created, configured and placed

Certain values are entered by the user in the GUI, then:
In the showing_on_calendar_tab function:

Whichever workout the user chose affects how the exercise attribute is formatted. If the user chose the seconds workout, then choice would become 1 and the second set of exercise results become exercises

Then l, e, and b are re-created, configured and placed and link to the summary_part1 function
In the summary_part1 function:

The value of the entry box decides which exercise is shown. The index is one lower than the

entered value to keep index integrity [indexes start from 0]

The show function from the Workout file [discussed after this] shows the specific exercise The calendar widget is re-created, and events re-added, and all the label, entries and buttons re-created ready for the next event click

```
def revert(self):
  def jump():
    self.display_Calendar_widget()
  tk.Button(tab_search("Calendar"),text="Click to go back to today",command=jump,background="black",foreground="yellow").place(x=225,y=300)
def add_event(self):
    select = c.execute('''SELECT date from Summary''').fetchall()
    events = []
    for i in select:
      events.append(i[0])
    print(events)
    for i in events:
      self.formatted2 = datetime.strptime(i, "%Y-%m-%d").date()
      self.event_id = self.calendar.calevent_create(self.formatted2, "Red Dot", "dot")
```

The revert function re-creates the calendar and changes it back to the current day

The add_event method searches the database for all available dates and loops through them and adds them to the calendar as events

Dynamic workout display

```
from constants import *
from Calendar import Calendar_Display
from Search_for_Images import *

class Workout_display_class():
  def __init__(self):
    self.images = {}
    self.H = 600
    self.W = 600

    self.inside2 = tk.Frame(tab_search("Workouts display"),height=self.H,width=self.W)
    self.canvas2 = tk.Canvas(self.inside2)

    self.change = tk.StringVar()
    self.var = tk.StringVar()

    self.inside = tk.Frame(tab_search("Workouts display"),height=self.H,width=self.W)
    self.inside.place(x=0,y=0)


    # Add canvas
    self.canvas = tk.Canvas(self.inside)
    self.canvas.pack(side = "left")
    self.canvas.config(width=2/5*self.W, height=self.H,background="yellow")
    # Add scrollbar
    self.scrollbar = tk.Scrollbar(self.inside,orient="vertical",command = self.canvas.yview,highlightthickness=border,highlightbackground="black")
    self.scrollbar.pack(side = "left", fill = "y")
    # Confugure canvas
    self.canvas.config(yscrollcommand = self.scrollbar.set)
    self.canvas.bind('<Configure>',lambda e: self.canvas.configure(scrollregion = self.canvas.bbox("all")))
    # Create another frame inside the canvas
    self.Sub_frame = tk.Frame(self.canvas)
    # Add this new frame to a window in the canvas
    self.canvas.create_window((100,250), window = self.Sub_frame, anchor = "nw")
    self.var = tk.StringVar()

    self.load()
    self.other_class = Specific_exercise_class()

    self.var.trace('w', self.on_radio_button_change)
```

This initialises all the attributes needed for this class. It also instantiates the "load" method and assigns another class to an attribute in this class, so that the other class can be called from this class linking them.

The line self.var.trace('w', self.on_radio_button_change) in the code sets up a trace on the Tkinter variable self.var.

In Tkinter, a trace is a way to associate the callback function with changes to my Tkinter variable.

Here, the trace is set on write operations ('w'), meaning the callback function self.on_radio_button_change will be called whenever the value of self.var is changed.

```
  def load(self):
    self.Sub_frame.pack()
    a = c.execute("SELECT * From Program_Routine")
    all_workouts = a.fetchall()
    for line in all_workouts:
        try:
          img = Image.open(main(line[1]+' Day'+file_extension))
          img = img.resize((100, 100))
          self.images[line[1]] = ImageTk.PhotoImage(img)
          self.l = tk.Radiobutton(self.Sub_frame, text=line[1]+' Day',image=self.images[line[1]],compound=tk.TOP,command=lambda value2=line[1]:
          self.on_radio_button_change(value2), variable=self.var, value=line[1])
          self.l.pack()
        except:
          self.l = tk.Radiobutton(self.Sub_frame, text=line[1]+' Day',command=lambda value2=line[1]: self.on_radio_button_change(value2),
          variable=self.var, value=line[1])
          self.l.pack()
  def on_radio_button_change(self,*args):
    selected_value = self.var.get()
    self.other_class.select(selected_value)
    self.other_class.Specific_workout(selected_value)


from Treeview import display_in_treeview
def x():
  Database = display_in_treeview()
Button = tk.Button(tab_search("Display database in treeview"),text="Click to view all exercises",command=x)
Button.pack()
```

This code displays all the different workout days as a radiobutton with Images

When the radiobutton is clicked the function self.on_radio_button_change is Called with the parameter passed as the name of the workout day for example "Leg" for "Leg Day".

The self.on_radio_button_change function accepts all arguments. The value of the selected_value variable is gotten by finding the current value of the radiobutton and entering it as a parameter into specific functions of the Specific_exercise_class

The last 5 lines initialise the treeview class which is in a separate file and explained later.

```
class Specific_exercise_class(display_in_treeview):
  def __init__(self,**kwargs):
    super().__init__(**kwargs)
    self.H = 600
    self.W = 600
    self.HeadingVar = tk.IntVar()
    self.clicked = tk.StringVar()
    self.indexes = []
    self.inside = tk.Frame(tab_search("Workouts display"),height=self.H,width=self.W)
    self.inside2 = tk.Frame(tab_search("Workouts display"),height=self.H,width=self.W)
    self.canvas2 = tk.Canvas(self.inside2)
    self.change = tk.StringVar()
    self.var = tk.StringVar()


    self.inside3 = tk.Frame(tab_search("Workouts display"),height=self.H,width=self.W)
    self.canvas3 = tk.Canvas(self.inside3)
    self.Sub_frame3 = tk.Frame(self.canvas3)
```

This code initialises all the attributes of the Specific exercise class

```
  def select(self,SELECT):
    for widget in self.canvas2.winfo_children():
      widget.destroy()
    self.inside2.place(x=self.W-450,y=0)

    highest = c.execute('''
    SELECT ID FROM Program_Routine
    WHERE name = ?
    ''',(SELECT,))
    Program_ID = highest.fetchall()[0][0]
    mid_level = c.execute('''
    SELECT ID,name FROM Workout
    WHERE Program_Routine_ID = ?
    ''',(Program_ID,))
    Muscles = mid_level.fetchall()
    self.y = []
    for i in range(len(Muscles)):
      muscle_name = Muscles[i][1]
      # Create the SQL query with a placeholder
      sql_query = "SELECT name FROM Exercise WHERE muscle = ?"
      # Execute the query with the parameter
      x = c.execute(sql_query, (muscle_name,))
      p = x.fetchall()
      for j in p:
        self.y.append(j[0])
      if list(set(self.y)) != self.y:
        self.y = list(set(self.y))
  def add(self):
      current = []
      for i in self.indexes:
        current.append(self.y[i])
      #print(current)
      now = datetime.now()
      now = now + timedelta(days=6)

      if self.HeadingVar.get() == 1:
        find = c.execute(f'''SELECT exercises FROM Summary WHERE date = '{now.strftime('%Y-%m-%d %H:%M:%S')}' ''').fetchall()
        print(find)
        if current != find:
```

The first task this method completes is it clears all the widget inside the "canvas2" window which is contained in the first tab

It then searches for the ID of the workout day in the database and saves it to the variable "highest" which represents the highest level in the hierarchy of the database
Then it searches for the ID and name of all the workouts in that workout day and saves it to the variable "Muscles"

Then every exercise within each of those workouts is iterated through and put in the self.y array

Then the self.y array is checked to make sure there are no repeats by using the set () method of python

```
def add(self):
    current = []
    for i in self.indexes:
      current.append(self.y[i])
    #print(current)
    now = datetime.now()
    #now = now + timedelta(days=6)
    current = json.dumps(current)
    if self.HeadingVar.get() == 1:
      from History import History
      find = c.execute(" SELECT exercises FROM Summary WHERE date = ? ",(now.strftime('%Y-%m-%d'),)).fetchall()
      changed = ast.literal_eval(current)
      for i in changed:
        temporary = c.execute('''SELECT reps,sets,PR_weight,Rep_weight,Enjoyable,How_difficult_it_felt,
        equipment,difficulty,muscle FROM Exercise WHERE name = ?''',(i,)).fetchall()
        temporary = list(temporary[0])
        temporary.insert(0,i)
        temporary.insert(1,now.strftime('%Y-%m-%d'))
        temporary = tuple(temporary)
        c.execute('''INSERT INTO History (exercise,date,reps,sets,PR_weight,Rep_weight,Enjoyable,
        How_difficult_it_felt,equipment,difficulty,muscle) VALUES (?,?,?,?,?,?,?,?,?,?,?)''',temporary)
        connection.commit()


      if current != find:
        query = '''
        INSERT INTO Summary (exercises, date) VALUES (?, ?)
        '''
        values = (current, now.strftime('%Y-%m-%d'))
        c.execute(query, values)
        connection.commit()
      C = Calendar_Display(now)
    else:
      c.execute(f'''
      DELETE FROM Summary
      WHERE date = ?
      ''',(now.strftime('%Y-%m-%d'),))
      c.execute('''DELETE FROM History WHERE date = ?''',(now.strftime('%Y-%m-%d'),))
      C = Calendar_Display(now)
      C.delete_event()
```

Then the add method is started: Firstly, it appends all the exercises with indexes in self. Indexes to current.

Then it sets a certain time through "datetime.now()" [and the commented part afterwards forwards it by 6 days – that's there because it was a straightforward way, I found to obtain other dates while maintaining the same datetime format].

Then the current array is formatted so that it can correctly be passed as a parameter to the SQL query because SQL doesn't except arrays as records so it must to change to JSON form

The next part checks whether the "Your Workout" checkbutton has been pressed.
If it has then it appends the exercises to the database and calendar if they aren't already there and then showing the calendar.

If the "Your Workout" button is deselected, then it deletes those exercises from the database and the calendar. Then shows the calendar

```
def Specific_workout(self,value):

  self.canvas2.pack()
  self.canvas2.config(width=3/5*self.W, height=self.H,background="red")
  # Create another frame inside the canvas
  self.Sub_frame2 = tk.Frame(self.canvas2)
  # Add this new frame to a window in the canvas
  self.canvas2.create_window((250,100), window = self.Sub_frame2, anchor = "ne")


  self.Heading = tk.Checkbutton(self.Sub_frame2, text="Your workout:",variable=self.HeadingVar, onvalue=1, offvalue=0,command=self.add)
  self.Heading.pack()
  self.indexes = []
  new = {}
  for i in self.y:
    probability = c.execute('''SELECT Enjoyable FROM Exercise WHERE name = ?''',(i,)).fetchall()
    new[i]=probability[0][0]
  for i in range(6):
      choose = int(max(list(new.values())))
      key = next(key for key, value in new.items() if value == choose)
      self.num = self.y.index(key)
      self.indexes.append(self.y.index(key))
      self.change.set(key)  # Set the initial value for the radio button
      p = tk.Radiobutton(self.Sub_frame2, text=f'Exercise {i + 1}: {self.change.get()}', variable=self.change,
      value=key, command=lambda value=key: self.show(value,"Workouts display","active",300,600,100), wraplength=200)
      p.pack()

      self.m = tk.Button(self.Sub_frame2, text="Change", background="red", command=lambda value=[key,i]: self.display_dropdown(value))
      self.m.pack()
      del new[key]
```

It configures all the canvas and frames and windows necessary for the specific workout on the same tab

It creates a Checkbutton to be clicked when the workout is done
It randomly generates 6 exercises to do for that workout from the list of exercises in self.y

These exercises are all displayed as radio buttons with corresponding images, and all linked to the function self.show ()

There are buttons beneath each exercise that allow the exercise to be changed through the function display dropdown [discussed below]

```
def display_dropdown(self,value):
  def display():
    for widget in self.Sub_frame2.winfo_children():
      widget.destroy()

    self.Heading = tk.Checkbutton(self.Sub_frame2, text="Your workout:",variable=self.HeadingVar, onvalue=1, offvalue=0,command = self.add)
    self.Heading.pack()
    for i in range(6):
      if i == value[1]:
        self.change.set(self.clicked.get())  # Set the value for the changed exercise
        #print("Value: ",value[1],value[0],",",self.change.get(),", index: ",self.y.index(self.change.get()))

        p = tk.Radiobutton(self.Sub_frame2, text=f'Exercise {i + 1}: {self.change.get()}', variable=self.change,
        value=self.change.get(),command=lambda value=self.change.get(): self.show(value,"Workouts display","active",300,600,100), wraplength=200)
        p.pack()
        self.m = tk.Button(self.Sub_frame2, text="Change", background="red", command=lambda value=[self.y[self.num],i]: self.display_dropdown(value))
        self.m.pack()
        new_index = self.y.index(self.change.get())
        self.indexes[value[1]] = new_index
      else:
        #print(self.change.get(),", index: ",self.y.index(self.change.get()))
        p = tk.Radiobutton(self.Sub_frame2, text=f'Exercise {i + 1}: {self.y[self.indexes[i]]}', variable=self.change,
        value=self.y[self.indexes[i]],command=lambda value=self.y[self.indexes[i]]: self.show(value,"Workouts display","active",300,600,100),
        wraplength=200)
        p.pack()
        self.m = tk.Button(self.Sub_frame2, text="Change", background="red", command=lambda value=[self.y[self.num],i]: self.display_dropdown(value))
        self.m.pack()
  self.clicked.set(value[0])
  drop = tk.OptionMenu(self.Sub_frame2, self.clicked, *self.y)
  drop.pack()
  tk.Button(self.Sub_frame2,text="Confirm",command=display).pack()
  l = tk.Label(self.Sub_frame2)
```

The display dropdown shows all the possible exercises that can be replaced with that exercise in a dropdown menu and allows the user to replace that exercise with whatever one they want.

It then re-creates all the different radio buttons and buttons with the newly replaced exercise(s)

```
def show(self,selection,tab,state,size,x,y):
  tab = tab_search(tab)
  l = tk.Label(tab)
  l.place(x=x,y=y+300)
  s = tk.Label(tab,wraplength=500)
  s.place(x=x+300,y=y+80)
  for widget in self.Sub_frame3.winfo_children():
    widget.destroy()

  self.Sub_frame3 = tk.Frame(self.canvas3)
  self.inside3.place(x=self.W-150,y=0)
  self.canvas3.pack(side = "left")
  self.canvas3.config(width=self.W, height=self.H,background="green")
  self.canvas3.create_window((250,100), window = self.Sub_frame3, anchor = "ne")

  img_path = main(selection+'.jpg')
  path = r"C:\Users\khait\OneDrive\Desktop\Project pictures\GIFs"+selection+'.gif'
  def show_it():
      SQL_command = c.execute(f'''SELECT name,reps,sets,PR_weight,Rep_weight,Enjoyable,How_difficult_it_felt,equipment,difficulty,muscle FROM Exercise
      WHERE name = "{selection}"
        ''')
      info = SQL_command.fetchone()
      if state == "active":
        l.config(text=f'''This exercise: \n Name: {info[0]} \n Works the: {info[9].capitalize()} \n Reps: {info[1]} \n Sets: {info[2]} \n
        Equipment needed: {info[7].capitalize()} \n Difficulty level: {info[8].capitalize()} \n Current Pr Weight: {info[3]} \n Current Rep Weight:
        {info[4]} \n How much you liked the exercise: {info[5]}, \n How difficult you found it: {info[6]}''',wraplength=550)
        l.place(x=x,y=y+300)
      else:
        s.config(text=f'''Summary: \n Name: {info[0]} \n Works the: {info[9].capitalize()} \n Reps: {info[1]} \n Sets: {info[2]} \n
        Equipment needed: {info[7].capitalize()} \n Difficulty level: {info[8].capitalize()} \n Current Pr Weight: {info[3]} \n Current Rep Weight:
        {info[4]} \n How much you liked the exercise: {info[5]}, \n How difficult you found it: {info[6]}''',wraplength=550)
        s.place(x=x+300,y=y+80)

  if state  == "active":
    def To_Treeview():
      from GUI_basics import Frame
      tab_search("Display database in treeview").tkraise()

      self.search_entry.insert(0,selection)
      super(Specific_exercise_class, self).Search()
      super(Specific_exercise_class, self).call()
      super(Specific_exercise_class, self).view()
      super(Specific_exercise_class, self).define()
```

```
        Button = tk.Button(tab,text="Done",command=To_Treeview)
        Button.place(x=x+300,y=y-100)
```

The show function displays an image for a specific exercise and all its relevant information.

It takes 5 parameter that allow the exercise to be decided, the placement of the image and text related to the exercise to be decided, to allow the button to confirm exercise finished to be created or not and change the size of the image

For example, in the calendar summary the done button is blocked from being created
It uses the main () function to obtain the image path

If the done button IS created, then the attributes of treeview are inherited and the user is sent to the treeview page to update their progress

```
 try:
    img = Image.open(img_path)
    img = img.resize((size, size))
    self.photo = ImageTk.PhotoImage(img)
    tk.Label(tab,image=self.photo).place(x=x,y=y)
    show_it()
 except Exception as e:
    try:
      print("Exception: ",e,'sel: ',selection)
      Scrape_and_save_Images_Locally_Execution_Function(False,[selection])
      img = Image.open(img_path)
      img = img.resize((size, size))
      self.photo = ImageTk.PhotoImage(img)
      tk.Label(tab,image=self.photo).place(x=x,y=y)
      show_it()
    except Exception as e:
      print('Exception:',e)
      tk.Label(tab_search(tab),text='Image current unavaiable.....').place(x=350,y=100)
      #show_it()
```

This is the part that opens the image and renders it in the GUI.

Caloric section

```
from constants import *
class Caloric():
  def __init__(self,sex):
    self.E = None
    self.Text = None
    self.Button = None
    self.sex = sex
    self.keys = []
    self.array = []
    self.array2 = []
    self.vals = []
    self.Base()
    self.lbl = tk.Label(tab_search("Caloric stuff"))
    #self.convert()
  def delete(self):
    c.execute('''DELETE FROM Nutrition''')
  def print(self):
    print(c.execute('''SELECT * FROM Nutrition''').fetchall())

  def convert(self):
    pounds = int(input("Enter pounds to be converted to kg"))
    kg = 0.454 * pounds
    tk.Label(tab_search("Caloric stuff"),text=round(kg,3)).pack(side="right")
  def Calorie_split(self):
    self.vals = []
    self.keys = []
    for i in tab_search("Caloric stuff").winfo_children():
      if i not in (self.Text, self.E, self.Button,self.lbl):
          i.destroy()
    split = self.E.get()
    api_url = 'https://api.api-ninjas.com/v1/nutrition?query={}'.format(split)
    response = requests.get(api_url, headers={'X-Api-Key': 'EYubhelGF0jRmCC4ZtZKZg==pWcZsqUXcpnwlFW7'})
    request = response.json()
    for a_set in request:
      for key,value in a_set.items():
        self.keys.append(key)
        self.vals.append(value)
        tk.Label(tab_search("Caloric stuff"),text=f'{key.capitalize()} : {value}',foreground="blue").pack(side="top")
```

Initialises everything

Instantiates “Base” method

Creates a label called self.lbl

Creates a method that clears the Nutrition table

Creates a method that displays all the information in the nutrition table

Creates a method that converts pounds to kg

Create a method that tells the calories in a certain food using an API and saves it to self.keys and self.vals

This one isn't saved to a database yet so requires internet connection to work, which may be a limitation that I will address in due course

```
def insert(self):
  check = '''
    SELECT * FROM Nutrition
    WHERE Name = ? AND Calories = ? AND Serving_Size = ? AND Total_fat = ?
    AND Saturated_fat = ? AND Protein = ? AND Sodium = ? AND Potassium = ?
    AND Cholesterol = ? AND Carbohydrate_TOTAL = ? AND Fiber = ? AND Sugar = ?
  '''
  c.execute(check, self.vals)
  existing_record = c.fetchone()
  if existing_record:
    pass
  else:
    insert_sql = '''
    INSERT INTO Nutrition (Name, Calories, Serving_Size, Total_fat, Saturated_fat, Protein, Sodium, Potassium, Cholesterol, Carbohydrate_TOTAL, Fiber, Sugar)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    '''
    c.execute(insert_sql, self.vals)
    connection.commit()
```

Inserts nutrition split information into database if not already there.

Perhaps as a start towards using less internet connection this the data could be queried whether the data is already in the database before using the API.

The logic at the start of this method could perhaps be used in the "Calorie_split" method

```
def Chart(self):
  k = self.keys
  v = self.vals
  self.values = []
  self.labels = []
  canvas_width =  500
  canvas_height = 500
  # Calculate the center of the canvas
  center_x = canvas_width / 2
  center_y = canvas_height / 2

  tk.Label(tab_search("Caloric stuff"), text="Split").place(x=center_x, y=center_y-200)

  canvas = tk.Canvas(tab_search("Caloric stuff"), width=canvas_width, height=canvas_height)
  canvas.place(x=center_x - 100, y=center_y-100)

  colours = ['red', 'green', 'orange', 'blue', 'yellow', 'white', 'purple', 'pink', 'gold', 'turquoise', 'violet',
             'indigo']
  start = 0
  colour_index = 0
  for index, value in enumerate(v):
      if k[index] != "calories" and k[index] != "serving_size_g":
        if type(value) == float or type(value) == int:
            self.values.append(value)
            self.labels.append(k[index])
  total = 0
  for i in self.array:
    total = total + int(i)
  y = np.array(self.values)
  mylabels = self.labels
  fig, ax = plt.subplots(figsize=(30,30))
  x = ax.pie(y, labels=mylabels,autopct='%1.1f%%')
  # Embed the Matplotlib figure in the Tkinter window
  canvas = FigureCanvasTkAgg(fig, master=tab_search("Caloric stuff"))
  canvas.draw()
  canvas.get_tk_widget().place(x=center_x - 100, y=center_y-100, width=canvas_width, height=canvas_height)
```

This creates a pie chart of the nutrition split

```
  def Caloric_general(self):
    if self.sex == "Male":
      maintain = 2500
      cut = 1900
      bulk = maintain + 2000
      tk.Label(tab_search("Caloric stuff"),text=f'To maintain: {maintain} \n To cut: {cut} \n To bulk: {bulk}').pack(side="bottom")
    else:
      maintain = 2000
      cut = 1400
      bulk = maintain + 1600
      tk.Label(tab_search("Caloric stuff"),text=f' To maintain: {maintain} \n To cut: {cut} \n To bulk: {bulk}').pack(side="bottom")
  def combined(self):
    self.delete()
    self.Calorie_split()
    self.Caloric_general()
    self.insert()
    #self.Recipe()
    self.Chart()
    #self.print()

  def Base(self):
    self.Text = tk.Label(tab_search("Caloric stuff"),text="Enter to get calories of a meal : ")
    self.Text.place(x=0,y=0)
    self.E = tk.Entry(tab_search("Caloric stuff"))
    self.E.place(x=200,y=0)
    self.Button = tk.Button(tab_search("Caloric stuff"),text="Click",command=self.combined)
    self.Button.place(x=0,y=60)
```

The “Caloric_general” method calculates the number of calories required for a bulk, maintaining of weight or cut
The “Combined” method sequentially instantiates all the necessary methods of the class
The “Base” method places all the widgets on the GUI window and binds the self. Button to the “combined” method

Treeview class

```
from constants import *
from Calendar import Calendar_Display
from Graphing import Graphing
class display_in_treeview():
  def __init__(self):
    self.entries = []
    self.search_entry = None
    table_name = "Exercise"
    c.execute(f"PRAGMA table_info({table_name})")
    table_info = c.fetchall()
    self.column_names = [info[1] for info in table_info]
    self.column_names = self.column_names[:-1]
    self.horizontal = 75
    self.vertical = 280
    self.labels = {}
    self.entries = {}
    self.call()
    self.view()
    self.define()
    #self.Record()
```

Initialises all necessary values
Saves column names of Exercise table into an array

```
  def Update_ID(self):
    visible_items = self.my_tree.get_children("")
    i = 1
    for item in visible_items:
        self.my_tree.set(item, column="ID", value=i)
        i += 1
  def call(self):
    SQL = c.execute('''
SELECT name From Program_Routine
''')
    Program_Headers = SQL.fetchall()
    columns = ", ".join(self.column_names)
    params = ", ".join(["?" for _ in self.column_names])
    SQL2 = c.execute(f'''
SELECT {columns} from Exercise
''')

    self.Exercise_Headers = SQL2.fetchall()
```

The “Update_ID” very simply just re-writes all the ID values in the Treeview sequentially

The use of it becomes clearer later in the class

```
def view(self):
  style = ttk.Style()
  # Pick a theme
  style.theme_use("default")
  # Configure our Treeview colours
  style.configure("Treeview",
                  background = "beige",
                  foreground = "black",
                  rowheight = 25,
                  fieldbackground = "silver")
  style.map('Treeview', background = [('selected','steel blue')])
  TreeViewFrame = tk.Frame(tab_search("Display database in treeview"),height=400,width=200,background="indigo")
  TreeViewFrame.place(x=0,y=25)
  self.my_tree = ttk.Treeview(TreeViewFrame)
  # Define out columns
  self.my_tree['columns'] = self.column_names
  # Format our columns
  self.my_tree.column('#0',width=0, minwidth=0)
  column_name_to_index = {name: i for i, name in enumerate(self.column_names)}
  width = (1000-75-250)//len(self.column_names)
  for name in self.column_names:
    if name == "ID":
      column_index = column_name_to_index[name]
      self.my_tree.column(column_index, anchor="center", width=75)
    elif name == "name":
      column_index = column_name_to_index[name]
      self.my_tree.column(column_index, anchor="center", width=250)
    else:
      column_index = column_name_to_index[name]
      self.my_tree.column(column_index, anchor="center", width=width)
# Create Headings
self.my_tree.heading("#0", text=" ")
for index, heading in enumerate(self.column_names):
  self.my_tree.heading(index, text=heading)

# Create striped row tags
self.my_tree.tag_configure('odd_row',background="light green")
self.my_tree.tag_configure('even_row',background="gold")

count = 0
for i in self.Exercise_Headers:
  v = ()
  for j in range(len(self.column_names)):
    if type(i[j]) == str:
      v += (i[j].capitalize(),)
    else:
      v += (i[j],)
  if count % 2 == 0:
    self.my_tree.insert(parent = '', index='end',iid=count, text=" ",values=v,tags=('even_row',))
    self.my_tree.pack()
  else:
    self.my_tree.insert(parent = '', index='end',iid=count, text=" ",values=v,tags=('odd_row',))
    self.my_tree.pack()
  count += 1
#my_tree.insert(parent = '1', index='end',iid=151,text="Parent",values=("hamstrings","horizontal leg press",151,55,20,"Giant 8000kg barbell",
self.Update_ID()
```

This creates the Treeview

It loops through the column names and assigns them as headers

It loops through the exercises and each exercises different characters are saved to a repeatedly emptying tuple list called “v”

```
def Update_row_colors(self):
  row_index = 0
  for iid in self.my_tree.get_children(""):
    # Alternate the row colors based on the row_index
    tag = "even_row" if row_index % 2 == 0 else "odd_row"

    # Set the tag of the Treeview item to the appropriate tag
    self.my_tree.item(iid, tags=(tag,))

    # Increment the row index for the next iteration
    row_index += 1
def Search(self):
  query = self.search_entry.get()
  self.my_tree.selection_remove(self.my_tree.selection())

  # Search and highlight matching items
  for item in self.my_tree.get_children():
      values = self.my_tree.item(item, "values")
      if query.lower() in " ".join(map(str, values)).lower():
          self.my_tree.selection_add(item)
          self.my_tree.see(item)

  #self.search_entry.delete(0,'end')
```

The “Update_row_colours” loops through all the Treeview’s “children” which just means all the records and assigns them alternating tags again to keep the striped colour scheme consistent

The “Search” method searches for a specific exercise record. It lowercases the query and finds all the exercises that contain the query so if the exercise name is incomplete when searched for the exercise is still able to be found

```
def Up(self):
  rows = self.my_tree.selection()
  for row in rows:
    self.my_tree.move(row,self.my_tree.parent(row),self.my_tree.index(row)-1)
  self.Update_ID()
  self.Update_row_colors()
def Down(self):
  rows = self.my_tree.selection()
  for row in reversed(rows):
    self.my_tree.move(row,self.my_tree.parent(row),self.my_tree.index(row)+1)
  self.Update_ID()
  self.Update_row_colors()
def Clear(self):
  for key,value in self.entries.items():
    getattr(value,'delete')(0,'end')

def Delete(self):
  select = self.my_tree.selection()
  previously_selected = select[-1]
  self.my_tree.delete(*select)
  values = self.my_tree.item(item, "values")
  c.execute("DELETE FROM Exercise WHERE ID = ?", (values[0],))
  connection.commit()
  self.Update_row_colors()
  self.Update_ID()
  self.my_tree.focus(previously_selected)
```

This adds functionality to the treeview
The user can move records up or down
The user can clear the entry boxes
The user can delete records

```
def Update(self):
  selected = self.my_tree.focus()
  new_attributes = ()
  existing_attributes = self.my_tree.item(selected, "values")
  for i in self.entries.items():
    x = getattr(i[1],'get')()
    if x != "":
      new_attributes+= (x,)
    else:
      new_attributes += (existing_attributes[len(new_attributes)],)
  selected_item = self.my_tree.selection()
  self.my_tree.item(selected,text="",values=new_attributes)

  formatted = []
  table_name = "Exercise"
  c.execute(f"PRAGMA table_info({table_name})")
  table_info = c.fetchall()
  for index,column_name in enumerate(self.column_names):
    data_type = table_info[index][2]
    #print(data_type)
    if data_type == "INTEGER" or data_type == "FLOAT":
      if new_attributes[index] == "":
        formatted.append(f"{column_name} = None,")
      else:
        formatted.append(f"{column_name} = {new_attributes[index]},")

    else:
      if new_attributes[index] == "":
        formatted.append(f" {column_name} = 'None', ")
      else:
        formatted.append(f"{column_name} = '{new_attributes[index]}',")
  formatted = '\n'.join(formatted)
  formatted = formatted[:len(formatted)-1]
  #print(new_attributes[0])
  c.execute(f'''UPDATE Exercise SET {formatted} WHERE ID = {new_attributes[0]}''')
  connection.commit()
```

This one updates the database and the Treeview when the user changes a certain characteristic about a record

```
def Add(self):
  attributes = ()
  for i in self.entries.items():
    x = getattr(i[1],'get')()
    attributes += (x,)
  parent_item = self.my_tree.selection()  # Get the selected parent item
  if parent_item:
    index = self.my_tree.index(parent_item) + 1  # Insert below the selected parent item
    self.my_tree.insert("", index, text="New record", values=attributes)
    self.Update_row_colors()
    self.Update_ID()
  else:
    self.my_tree.insert("", index='end', text="New record", values=attributes)
    self.Update_row_colors()
    self.Update_ID()
def Record(self, e):
  self.Clear()
  found = self.my_tree.focus()
  if found:
      values = self.my_tree.item(found, 'values')

      for index, entry in self.entries.items():
        if index == 1:
          continue
        if index - 1 < len(values):
            entry.delete(0, tk.END)
            entry.insert(0, values[index - 1])
```

The "Add" method allows a user to add an exercise. If they don't select where they want the exercise to be added it simply adds at the bottom of the Treeview. If the user selects a certain record, it inserts the new record underneath that one

```
def define(self):

  for i in self.column_names:
    if i.isupper():
      self.labels[self.column_names.index(i)+1]=i+'_label'
      self.entries[self.column_names.index(i)+1] = tk.Entry(tabs['tab3'],width=20)
    else:
      self.labels[self.column_names.index(i)+1]=i.capitalize()+'_label'
      self.entries[self.column_names.index(i)+1] = tk.Entry(tabs['tab3'],width=20)

    def column(arr):
      ox = self.horizontal
      oy = self.vertical + 60
      num_columns = 2
      nums = []
      label_items = list(arr.items())
      entry_items = list(self.entries.items())
      for key,value in label_items:
        j = key-1
        nums.append((ox, oy))
        ox += 300

        if j % (num_columns) == num_columns - 1:
          oy += 45
          ox = self.horizontal
        if self.column_names[key-1] == "ID":
          value = tk.Label(tab_search("Display database in treeview"),text=f"{self.column_names[key-1]} : ")
          self.entries[key]= tk.Entry(tab_search("Display database in treeview"),width=25)
        else:
          value = tk.Label(tab_search("Display database in treeview"),text=f"{self.column_names[key-1]} : ")
          self.entries[key]= tk.Entry(tab_search("Display database in treeview"),width=25)
          value.place(x = nums[key-1][0]-60,y = nums[key-1][1])
          self.entries[key].place(x = nums[key-1][0] + 20, y = nums[key-1][1] )
```

This dynamically places the entry boxes under the Treeview to allow the user to edit records

```
ox = self.horizontal
oy = self.vertical
column(self.labels)
little = 20
Delete_Button = tk.Button(tab_search("Display database in treeview"),text="Delete Exercise",command= self.Delete)
Delete_Button.place(x=ox-60,y=oy+little)
Clear_Button = tk.Button(tab_search("Display database in treeview"),text="Clear entry boxes",command= self.Clear)
Clear_Button.place(x=ox+75,y=oy+little)
Move_Up_Button = tk.Button(tab_search("Display database in treeview"),text="Move up",command= self.Up)
Move_Up_Button.place(x=ox+205,y=oy+little)
Move_Down_Button = tk.Button(tab_search("Display database in treeview"),text="Move down",command= self.Down)
Move_Down_Button.place(x=ox +290,y=oy+little)
Update_Button = tk.Button(tab_search("Display database in treeview"),text="Update values",command= self.Update)
Update_Button.place(x=ox+380,y=oy+little)
Add_Exercise_Button = tk.Button(tab_search("Display database in treeview"),text="Add Exercise",command= self.Add)
Add_Exercise_Button.place(x=ox+500,y=oy+little)
tk.Label(tab_search("Display database in treeview"),text="Click update button twice to confirm update",wraplength=100).place(x = ox+630,y=oy+(3*little))

self.search_entry = tk.Entry(tab_search("Display database in treeview"))
self.search_entry.place(x=ox+700,y=oy+little)
search_button = tk.Button(tab_search("Display database in treeview"),text="Search",command=self.Search)
search_button.place(x=ox+620,y=oy+little)
self.my_tree.bind("<ButtonRelease-1>",self.Record)
self.my_tree.bind("<<TreeviewSelect>>", lambda event: self.Update_row_colors)
```

This places all the functionality widgets directly beneath the Treeview and binds the selecting of a record to self. Record method.

Graphing

```
from constants import *
class Graphing():
  def __init__(self):
    self.start = 0
    self.reps_list = []
    self.pr_weights_list = []
    self.rep_weights_list = []
    self.sets_list = []
    self.exercises = []
    self.options = ["Reps against exercises","Reps against rep weight", "Reps against PR weight","Rep weight against exercises","PR weight against exercises"
    self.var = tk.StringVar(tab_search("Workout graph summaries"))
    self.shown = tk.Button(tab_search("Workout graph summaries"), text="Click to show graph", command=lambda: self.choice(1))
    self.shown.place(x=0,y=0)
    self.down = tk.Button(tab_search("Workout graph summaries"),text="           Down                  ")
    self.down.config(command=lambda: self.choice(2))
    self.res = tk.Button(tab_search("Workout graph summaries"),text="reset",command=self.go_back)
    self.res.place(x=0,y=400)
  def draw(self,values,labels,x_axis,y_axis):
    plt.close('all')
    fig, ax = plt.subplots(figsize=(5, 5))
    bars = ax.barh(labels, values, color='skyblue')

    # Add labels and values to the bars
    for bar in bars:
      yval = bar.get_width()
      plt.text(yval, bar.get_y() + bar.get_height()/2, round(yval, 2), va='center')

    ax.set_xlabel(x_axis)
    ax.set_ylabel(y_axis)

    plt.yticks(rotation=45, ha='right',fontsize=8)

    # Embed the Matplotlib figure in the Tkinter window
    canvas_width =  800
    canvas_height = 500

    # Calculate the center of the canvas
    center_x = canvas_width / 2
    center_y = canvas_height / 2
    canvas = FigureCanvasTkAgg(fig, master=tab_search("Workout graph summaries"))
    canvas.draw()
    canvas.get_tk_widget().place(x=center_x-300, y=center_y - 100, width=canvas_width, height=canvas_height)
```

The constructor initialises everything

The draw method draws a horizontal bar chart
It takes 4 inputs, the values its plotting, the categories it's plotting, the name of the x axis and the name of the y axis. [In this case the y axis acts as the conventional x axis as is known by most due to it being a horizontal bar graph.]

```
def search(self):
  self.reps_list = []
  self.pr_weights_list = []
  self.rep_weights_list = []
  self.sets_list = []

  select = c.execute(f'''SELECT name FROM Exercise ''').fetchall()
  number_of_entries_on_specific_date = len(select)
  self.exercises = []
  for i in range(number_of_entries_on_specific_date):
    self.exercises.append(select[i][0])
  for p in self.exercises:
    self.reps_list.append(c.execute(f'''SELECT reps FROM Exercise WHERE name = "{p}"''').fetchone()[0])
    self.pr_weights_list.append(c.execute(f'''SELECT PR_weight FROM Exercise WHERE name = "{p}"''').fetchone()[0])
    self.rep_weights_list.append(c.execute(f'''SELECT Rep_weight FROM Exercise WHERE name = "{p}"''').fetchone()[0])
    self.sets_list.append(c.execute(f'''SELECT sets FROM Exercise WHERE name = "{p}"''').fetchone()[0])
```

This method saves the reps, weight and sets of each exercise to a local list

```
def move(self,number):
  if number == 1:
    if self.start == 0:
      def run():
        self.draw(self.reps_list[0:15],self.exercises[0:15],"Reps","Exercises")
      self.start = self.start +15
      run()
      self.down.place(x=0,y=0)
    else:
      def run():
        self.draw(self.reps_list[self.start:self.start+15],self.exercises[self.start:self.start+15],"Reps","Exercises")
      self.start = self.start +15
      run()
  elif number == 2:
    if self.start == 0:
      def run():
        self.draw(self.reps_list[0:15],self.rep_weights_list[0:15],"Reps","Rep weights")
      self.start = self.start +15
      run()
      self.down.place(x=0,y=0)
    else:
      def run():
        self.draw(self.reps_list[self.start:self.start+15],self.rep_weights_list[self.start:self.start+15],"Reps","Rep weights")
      self.start = self.start +15
      run()
  elif number == 3:
    if self.start == 0:
      def run():
        self.draw(self.reps_list[0:15],self.pr_weights_list[0:15],"Reps","PR weights")
      self.start = self.start +15
      run()
      self.down.place(x=0,y=0)
    else:
      def run():
        self.draw(self.reps_list[self.start:self.start+15],self.pr_weights_list[self.start:self.start+15],"Reps","PR weights")
      self.start = self.start +15
      run()
  elif number == 4:
    if self.start == 0:
      def run():
        self.draw(self.rep_weights_list[0:15],self.exercises[0:15],"Rep weight","Exercises")
      self.start = self.start +15
      run()
      self.down.place(x=0,y=0)
elif number== 5:
  if self.start == 0:
    def run():
      self.draw(self.pr_weights_list[0:15],self.exercises[0:15],"PR weight","Exercises")
    self.start = self.start +15
    run()
    self.down.place(x=0,y=0)
  else:
    def run():
      self.draw(self.pr_weights_list[self.start:self.start+15],self.exercises[self.start:self.start+15],"PR weight","Exercises")
    self.start = self.start +15
    run()
elif number== 6:
  if self.start == 0:
    def run():
      self.draw(self.sets_list[0:15],self.pr_weights_list[0:15],"Sets","PR weights")
    self.start = self.start +15
    run()
    self.down.place(x=0,y=0)
  else:
    def run():
      self.draw(self.sets_list[self.start:self.start+15],self.pr_weights_list[self.start:self.start+15],"Sets","PR weights")
    self.start = self.start +15
    run()
elif number== 7:
  if self.start == 0:
    def run():
      self.draw(self.sets_list[0:15],self.rep_weights_list[0:15],"Sets","Rep weights")
    self.start = self.start +15
    run()
    self.down.place(x=0,y=0)
  else:
    def run():
      self.draw(self.sets_list[self.start:self.start+15],self.rep_weights_list[self.start:self.start+15],"Sets","Rep weights")
    self.start = self.start +15
    run()
  elif number== 8:
    if self.start == 0:
      def run():
        self.draw(self.sets_list[0:15],self.reps_list[0:15],"Sets","Reps")
      self.start = self.start +15
      run()
      self.down.place(x=0,y=0)
    else:
      def run():
        self.draw(self.sets_list[self.start:self.start+15],self.reps_list[self.start:self.start+15],"Sets","Reps")
      self.start = self.start +15
      run()
  elif number== 9:
    if self.start == 0:
      def run():
        self.draw(self.sets_list[0:15],self.exercises[0:15],"Sets","Exercises")
      self.start = self.start +15
      run()
      self.down.place(x=0,y=0)
    else:
      def run():
        self.draw(self.sets_list[self.start:self.start+15],self.exercises[self.start:self.start+15],"Sets","Exercises")
      self.start = self.start +15
      run()
  else:
    pass
```

This creates 9 viable options of graphs that can be created with a down button that when pressed moves down to show the following data, simulating the scrolling function on

websites

```
  def go_back(self):
    self.search()
    self.start = 0
    self.move(self.options.index(self.var.get())+1)
  def select(self,variable):
    self.go_back()
  def choice(self,loop):
    if loop == 1:
      self.var.set("Make a choice")
      dropdown_menu = tk.OptionMenu(tab_search("Workout graph summaries"), self.var, *self.options,command=self.select)
      dropdown_menu.place(x=700,y=50)
    else:
      self.move(self.options.index(self.var.get())+1)
```

The "go_back" method does the whole information searching again and that creates the graph based on the option the user picked (explained further below)

The choice method says that if the loop value is 1 then it creates a dropdown menu of the 9 possible graphs for the user to pick

If the choice isn't 1 then it simply runs the move method again which is simply the effect of pressing the down button

Search for Images file

```
from constants import *


def Find_Images(x):
    entry = ''
    special_characters = '!@#$%^&*()-_+=[]{}|;:,.<>?/\\'
    special_characters = string.punctuation + special_characters
    special_characters = [char for char in special_characters]
    for i in special_characters:
        if i in x:
            entry = x.replace(i,' ').replace(' ', '+')
    else:
        entry = x.replace(' ', '+')
    url = f'https://uk.images.search.yahoo.com/search/images;_ylt=AwrkPPD3O8Zkjaclgj8M341Q;_ylu=Y29sbwNpcjIEcG9zAzEEdnRpZAMEc2VjA3BpdnM-?p={entry}+gym+exerci
    page = requests.get(url)
    soup = BeautifulSoup(page.content,'html.parser')
    holder = soup.find('img')
    if holder is not None:
        if holder and 'src' in holder.attrs:
            img_url = holder['src']
        elif holder and 'data-src' in holder.attrs:
            img_url = holder['data-src']
        else:
                print(f"Image not found for {entry}")

        holder['alt'] = f'{x}'
        try:
            img_url = holder['data-src']
            name = holder['alt']+file_extension
            urllib.request.urlretrieve(img_url,f'{base_path}/{name}')
        except:
            try:
                img_url = holder['src']
                name = holder['alt']+'.jpg'
                urllib.request.urlretrieve(img_url,f'{base_path}/{name}')
            except Exception as e:
                print(e)
def Scrape_and_save_Images_Locally_Execution_Function(Fill,trial):
  connection = sqlite3.connect("My database")
  c = connection.cursor()
  exer = c.execute('''
  SELECT name FROM Exercise
  ''').fetchall()
  threads = []
  if Fill == True:
      for i in exer:
        t = threading.Thread(target=Find_Images,args=(i[0],))
        threads.append(t)
        t.start()

      for j in threads:
          j.join()
  else:
      for i in trial:
        t = threading.Thread(target=Find_Images,args=(i,))
        threads.append(t)
        t.start()

      for j in threads:
          j.join()
```

This uses threading to scrape the internet for images by dynamically modifying the URL of the searched image and dynamically downloading and saving the image to a specified directory

History file

```
from constants import *
def History():
    x = c.execute('''SELECT reps,sets,PR_weight,Rep_weight,Enjoyable,How_difficult_it_felt,equipment,difficulty,muscle FROM History''').fetchall()
    text_widget = tk.Text(tab_search("History"), wrap="none")

    # Insert the data into the Text widget
    for i in range(len(x)):
      if i == 0:
        text_widget.insert("end"," ".join(map(str, x[i])))
      else:
        text_widget.insert("end", "\n" + " ".join(map(str, x[i])))

    # Add the Text widget to the window
    text_widget.pack(side="top", fill="both", expand=True)

    # Create a scrollbar for the Text widget
    scrollbar = tk.Scrollbar(tab_search("History"), command=text_widget.yview)
    scrollbar.pack(side="right", fill="y")
    text_widget.configure(yscrollcommand=scrollbar.set)

History()
```

Main file

```
from basics import *
from login import login
from start import start
from Register import GUI_login,login_window




#c.execute('''DELETE FROM PROFILE''')
available = c.execute('''SELECT * FROM Profile''').fetchall()

if available:
  GUI_login(2)
else:
  GUI_login(1)


login_window.mainloop()
```

This file simply runs the application

Without this the application wouldn't work

GUI Login file

```
from constants import root,W,H
from PIL import Image,ImageTk, Image, ImageSequence
from basics import *
import tkinter as tk
root.withdraw()
login_window = tk.Toplevel(root)
login_window.geometry(f'{W}x{H}')
login_window.lift()

class GUI_login():
  def __init__(self,choice):
    self.choice = choice
    self.img = None
    self.img2 = None
    self.photo_for_login = None
    self.photo_for_registering = None
    self.image = None
    self.image2 = None
    if self.choice == 1:
      self.method1()
    elif self.choice == 2:
      self.method2()
    else:
      pass
  def run_it(self):
    from start import start
    start()
  def method1(self):
    def SignUp():
      name = name_entry_box.get()
      age = age_entry_box.get()
      gender = gender_entry_box.get()
      password = password_entry_box.get()
      try:
        name = str(name)
        gender = str(gender)
        password = str(password)
        c.execute(f'''
        INSERT INTO Profile (name,age,gender,split,password,PATH) VALUES ("{name}",{age},"{gender}","Five Day Split","{password}","{find_file("SignIn.jpg")}")
        ''')
        connection.commit()
        root.quit()
        self.run_it()
      except ValueError:
        name_entry_box.delete(0, "end")
        age_entry_box.delete(0, "end")
        gender_entry_box.delete(0, "end")
        password_entry_box.delete(0, "end")
        self.method1()
    x = 500
    y = 50

    tk.Label(login_window,text="Sign up: ",font=("Helvetica",18,"bold")).place(x=x,y=y)

    name_lbl = tk.Label(login_window,text="Name: ",font=("Helvetica",18,"bold"),foreground="red")
    name_lbl.place(x=x - 150,y = y + 50)

    name_entry_box = tk.Entry(login_window,width=30)
    name_entry_box.place(x=x,y = y + 60)


    age_lbl = tk.Label(login_window,text="Age: ",font=("Helvetica",18,"bold"),foreground="red")
    age_lbl.place(x=x - 150,y = y + 100)

    age_entry_box = tk.Entry(login_window,width=30)
    age_entry_box.place(x=x,y = y + 110)


    gender_lbl = tk.Label(login_window,text="Gender: ",font=("Helvetica",18,"bold"),foreground="red")
    gender_lbl.place(x=x - 150,y = y + 150)

    gender_entry_box = tk.Entry(login_window,width=30)
    gender_entry_box.place(x=x,y = y + 160)


    password_lbl = tk.Label(login_window,text="Password: ",font=("Helvetica",18,"bold"),foreground="red")
    password_lbl.place(x=x - 150,y = y + 200)

    password_entry_box = tk.Entry(login_window,width=30)
    password_entry_box.place(x=x,y = y + 210)

    SignUp_Confirm_Button = tk.Button(login_window,text="Sign up",font=("Helvetica",15,"bold"),foreground="blue",command=SignUp)
    SignUp_Confirm_Button.place(x=x+5,y= y + 250)
```

```
def method2(self):
  def LogIn():
    name = name_entry_box2.get()
    password = password_entry_box2.get()
    finding_from_profile = c.execute(f'''
    SELECT * FROM Profile WHERE name = ? AND password = ?
    ''',(name,password)).fetchall()
    if finding_from_profile:
      root.quit()
      self.run_it()
    else:
      name_entry_box2.delete(0, "end")
      password_entry_box2.delete(0, "end")
      self.method2()


  x2 = 500
  y2 = 50

  tk.Label(login_window,text="Login: ",font=("Helvetica",18,"bold")).place(x=x2,y=y2)

  name_lbl2 = tk.Label(login_window,text="Name: ",font=("Helvetica",18,"bold"),foreground="red")
  name_lbl2.place(x=x2 - 100,y = y2 + 50)

  name_entry_box2 = tk.Entry(login_window,width=30)
  name_entry_box2.place(x=x2,y = y2 + 60)


  password_lbl2 = tk.Label(login_window,text="Password: ",font=("Helvetica",18,"bold"),foreground="red")
  password_lbl2.place(x=x2 - 150,y = y2 + 100)

  password_entry_box2 = tk.Entry(login_window,width=30)
  password_entry_box2.place(x=x2,y = y2 + 110)

  Login_Confirm_Button2 = tk.Button(login_window,text="Login",font=("Helvetica",15,"bold"),foreground="blue",command=LogIn)
  Login_Confirm_Button2.place(x=x2+5,y= y2 + 250)
```

This is simply the logic for the placement of the Signup and Login system within a separate window from the main application that opens first.

It also inserts the user's credentials into a database when signing up and queries from the database to validate the user's credentials when logging in.

The window is destroyed when the sign up or login is successful

It also hides the main application window, so it appears like only the login window / signup window has opened

Start file

```
from shared import complete
from constants import *
from GUI_basics import *
from History import History
from Profile import Profile
from Workout import Workout_display_class
from Calendar import Calendar_Display
from Cal import Caloric
from Treeview import display_in_treeview
from Graphing import Graphing
from Search_for_Images import *

def start():
    print("Starting appplication")
    root.deiconify()
    root.lift()
    P = Profile()
    tab_search("Workouts display").tkraise()
    Workout = Workout_display_class()
    now = datetime.now()
    Cal = Calendar_Display(now)
    Cal.display_Calendar_widget()
    C = Caloric("Male")
    G = Graphing()
    G.search()
    #Scrape_and_save_Images_Locally_Execution_Function(True,0)
    #Scrape_and_save_Images_Locally_Execution_Function(False,['Arms Day','Shoulders Day','Legs Day','Back Day','Full body Day'])
    #Scrape_and_save_Images_Locally_Execution_Function(False,['Legs Day'])
    try:
        root.mainloop()
    except Exception as e:
        print(f"An error occurred: {e}")
```

At this point, the login / signup window has been destroyed. The main application is then showed again through the "root.deiconify" command
This file simply initialises and imports all the other necessary files in the right order so that the program can run correctly

Database Creation file

```
import sqlite3
import requests
def Database():
  print("Please wait while we set up your database, this may take some time")
  connection = sqlite3.connect("My database")
  c = connection.cursor()
  c.execute("PRAGMA foreign_keys = ON")
  c.execute('''
  CREATE TABLE IF NOT EXISTS History (
  ID INTEGER PRIMARY KEY,
  exercise TEXT,
  date DATETIME,
  reps INTEGER,
  sets INTEGER,
  PR_weight FLOAT,
  Rep_weight FLOAT,
  Enjoyable INTEGER,
  How_difficult_it_felt INTEGER,
  equipment TEXT,
  difficulty TEXT,
  muscle TEXT
  )
  ''')
  c.execute('''
  CREATE TABLE IF NOT EXISTS Summary (
  ID INTEGER PRIMARY KEY,
  exercises TEXT,
  date DATETIME
  )
  ''')
  c.execute('''
    CREATE TABLE IF NOT EXISTS Nutrition (
    ID INTEGER PRIMARY KEY,
    Name TEXT,
    Calories FLOAT,
    Serving_Size FLOAT,
    Total_fat FLOAT,
    Saturated_fat FLOAT,
    Protein FLOAT,
    Sodium FLOAT,
    Potassium FLOAT,
    Cholesterol FLOAT,
    Carbohydrate_TOTAL FLOAT,
    Fiber FLOAT,
    Sugar FLOAT
    )
    ''')
  c.execute('''CREATE TABLE IF NOT EXISTS Profile (
      ID INTEGER PRIMARY KEY,
      name TEXT,
      age INTEGER,
      gender TEXT,
      split TEXT,
      password TEXT,
      PATH TEXT
)''')
  c.execute('''
  CREATE TABLE IF NOT EXISTS Split (
  ID INTEGER PRIMARY KEY ,
  name TEXT
  )''')
  c.execute('''CREATE TABLE IF NOT EXISTS Program_Routine (
  ID INTEGER PRIMARY KEY,
  name TEXT,
  Number_of_workouts_completed INTEGER,
  Split_ID INTEGER,
  FOREIGN KEY (Split_ID) REFERENCES Split(ID)
  )''')
  c.execute('''CREATE TABLE IF NOT EXISTS Workout (
  ID INTEGER PRIMARY KEY,
  name TEXT,
  Program_Routine_ID INTEGER,
  FOREIGN KEY (Program_Routine_ID) REFERENCES Program_Routine(ID)
  )''')
  c.execute('''CREATE TABLE IF NOT EXISTS Exercise (
  ID INTEGER PRIMARY KEY,
  name TEXT,
  reps INTEGER,
  sets INTEGER,
  PR_weight FLOAT,
  Rep_weight FLOAT,
  Enjoyable INTEGER,
  How_difficult_it_felt INTEGER,
  equipment TEXT,
  difficulty TEXT,
  muscle TEXT,
  Workout_ID INTEGER,
  FOREIGN KEY (Workout_ID) REFERENCES Workout(ID)
  )
  ''')
```

The two screenshots above represent the creation of a database called “My database” and the creation of the tables within this database and what values each record in each table in this database will hold.

```
  def m(option,muscle):
    url = "https://api.api-ninjas.com/v1/exercises"
    global response
    response = requests.get(f'{url}?{option}={muscle}',headers={'X-Api-Key': 'EYubhelGF0jRmCC4ZtZKZg==pWcZsqUXcpnwlFW7'})
    global request
    request = response.json()
    for exercise in request:
      exercise.setdefault('reps', 12)
      exercise.setdefault('sets', 3)
      exercise.setdefault('PR_weight', 0.0)
      exercise.setdefault('Rep_weight', 0.0)
      exercise.setdefault('Enjoyable', 0)
      exercise.setdefault('How_difficult_it_felt', 0)
    return request
  arr = ["muscle","type"]
  arr2 = ["glutes","biceps","forearms","traps","hamstrings","triceps","calves","quadriceps","abductors","lats","lower_back","middle_back","abdominals","cardi

  c.execute('''
  INSERT INTO Split VALUES (
  0,"Five Day Split"
  )
  ''')
  class insert():
    def __init__(self,indexes_of_workouts_array,Program_ID,Program_Name):
      self.index = indexes_of_workouts_array
      self.Program_ID = Program_ID
      self.Program_Name = Program_Name
      self.current = [m(arr[1],arr2[self.index[0]])]
      for i in range(len(self.index)-1):
        self.current.append(m(arr[0],arr2[self.index[i]]))

    def iterate(self):
      c.execute(f'''
      INSERT INTO Program_Routine VALUES (
      {self.Program_ID},"{self.Program_Name}",0,0
      )
      ''')

      connection.commit()

      for i in range(len(self.current)):
        c.execute(f'''
        INSERT INTO Workout (name,Program_Routine_ID)VALUES(
       "{arr2[self.index[i]]}",{self.Program_ID}
        )
        ''')
        connection.commit()
        for j in range(len(self.current[i])):
          c.execute(f'''
          INSERT INTO Exercise (name,reps,sets,PR_weight,Rep_weight,Enjoyable,How_difficult_it_felt,equipment,difficulty,muscle,Workout_ID)VALUES(
          "{self.current[i][j]['name']}",{self.current[i][j]['reps']},{self.current[i][j]['sets']},{self.current[i][j]['PR_weight']},{self.current[i][j]['Rep
          )
          ''')
          connection.commit()


  arr2 = ["glutes","biceps","forearms","traps","hamstrings","triceps","calves","quadriceps","abductors","lats","lower_back","middle_back","abdominals","cardi

  Program_Days = {
    "Legs": [0,4,6,7,8], "Arms": [1,2,5], "Back": [9,10,11], "Shoulders": [3], "Full body": [1,3,4,5,7,9,11,12],"Chest":[16]
  }

  nums = []
  nums.append(0)
  count = 0
  for i,value in enumerate(Program_Days):
    count = count + len(Program_Days[value])
    nums.append(count)
  nums.pop()
  workouts_IDs = []
  for j,value2 in enumerate(Program_Days):
    for z,value3 in enumerate(Program_Days[value2]):
      workout_ID = z+nums[j]
      workouts_IDs.append(workout_ID)
      program_ID = j

  for p,value4 in enumerate(Program_Days):
    ins=  insert(Program_Days[value4],p,value4)
    ins.iterate()
Database()
```

This dynamically populates the database with data from the Workout API sorting the data into the 4 tables specified above - [Split, Program_Routine, Workout, Exercise]

Implementation techniques:

Inheritance

```
class Workout_display_class():
  def __init__(self):
    self.images = {}
    self.H = 600
    self.W = 600

    self.inside2 = tk.Frame(tab_search("Workouts display"),height=self.H,width=self.W)
    self.canvas2 = tk.Canvas(self.inside2)

    self.change = tk.StringVar()
    self.var = tk.StringVar()

    self.inside = tk.Frame(tab_search("Workouts display"),height=self.H,width=self.W)
    self.inside.place(x=0,y=0)


    # Add canvas
    self.canvas = tk.Canvas(self.inside)
    self.canvas.pack(side = "left")
    self.canvas.config(width=2/5*self.W, height=self.H,background="yellow")
    # Add scrollbar
    self.scrollbar = tk.Scrollbar(self.inside,orient="vertical",command = self.canvas.yview,highlightthickness=border,highlightbackground="black")
    self.scrollbar.pack(side = "left", fill = "y")
    # Confugure canvas
    self.canvas.config(yscrollcommand = self.scrollbar.set)
    self.canvas.bind('<Configure>',lambda e: self.canvas.configure(scrollregion = self.canvas.bbox("all")))
    # Create another frame inside the canvas
    self.Sub_frame = tk.Frame(self.canvas)
    # Add this new frame to a window in the canvas
    self.canvas.create_window((100,250), window = self.Sub_frame, anchor = "nw")
    self.var = tk.StringVar()
```

```
class Specific_exercise_class(display_in_treeview):
  def __init__(self,**kwargs):
    super().__init__(**kwargs)
      def To_Treeview():
        from GUI_basics import Frame
        tab_search("Display database in treeview").tkraise()

        inst = super(Specific_exercise_class, self)
        inst.Search(selection)
```

Very clearly the specific exercise class is inheriting from the treeview class. It inherits all the parameters and functions of the treeview class. An instance of that inherited class is created in the To_Treeview method - {refer above for full code context}

Polymorphism

```
class Specific_exercise_class(display_in_treeview):
  def __init__(self,**kwargs):
    super().__init__(**kwargs)
```

Since the specific exercise class accepts any number of arguments through the" **kwargs" parameter this is an example of compile time polymorphism because there can be a different number of arguments to treeview each time an instance is created.

Composition

```
class Calendar_Display():
  def __init__(self,d):
    self.calendar  = None
```

```
  def display_Calendar_widget(self):
    self.calendar = Calendar(
      tab_search("Calendar"),
      selectmode="day",
      month=int(self.month),
      day=int(self.day),
      year=int(self.year),
      selectbackground = "red",
      selectforeground = "white"
      )
    self.calendar.place(x=0,y=0,width=W+300)
    self.calendar.bind("<<CalendarSelected>>", self.handle_date_click)
    self.add_event()
  def delete_event(self):
    date = self.date
    if date is not None:
      events_on_date = self.calendar.calevent_get(date)

      for event in events_on_date:
        self.calendar.calevent_delete(event)
```

```
def handle_date_click(self, event):
  reps = []
  rep_weights = []
  pr_weights = []

  for i in tab_search("Calendar").winfo_children():
    if i in (self.e,self.e2):
      i.delete(0, 'end')
    elif i == self.calendar:
      pass
    else:
      i.destroy()
  def summary_part1():
    val = int(self.e.get())-1
    specific_exercise_instance = Specific_exercise_class()
    specific_exercise_instance.show(self.exercises[val],"Calendar",None,200,self.x-600,400)

    self.display_Calendar_widget()
  def showing_on_calendar_tab():
    choice = int(self.e2.get())-1
    select = c.execute(f'''SELECT exercises FROM Summary WHERE date = '{self.formatted_date}' ''').fetchall()
    self.exercises = ast.literal_eval(select[choice][0])

    self.l = tk.Label(tab_search("Calendar"),text = "Pick the exercise you want to view: 1 - 6: ")

    self.e = tk.Entry(tab_search("Calendar"))
    self.b = tk.Button(tab_search("Calendar"),text="Click to show")
    self.l.place(x=self.x,y=self.y)
    self.e.place(x=self.x,y=self.y+50)
    self.b.config(command=summary_part1)
    self.b.place(x=self.x,y=self.y+100)

  self.exercises = ast.literal_eval(c.execute(f'''SELECT exercises FROM Summary WHERE date = '{self.date.strftime('%Y-%m-%d')}' ''').fetchall()[0][0])
  from Workout import Specific_exercise_class
  date_string = self.calendar.get_date()
  date_object = datetime.strptime(date_string, "%m/%d/%y").date()
  self.formatted_date = date_object.strftime('%Y-%m-%d')
  select = c.execute(f'''SELECT exercises FROM Summary WHERE date = '{self.formatted_date}' ''').fetchall()
  number_of_entries_on_specific_date = len(select)
  self.exercises = []
  for i in range(number_of_entries_on_specific_date):
    for j in ast.literal_eval(select[i][0]):
      self.exercises.append(j)
  for p in self.exercises:
    reps.append(c.execute(f'''SELECT reps FROM Exercise WHERE name = "{p}"''').fetchone()[0])
```

```
class Workout_display_class():
    def __init__(self):
def add(self):
    current = []
    for i in self.indexes:
      current.append(self.y[i])
    #print(current)
    now = datetime.now()
    #now = now + timedelta(days=6)
    current = json.dumps(current)
    if self.HeadingVar.get() == 1:
      from History import History
      find = c.execute(" SELECT exercises FROM Summary WHERE date = ? ",(now.strftime('%Y-%m-%d'),)).fetchall()
      changed = ast.literal_eval(current)
      for i in changed:
        temporary = c.execute('''SELECT reps,sets,PR_weight,Rep_weight,Enjoyable,How_difficult_it_felt,
        equipment,difficulty,muscle FROM Exercise WHERE name = ?''',(i,)).fetchall()
        temporary = list(temporary[0])
        temporary.insert(0,i)
        temporary.insert(1,now.strftime('%Y-%m-%d'))
        temporary = tuple(temporary)
        c.execute('''INSERT INTO History (exercise,date,reps,sets,PR_weight,Rep_weight,Enjoyable,
        How_difficult_it_felt,equipment,difficulty,muscle) VALUES (?,?,?,?,?,?,?,?,?,?,?)''',temporary)
        connection.commit()


      if current != find:
        query = '''
        INSERT INTO Summary (exercises, date) VALUES (?, ?)
        '''
        values = (current, now.strftime('%Y-%m-%d'))
        c.execute(query, values)
        connection.commit()
      C = Calendar_Display(now)
    else:
      c.execute(f'''
      DELETE FROM Summary
      WHERE date = ?
      ''',(now.strftime('%Y-%m-%d'),))
      c.execute('''DELETE FROM History WHERE date = ?''',(now.strftime('%Y-%m-%d'),))
      C = Calendar_Display(now)
      C.delete_event()
          C = Calendar_Display(now)
          C.delete_event()
```

All these screenshots show composition as the workout class calls events of the calendar class within itself

```
        self.other_class = Specific_exercise_class()
        self.l = tk.Radiobutton(self.Sub_frame, text=line[1]+' Day',command=lambda value2=line[1]: self.on_radio_button_change(value2),
        variable=self.var, value=line[1])
        self.l.pack()
def on_radio_button_change(self,*args):
  selected_value = self.var.get()
  self.other_class.select(selected_value)
  self.other_class.Specific_workout(selected_value)
```

This also shows composition as the workout display class specifies an instance of the specific exercise class and then calls two methods of that class when the radiobutton is pressed, giving it it's dynamicity.

These are composition because the instances of classes called don't make sense outside the class and if they were to be destroyed the other instances of the classes would be affected

TESTING

Mistakes encountered while coding

```
AttributeError: 'Workout_display_class' object has no attribute 'other_class'
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Program Files\Python311\Lib\tkinter\__init__.py", line 1948, in __cal
l__
    return self.func(*args)
           ^^^^^^^^^^^^^^^^
  File "C:\Users\khait\OneDrive\Desktop\Project code backup.py", line 594, in <l
ambda>
    self.l = tk.Radiobutton(self.Sub_frame, text=line[1]+' Day',image=self.image
s[line[1]],compound=tk.TOP,command=lambda value2=line[1]: self.on_radio_button_c
hange(value2), variable=self.var, value=line[1])
                                                          ^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^
  File "C:\Users\khait\OneDrive\Desktop\Project code backup.py", line 601, in on
_radio_button_change
    self.other_class.select(selected_value)
    ^^^^^^^^^^^^^^^^
AttributeError: 'Workout_display_class' object has no attribute 'other_class'
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Program Files\Python311\Lib\tkinter\__init__.py", line 1948, in __cal
l__
    return self.func(*args)
           ^^^^^^^^^^^^^^^^
  File "C:\Users\khait\OneDrive\Desktop\Project code backup.py", line 594, in <l
ambda>
    self.l = tk.Radiobutton(self.Sub_frame, text=line[1]+' Day',image=self.image
s[line[1]],compound=tk.TOP,command=lambda value2=line[1]: self.on_radio_button_c
hange(value2), variable=self.var, value=line[1])
                                                          ^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^
  File "C:\Users\khait\OneDrive\Desktop\Project code backup.py", line 601, in on
_radio_button_change
    self.other_class.select(selected_value)
    ^^^^^^^^^^^^^^^^
AttributeError: 'Workout_display_class' object has no attribute 'other_class'
```

Since the Specific_exercise class inherited from the Workout_display class, self.other_class stores the instantiation of the other class through the Workout_display class.

If "on_button_click" doesn't implement the calling of the other class properly then this error was thrown.

This error frequently came up as I modified my Workout file in several ways affecting the "on_button_click" method

```
= RESTART: C:\Users\khait\OneDrive\Desktop\Project code backup.py
[(1, '1', 1, '1', None)]
[[]]
Enter 1 to sign up and 2 to login: 1
Sign up:
Enter your name: Armaan
Enter your age: 17
Enter your gender (male, female or prefer not to say): male
Create a password:13.5
Traceback (most recent call last):
  File "C:\Users\khait\OneDrive\Desktop\Project code backup.py", line 97, in <mo
dule>
    login(choice)
  File "C:\Users\khait\OneDrive\Desktop\Project code backup.py", line 92, in log
in
    c.execute(f'''INSERT INTO Profile (name,age,gender)VALUES({name},{age},{gend
er})''')
sqlite3.OperationalError: no such column: Armaan
>
```

This is a formatting error.

Since in this instance I didn't use SQL parameters and was using string formatting instead I forgot to add quotes around the value

Therefore, the code assumed it was a column name

```
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\khait\OneDrive\Desktop\Project code backup.py
Enter 1 to sign up and 2 to login: 2
Login:

Enter your name: Armaan
Enter your password: 11
Login successful
>>> bread and butter
[]

======== RESTART: C:\Users\khait\OneDrive\Desktop\Project code backup.py =======
Enter 1 to sign up and 2 to login: 2
Login:

Enter your name: Armaan
Enter your password: 11
Login successful
>>> cake
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Program Files\Python311\Lib\tkinter\__init__.py", line 1948, in __call__
    return self.func(*args)
           ^^^^^^^^^^^^^^^^
  File "C:\Users\khait\OneDrive\Desktop\Project code backup.py", line 1662, in combined
    self.Chart()
  File "C:\Users\khait\OneDrive\Desktop\Project code backup.py", line 1625, in Chart
    canvas.create_arc((2, 2, 150, 150), fill=colours[colour_index], outline=colours[colour_index], start=start,extent=100)
                                             ^^^^^^^^^^^^
NameError: name 'colour_index' is not defined
```

I tried to iteratively loop through the colours array in the constants file but forgot to initialise the colour_index variable outside the for loop

The loop was dynamically iterating through something else which is why I needed to define a separate value to iterate through the colour_index simultaneously

```
0
1
2
3
4
5
6
7
8
9
10
[393.6, 100.0, 17.6, 2.9, 3.0, 271, 142, 74, 57.2, 0.3, 42.1]
pasta
[393.6, 100.0, 17.6, 2.9, 3.0, 271, 142, 74, 57.2, 0.3, 42.1, 156.0, 100.0, 0.9,
 0.2, 5.7, 1, 58, 0, 31.3, 1.8, 0.6]
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
[393.6, 100.0, 17.6, 2.9, 3.0, 271, 142, 74, 57.2, 0.3, 42.1, 156.0, 100.0, 0.9,
 0.2, 5.7, 1, 58, 0, 31.3, 1.8, 0.6]
```

This was a series of print commands I used to check the database wasn't saving the same data repeatedly in the table

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Program Files\Python311\Lib\tkinter\__init__.py", line 1948, in __call__
    return self.func(*args)
           ^^^^^^^^^^^^^^^^
  File "C:\Users\khait\OneDrive\Desktop\Project code backup.py", line 1671, in combined
    self.Chart()
  File "C:\Users\khait\OneDrive\Desktop\Project code backup.py", line 1655, in Chart
    print(array)
          ^^^^^
NameError: name 'array' is not defined

========== RESTART: C:\Users\khait\OneDrive\Desktop\Project coc
Enter 1 to sign up and 2 to login: 2
Login:

Enter your name: Armaan
Enter your password: 111
Login unsuccesssful
Login:

Enter your name: Armaan
Enter your password: 11
Login successful
```

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\khait\Downloads\Python312\Lib\tkinter\__init__.py", line 1962, in __call__
    return self.func(*args)
           ^^^^^^^^^^^^^^^^
  File "C:\Users\khait\Desktop\Project\Cal.py", line 198, in combined
    self.Chart()
  File "C:\Users\khait\Desktop\Project\Cal.py", line 155, in Chart
    if text_size > 10:
       ^^^^^^^^^
UnboundLocalError: cannot access local variable 'text_size' where it is not associated with a value

========== RESTART: C:\Users\khait\Desktop\Project\Main.py
Enter 1 to sign up and 2 to login: 2
Login:

Enter your name: Armaan
Enter your password: 11
Login successful
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\khait\Downloads\Python312\Lib\tkinter\__init__.py", line 1962, in __call__
    return self.func(*args)
           ^^^^^^^^^^^^^^^^
  File "C:\Users\khait\Desktop\Project\Cal.py", line 197, in combined
    self.Chart()
  File "C:\Users\khait\Desktop\Project\Cal.py", line 157, in Chart
    text_x = center_x + bisector_length * math.cos(math.radians(middle_angle))
                                                   ^^^^^^^^^^^^
NameError: name 'middle_angle' is not defined
```

Moving my code around I erased the calculation for middle_angle accidentally so this error was thrown.

It was an easy mistake to fix

```
Login successful
Traceback (most recent call last):
  File "C:\Users\khait\Desktop\Project\Main.py", line 332, in <module>
    run_application()
  File "C:\Users\khait\Desktop\Project\Main.py", line 330, in run_application
    Call_Functions()
  File "C:\Users\khait\Desktop\Project\Main.py", line 291, in Call_Functions
    print(name)
NameError: name 'name' is not defined. Did you mean: 'Frame'?

============== RESTART: C:\Users\khait\Desktop\Project\Profile.py ==============

================ RESTART: C:\Users\khait\Desktop\Project\Main.py ===============
Enter 1 to sign up and 2 to login: 2
Login:

Enter your name: Armaan
Enter your password: 11
Login successful

============== RESTART: C:\Users\khait\Desktop\Project\Profile.py ==============
Traceback (most recent call last):
  File "C:\Users\khait\Desktop\Project\Profile.py", line 11, in <module>
    p = Profile()
  File "C:\Users\khait\Desktop\Project\Profile.py", line 4, in __init__
    self.show()
  File "C:\Users\khait\Desktop\Project\Profile.py", line 6, in show
    print(c.execute('''SELECT * FROM Profile''').fetchall())
NameError: name 'c' is not defined

================ RESTART: C:\Users\khait\Desktop\Project\Main.py ===============
Enter 1 to sign up and 2 to login: 2
Login:

Enter your name: Armaan
Enter your password: 11
Login successful
```

This is due to my import statements. Since all my definitions for cursor connection to the database was kept in one file for faster running speeds, I forgot to import the file into the Profile file

Again, a quite easy mistake to fix

```
========== RESTART: C:\Users\khait\Desktop
Enter 1 to sign up and 2 to login: 2
Login:

Enter your name: Armaan
Enter your password: 11
Login successful
Traceback (most recent call last):
  File "C:\Users\khait\Desktop\Project\Main.py", line 332, in <module>
    run_application()
  File "C:\Users\khait\Desktop\Project\Main.py", line 330, in run_application
    Call_Functions()
  File "C:\Users\khait\Desktop\Project\Main.py", line 293, in Call_Functions
    P = Profile(name)
TypeError: Profile.__init__() takes 1 positional argument but 2 were given
```

This is due to the syntax of the constructor in classes.

Since my Profile part is made using a class, I forgot to add name as a parameter in the constructor so therefore my input of name while creating an instance of a class threw an error

Quite a simple mistake to fix

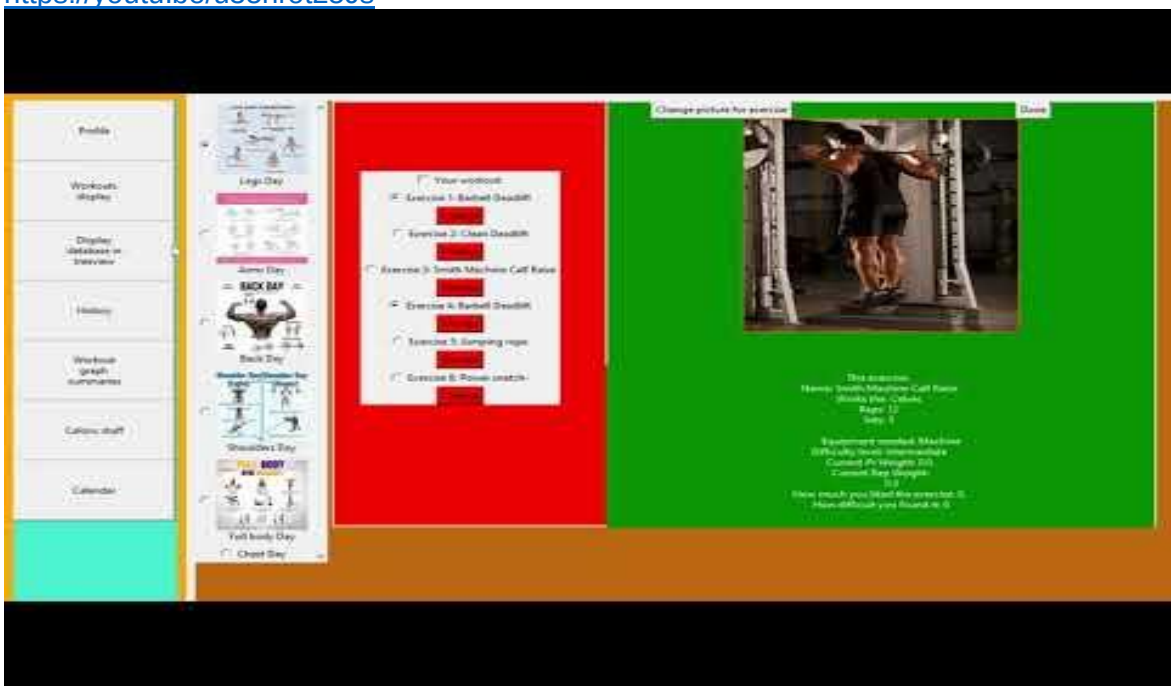
```
========== RESTART: C:\Users\khait\Desktop\Project\Code\Main.py ==========
Login

Enter your name: Armaan
Enter your password: jim
Login successful
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Program Files\Python312\Lib\tkinter\__init__.py", line 1962, in __call__
    return self.func(*args)
           ^^^^^^^^^^^^^^^^
  File "C:\Users\khait\Desktop\Project\Code\Workout.py", line 120, in add
    C = Calendar_Display(now,current)
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\khait\Desktop\Project\Code\Calendar.py", line 8, in __init__
    self.b = tk.Button(tab_search("Calendar"),text="Click to show",command = summary_part1)
                                                                              ^^^^^^^^^^^^
NameError: name 'summary_part1' is not defined

========== RESTART: C:\Users\khait\Desktop\Project\Code\Main.py ==========
Login

Enter your name: Armaan
Enter your password: jim
Login successful
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Program Files\Python312\Lib\tkinter\__init__.py", line 1962, in __call__
    return self.func(*args)
           ^^^^^^^^^^^^^^^^
  File "C:\Users\khait\Desktop\Project\Code\Calendar.py", line 54, in summary_part1
    val = int(e.get())
              ^
NameError: name 'e' is not defined. Did you mean: 'self.e'?
```

This was because I had changed my entry widget variable into an attribute of the Calendar class, so I forgot to replace “e” with “self.e”.

Remarkably simple mistake to fix

Video link to testing of application

https://youtu.be/u35nrot25Js



Evaluation

- *I think I still have a few objectives left to achieve before the final deadline:*
- *History of workouts table implemented in graph*
- *Eliminate the need of internet connection in the calorie information retrieval*
- *Create the probability logic for when difficulty is entered*
- *Create weight brackets based on reps*
- *Create separate table in database to account for varying reps in sets and drop, super and regular set*

- *I chose to make my NEA a computer app because it was what I was most familiar with and already had knowledge at my disposal that I could work on. In retrospect a mobile app may have been more practical, but a computer app is still useful for recording and tracking diet and workout.*