OLYMPICK

By

Ellie, Faiqa, Megan & Melanie

Introduction

Hello and welcome to our group project named Olympic.

We as a group are delighted to have this opportunity to show our existing and new skills that we have picked up during this entire Code First Girl's course and value every effort that our instructors have made and especially their patience to answer our questions, which at times probably were tedious to answer repetitively.

We would like to introduce ourselves and give a summary about how we got to the point of deciding our project **Olympick**. I believe the name is self-explanatory however in the next page you will find our aims and objectives which will outline the theme.

Our group consist of four members named Ellie, Faiqa, Megan and Melanie. We discovered our appetite for two areas: plants and sports. We started with the idea of a plant-based app which would give us the name and information of the plant deriving from an API somewhat like Wikipedia and use that information to integrate with new methods to use the plant for various recipes and essential oils. However, after a thorough intense research we couldn't find a public API for our project and terminated that idea. We then decided to go for another option which, Megan spotted a twitter post regarding a schedule for the Olympics this year. We loved the idea and wanted to solve this problem as we all love watching Olympics and even have great memories with families hence this touched our hearts. We did some research to find public APIs and decided on a name that would reflect the theme of our project. The juicer part of Olympick is in the next section.

Aims & Objectives of Olympick

Our aim of this app is to provide a schedule for sport fans and to create a program that would display a schedule of all the Olympic sports taking place this year. This would include time and place (although this year the Olympics are in Tokyo). Our aim was using a public API to and use SQL database to save favourite sport's schedule.

We have listed the objectives below:

1. Able to log-in
2. Able to view the all the Olympic sports events schedules
3. Able to choose and save a list of favourite events in their personalised schedule
4. Able to view the saved list of favourite events in their personalised schedule
5. Able to remove any events from the favourite list

Contents

Background

Details about Olympick

Olympick program is based on the major international multi-sport event normally held once every four years. The Olympick program is supposed to give a list of sport events that are taking place this year in Tokyo. We wanted to create a program that would allow the user to view all the sport events that are taking place. After viewing the events, the user can create their own events' schedule to view them with the necessary information, such as name of sport and time.

Here is a tweet that inspired us to create the Olympick program

How Olympick works

The program is based on creating an account to add, remove and update all favourite sport events. The program displays a list of events taking place deriving from the public API. The program verifies the user account and gives options to pursue the program. The user's favourite sports are saved in a database and the user can obtain their list of favourite sport events by viewing them on a personalised API e.g you have to login with username and password.

Rules & Requirements

We identified the user needs first and concluded the basic rules; it must be able to display all the events and the program must be able to create a user schedule that can be manipulated. Based on the two rules we drew out the program requirements (listed below), biggest problems, and tools that we require.

- User can create an account with password, and login with that username/password later
- App can verify if user is existing or new - e.g. has an existing username and correct password or not
- User can search for a particular sport
- User can view the list of all the sport events
- User can have the flexibility to add one or multiple favourite events to their personalised schedule
- User can have the flexibility to remove one or multiple events from their personalised schedule
- The database will update automatically after the user removes their events
- User can view their personalised schedule at any time
- User can view all of the events in UK time
- Multiple users can use the program at one time
- Program has a backup system
- Program should not allow duplicate users
- User can choose their journey - either via the terminal, or via the browser
- If user inputs incorrect values, our program will avoid errors with exception handling

Biggest Problems

- Too many users can slow down the server
- One user can create more than one account
- If user forgets password the solution has not been considered
- The program needs to include e-mail to reset password
- Program backup if the system crashes
- Time constraints to complete the project

Tools

- Python
- SQL database
- Flask for creating our API endpoints
- MySQL Connector to connect to our database with Python
- 2020 Olympics Public API: (https://olypi.com/)
- HTML5 to display schedules on the frontend
- Tests: Unit tests with mock and patch

We came up with some key features and extended goals (if time allowed) for the front-end design.

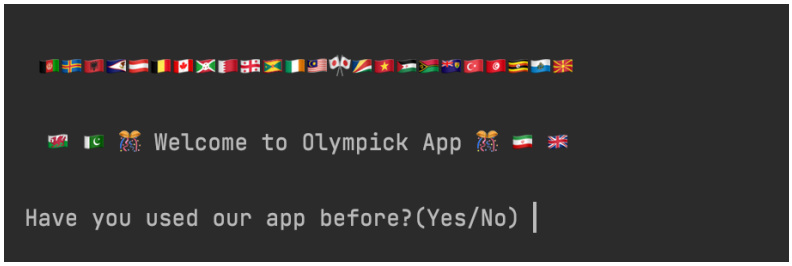
- Front-end for the user to login and input the sport name
- Filter the favourite events for the user
- *Extended goal:* Log-in and authentication
- *Extended goal:* An email reminder some time before an event
- *Extended goal:* Able to sync events schedule into Google calendar
- *Extended goal:* Design the front-end like a calendar
- *Extended goal:* Automated program execution

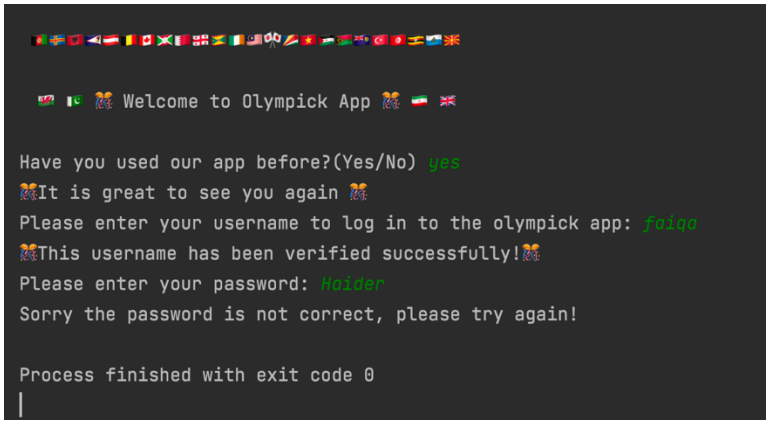
Specifications & Design

Technical requirements

The program uses Python and SQL languages. User credentials and data will be saved on to the SQL database to create a schedule. The user can add and remove any event from the list. The program automatically updates the database when an event is added or removed.

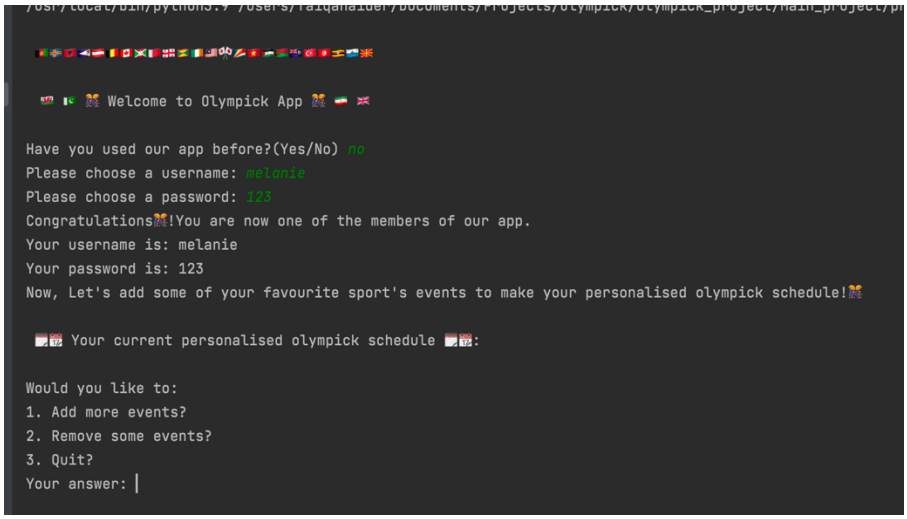
The group made the decision in the planning process to keep the UI fairly simple. It requires a user to interact with the terminal to make their choices. Upon running the program, a user is prompted to either log in or sign up. User data is authenticated by retrieving username and password data from the database and comparing it to the submitted details. Here, we rely heavily on ***conditional logic*** and have tried to include ***error handling***, where required. For example, you will receive an error message if you submit the wrong password or if you try to sign up with a username that already exists.





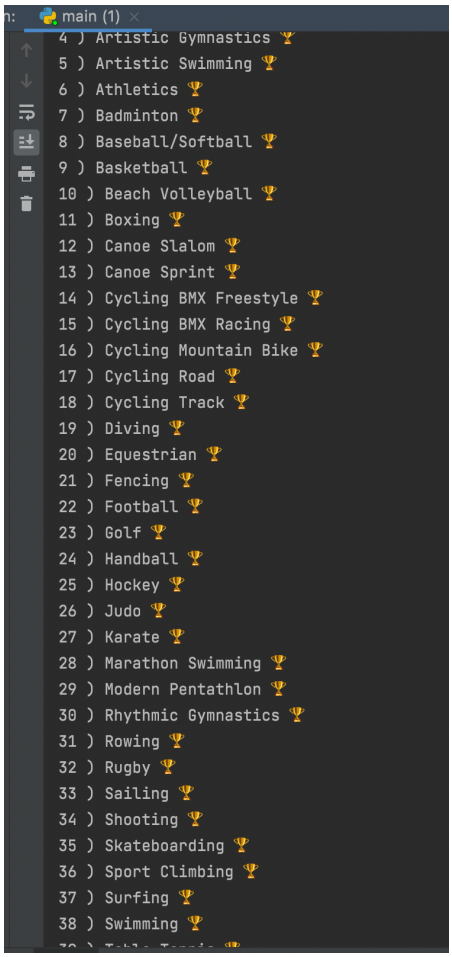
Password is case sensitive, returns incorrect password

Upon successful login, a user with a saved schedule will be provided with that schedule and three options to traverse the program. This requires a call to the database to retrieve the data stored under that username. A new user will only be provided with the three options. The provided options are you can add events to your schedule, delete events from your schedule, or exit the program.



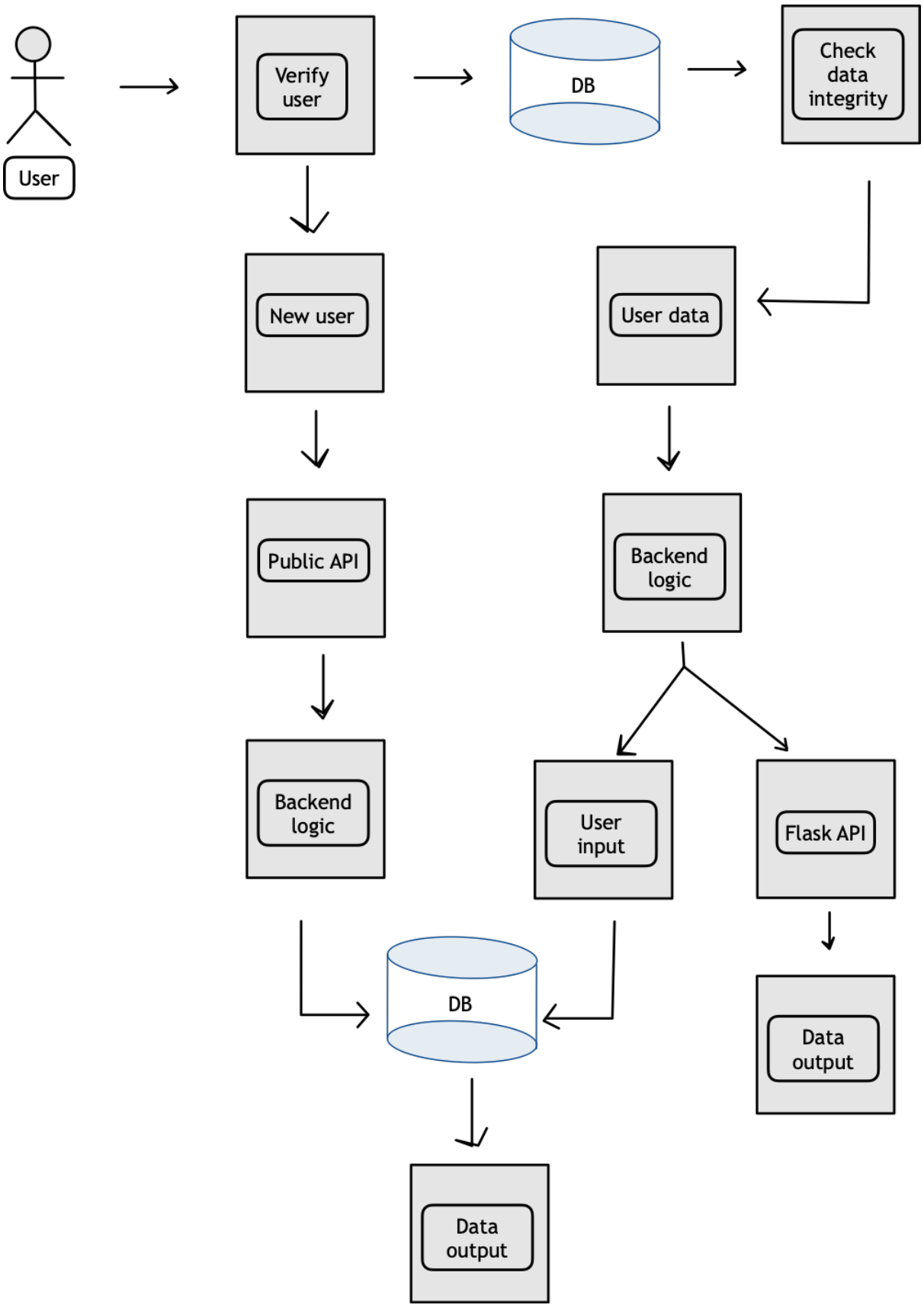
Choosing to add more events to your schedule requires two separate calls to the public API. The first call retrieves data of all the sports held in the API and the program returns them to the user. The user then selects which sport they wish to see a list of events for. A function checks whether the sport name is in the data, and if it is, returns the API's ID for that sport. Then another call to the API is made. This API call is wrapped in a decorator which takes the sport ID and returns a list of events of that sport, helpfully indexed for the user to add to their schedule. Once the user picks events to add to their schedule, these events are written to the database and stored under the user's name. The user is then provided the list of options again.

Removing events from a user's schedule works in a similar way to adding them. A user is prompted for the numbers of the events they wish to remove. This string is then split into a list, mapped over where values are converted into integers and indexed, and finally appended to a new list. This new list is fed into another function, which iterates over event names and adds them to a list to remove. The MySQL Connector cursor then executes a delete query on those events and the changes are committed. The user is then redirected back to the options. Once the schedule has been modified to the user's liking, they are then able to quit the program and are free to return whenever they want to view it again.



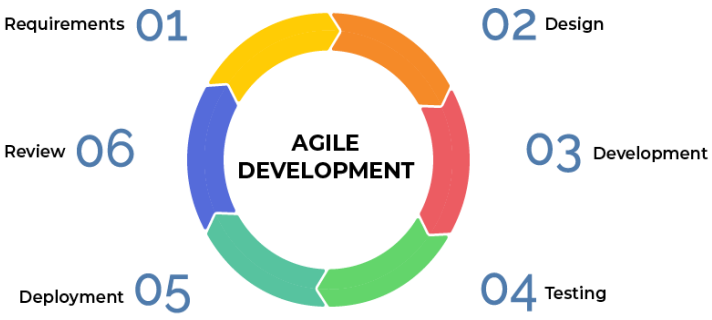
List of sport events taking place

Design & Architecture

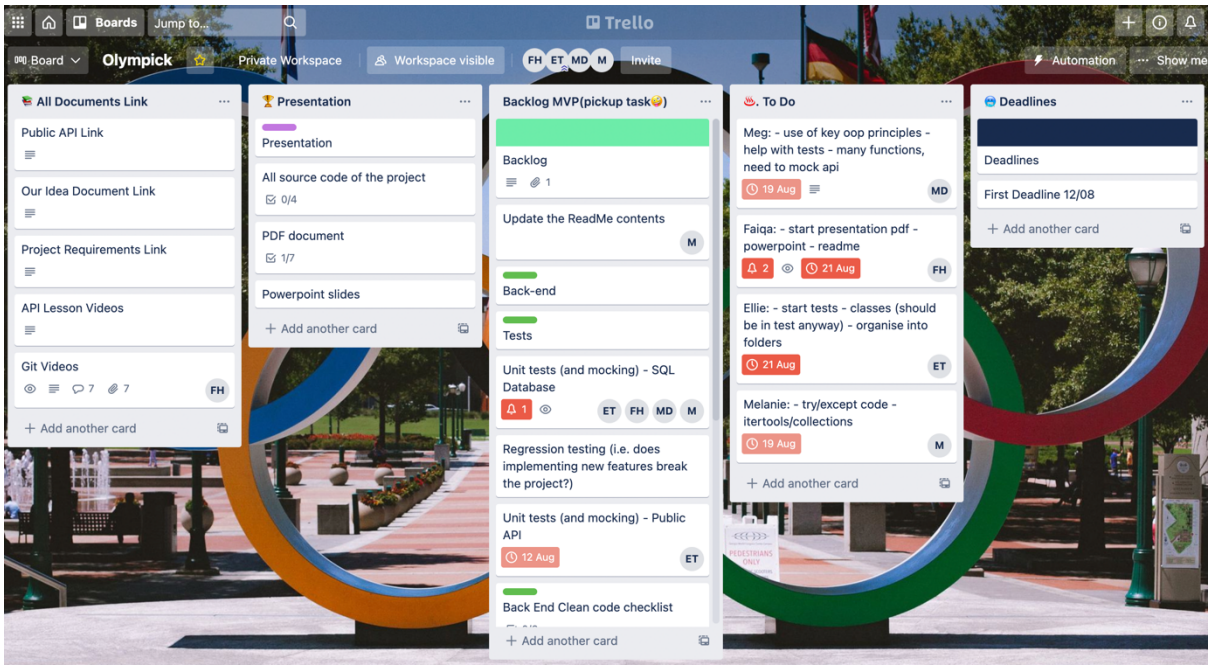


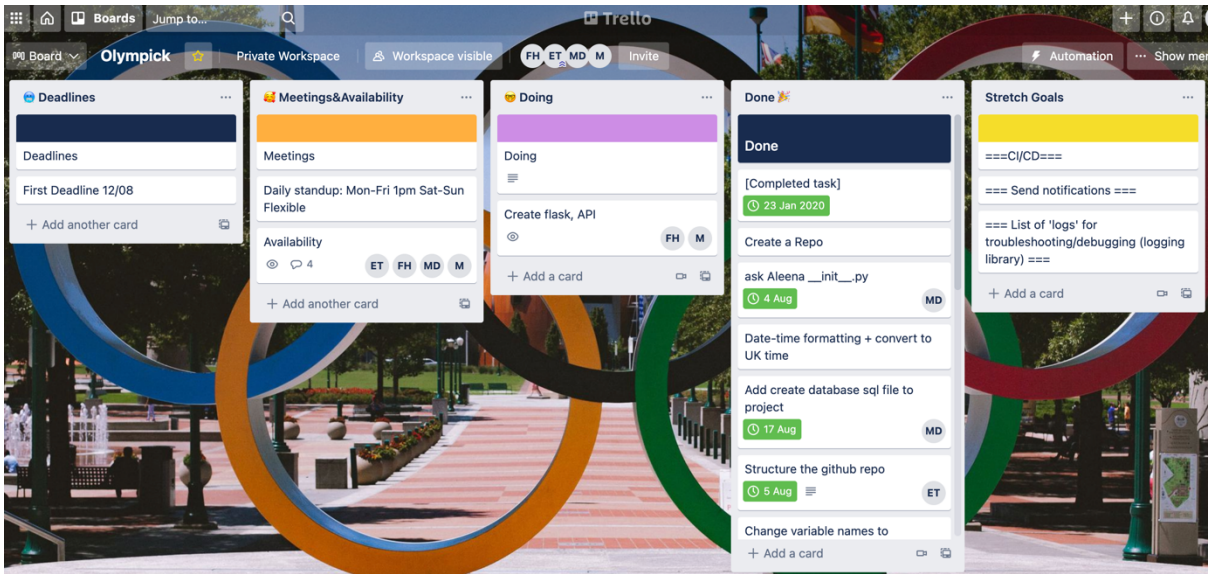
Implementation & Execution

We used Agile to implement an iterative approach. This was particularly useful because of time constraints and the ability to increment and implement the code as the program developed. Team members tested the running of the program and debugging along the way.



We used Trello Board to create tasks and sub sections of tasks to carry out. Project members were allocated tasks according to skill sets including small challenges to stretch our skill sets.





Code reviews were carried out on Github by all members of the team. OOP principles were reviewed and refactored for better practice. Ellie created some videos for the team, to explain how Github is going to be used and maintained.

Team Member Roles

The team members worked together implementing the code and debugging errors.

Teamwork:

- Research different ideas and discussed them
- Transmitting each other's knowledge from different areas e.g github, flask
- Found an Olympic public API
- Working with the Agile approach
- Allocating tasks
- Daily stand-up meetings for updates
- Review each other's pull request and codes
- Debugging each other's codes
- Encouraging each other along the journey

Meg:

- Programmed Public API code
- Programmed Flask code
- Programmed SQL/database connection code (including username/password verification)
- Programmed the 'main' file for these functionalities to run together on the terminal and the 'app' file for these functionalities to run together on the browser
- Debugged errors
- Code reviews
- Annotated code with explanations
- Hashed passwords
- Trello board management

Faiqa:

- Documentation
- Researched and try to code Flask
- Code reviews
- Error handling
- Debugged errors

Melanie:

- Code reviews
- Flask
- Readme
- Debugged error
- Error handling

Ellie:

- Handling Github repository
- Program public API with the decorator functions
- Debugged errors

- Error Handling
- Fixing branch conflicts
- Code reviews
- Trello board management
- Testing: unittesting
- User journey via terminal

Tools & Libraries used

- HTML files
- SQL Database
- Python
- Public Olympic API
- Git for version control
- GitHub for code review and for storing code
- Flask/Requests API for schedule view
- Mysql.connector
- Cache module from Flask
- Datetime library
- bcrypt for hashing
- Unittest mock

Achievements:

- User has the flexibility to access the program from the terminal or the browser
- Revised and implemented code that was most efficient for the program
- Manage to get the program to run
- Manage to include some of our extended goals i.e connecting database to API
- Manage to use Decorators and Generators to make the code easy to read/understand
- We added recursive functions
- *Extended goal:* Log-in and authentication
- Working front-end (HTML)
- Adhered to PEP8 guidelines (time permitting)
- Password hashing so that user security was guaranteed
- Unittesting

Challenges:

- Handling the Github repository
- Limited time to complete our extended goals for the project
- Conflicting schedules for group collaboration
- Faced challenges in the planning stage of the project which delayed in implementing the code
- Programming the database to execute multiple queries at a time - e.g. to remove or add multiple events to the schedule - was a challenge.
- We couldn't find an API that would fit within the project goal, and we adapted according to what was available.

- Writing down different tests for various functions

Changes along the way:

- Initially we started only with getting the data from the public API and then extended our idea to use SQL database with the public API to help the user to create their own personalised schedule
- Refactored some of the functions such as decorators, generators and recursion
- Created a SQL database and connect it with Flask
- Hash password for user data protection
- Cache module from Flask was used for memory efficiency and single responsibility principle - e.g. so that functions could be separate and serve different purposes whilst using essential variables from other functions
- Clean the code so it would comply with the best practice of PEP8.
- There is a desire to keep the code DRY but this was a challenge.
- We added extra features such as database and front end.

Testing and Evaluation

What was the testing strategy?

We pursued an analytical testing strategy - as a team, we defined our testing conditions to be covered after analyzing the test basis - we used our requirements for this.

As we tested based on our requirements - in a sense, our program's functionalities - our requirements were analyzed to derive the test conditions. Then our tests are designed, implemented and executed to meet those requirements.

We recorded our results in respect to our requirements, and using an Agile methodology, we refactored the code depending on whether a requirement passed or not.

We used unit-testing and mocking to achieve this result.

User testing

We wanted to ask one of the CFG students to test the program. However, we didn't get the time to do this. We did, however, test as a team and though our requirements all passed, we acknowledge there is room for improvement.

System limitation

We wanted to create a time decorator to test how long it takes for functions to return data. But we were unable to carry this out because of time complexity. System testing, we didn't get time to test the capacity of the program i.e., after how many users it would crash however we think this can depend on the computer capacity.

```python
from unittest import TestCase, main
from unittest.mock import patch
from use_functions_for_tests import list_of_all_sports

# Creating a mock data for checking the data we get from the olympic public API
class Test_list_of_all_sports_from_public_api(TestCase):
    @patch('use_functions_for_tests.list_of_all_sports')
    def test_list_of_all_sport_for_first_item(self,mock_data):
        mock_data.return_value.json.return_value = {
            "result": [
                {
                    "id": 1,
                    "name": "Opening and Closing Ceremonies",
                    "link": "/tokyo-2020/en/schedule/ceremony-schedule#20210723_CER"
                }
            ]
        }
        result_function = list_of_all_sports()
        # the list_of_all_sports function use exceptions , so we needed to check them here
        if result_function:
            self.assertEqual(result_function.json()['result'][0]['name'],"Opening and Closing Ceremonies")
        else:
            self.assertRaises(Exception, list_of_all_sports, "Something went wrong, we are not able to retrieve all sports from the pu
```

Conclusion

Faiqa

I felt this project has given me insight on how to work on a coding project, this was my very first software development project that I really enjoyed. During the early stages of the project, I felt it was difficult for me to know 'how to get started'. Working with skilled team members I felt I can learn a lot, as we develop the project. I contributed towards researching the kind of project we wanted to do and reading the code to get a better understanding of the implementation stage and user interaction testing. I developed a better understanding of what is expected in such a project. Overall, I felt that the implementation was challenging but not too difficult as there are many resources available, however the earlier stages of the development was very important, and I feel we haven't put in as much time towards the early stages. The most interesting part was being able to work together as a team and learn from each other.

The biggest problem we faced was finding the time to work together and to be able to complete the project to the best possible solution. This was a valuable experience and certainly one of the very first ones, which I will remember forever.

Melanie

I gained a lot from being involved in this project. I feel fortunate to have worked with the members of this team, who are all smart and capable and more than willing to help each other when the need arises. There have been obstacles to overcome and several constraints on our time, but I think we are all proud of what we have achieved.

Personally, I learned about all the steps needed to create an application in Python and SQL, languages I had little to no experience in, how to debug errors and handle errors, how important the planning stage is in the development of a project, and potential areas for my own growth and improvement.

Megan

I truly enjoyed working on this project, and in this team. I personally learned a huge amount from this project, and can now say that I can confidently program in areas that I wasn't sure I'd be able to - Flask, decorators and hashing have definitely all become more understandable concepts through practical programming. Time constraints definitely posed a problem - I know that I was disappointed that we didn't have the time to incorporate email notifications, regression testing and password resetting via email! I'm looking forward

to becoming more adept at Agile methodologies and TDD as I continue to code, which is something I felt my lack of experience of.
I feel that I learned a lot about programming concepts and teamwork, and I am really grateful that I had this experience with such fantastic team-mates.

Ellie

I really enjoyed working with my lovely team-mates, learning from them and teaching them all that I know - I showed my team-members how to use Git and GitHub and recorded videos to help them understand in their own time. As a front-end developer, I often use Agile methodologies and encouraged my team to work in Agile ways, showing them how to use Trello boards and have daily meetings. I learned how to use decorators and generators in daily programming, as well as recursive functions. I have a better understanding of unit testing now then I did before, and am excited to implement this in my own coding. Reviewing everyone's code was a great learning experience for me and helping people resolve conflicts in their branches was rewarding. This was my first Python project and also my first time in a team of all women, which was an environment that I have always wanted.