COMP 3512
Assignment #2: Single-Page App

Version: 1.0 Nov 3, 2024
Version: 2.0 Nov 22, 2024; changes in yellow
Due Monday Dec 9, 2024 at midnightish

Overview

This assignment provides an opportunity for you to demonstrate your ability to work with JavaScript. The application you will be creating is a Single-Page Application to view F1 (Formula 1) race data. You can work in pairs or by yourself. Don't delay finding a partner; if you cannot find a partner to work with, **don't ask me to be in a group of 3**, you will have to work alone. The assignment is very doable by one person, but it will be a more pleasant experience sharing the work with another.

Beginning

It is your responsibility to read all the assignment instructions thoroughly and carefully. If you are unclear about something, ask me. But before you do, read the material in question again!

Starting files are on D2L and are also available at:

```
https://github.com/mru-comp3512-archive/f2024-assign2.git
```

Grading

The grade for this assignment will be broken down as follows:

Visual Design	15%
Programming Design and Documentation	10%
Functionality (follows requirements)	75%

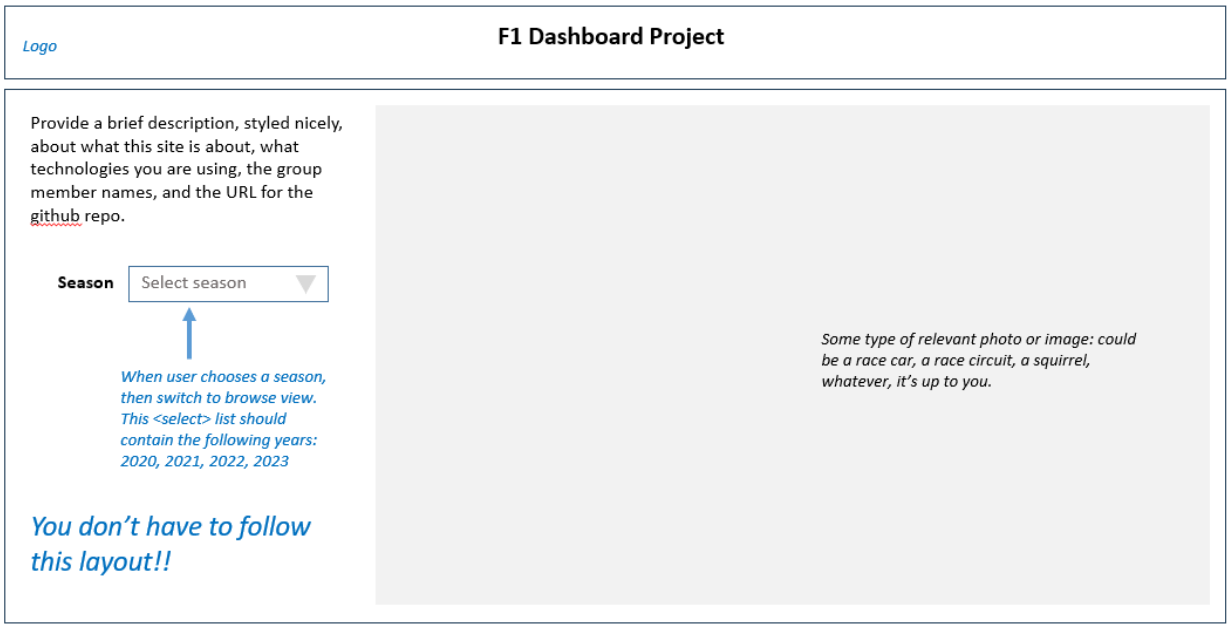
Requirements

This assignment consists of a **single** HTML page (hence single-page application or SPA) with the following functionality:

1. You can use a third-party CSS library/framework (indeed I would encourage you to use one). Tailwind CSS is a great thing to have on your resume, but it does have a bit of a learning curve (Lab7b in your gumroad package can help you quickly learn its essentials). Be careful though that your CSS library is not using/requiring its own JavaScript library as well. Some popular CSS libraries (e.g., Bootstrap) include their own JavaScript libraries to do fancy things with select lists, modals, etc. **You are not allowed to include these** so be sure you don't have any extra <script> tags from your CSS library!
2. Your assignment should have just a single HTML page which **must be** named `index.html`. This is quite different from your assign 1, where you had multiple PHP pages with markup; here you will have just one single HTML page.
3. The assignment should have two main views (**Home** and **Races**) and three dialog/popup views (**Favorites, Driver, Constructor**). When the program first starts, display the **Home** view.

4. The source of data is the provided F1 API. More detail will be provided below. If this was a real-world project, you would use the API that you created in assignment 1. However, that would complicate the hosting quite a bit: you would need a live PHP server and, sadly, in 2024 there are fewer simple free PHP hosting options available. Thus, your assignment will simply consume my F1 api.
5. **Make sure you are only requesting/fetching the season race data ONCE after the user selects the season!** To improve the performance of your assignment (and reduce the number of requests on my server), you must store the race data in `localstorage` after you fetch it from the API (see Exercise 10.11 in Lab 10). Notice that you have to use `JSON.stringify` to create a JSON string version of the array and then saving that in `localstorage`; similarly, after you retrieve it from `localstorage`, you will need to uses `JSON.parse()` to turn it into an array. Your page should thus check if this play data is already saved in local storage: if it is then use local data, otherwise fetch it and store it in local storage. This approach improves initial performance by eliminating this first fetch in future uses of the application. Be sure to test that your application works when local storage is empty before submitting (you can empty local storage in Chrome via the Application tab in DevTools). Please do this task early on so that you reduce the load on my server! **Failure to implement this functionality will result in a very large loss of marks so don't neglect it.**
6. You must write your own JavaScript. That is, no jQuery, no Bootstrap, no other third-party JavaScript libraries. You have all the knowledge you need from the lectures and the three JavaScript labs to complete this assignment. If you do find yourself, however, making use of some small snippet of code you found online (and I do mean small), then you must provide references (i.e., specify the URL where you found it) via comments within your .js file. Failure to properly credit other people's work in your JavaScript will result in a zero grade. Using antediluvian JavaScript found online will also result in lower marks. There is no need to credit code you found in the labs or lectures.
7. **Terminology**: In Formula 1, *races* happen on a *circuit* (e.g., the British Grand Prix is one the Silverstone circuit). There are 10 teams (called *constructors*) in F1; each constructor has two *drivers*. Before the race itself, there is *qualifying*, which determines the starting position. There are three rounds of qualifying, with 5 drivers knocked out in each of the first two rounds. A given race has a set number of laps; the first driver to cross the finish line after completing the set number of laps is the winner. In the race itself, there are points awarded to the driver *and* to the constructor for finishing in the first 10 places. Finishing 1st, 2nd, or 3rd gives the most points. There are other points that can be awarded but you don't care about that. At the season's end, there are two trophies: one for the driver with the most points and one for the constructor with the most points. In this assignment, you are only concerned with viewing races, displaying race and qualifying results, displaying information about circuits, drivers, and constructors, and specifying one's favorite circuits, drivers, and constructors.

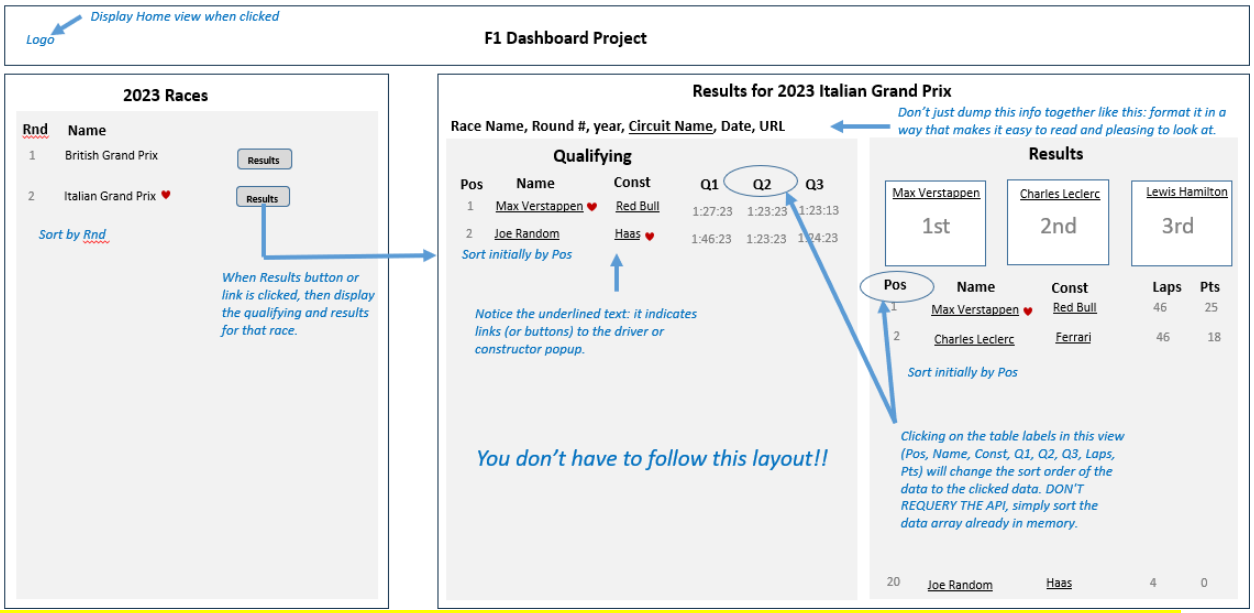
8. **Home View**. You will implement a page with similar functionality as that described below. The provided sketch below illustrates the functionality. You are welcome to modify the design as you see fit. When the user selects a season, then switch to **Races View**. The title for this view doesn't have to be "F1 Dashboard Project"; it, and the other bold labels in these views, are there to explain what functionality must be on the view, not necessarily the actual labels that must appear on your page.



9. **Races View**. You will display the races (a *race* is on a *circuit*) for the selected season/year sorted by the round. Initially, this view will just display the races (perhaps display a message saying please select a race). When a race is selected, then display the qualifying results and race results. In the race itself, finishing 1st, 2nd, or 3rd gives the most points, so you should display those positions more prominently.

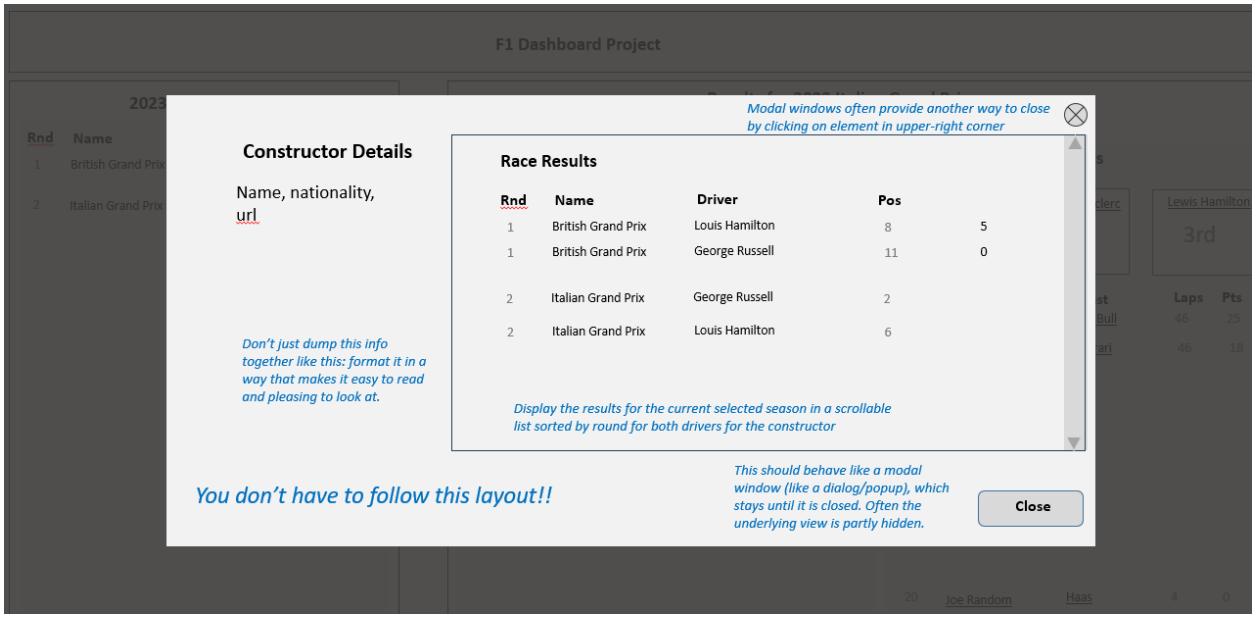
   People who follow F1 typically have their favorite drivers, constructors, and Circuits. In the **Driver, Constructor, and Circuit Popups** (see below), the user can add a driver, constructor, or circuit to their favorites list. In the Races View please indicate somehow if one of these is favorited. In the diagram shown here, I've used a heart icon but you can do it however you'd like.

   Initially display the races sorted by round, and the qualifying and results sorted by position. If the user clicks on the column headings, sort the list on the clicked field. The list should refresh whenever the user changes the sort option. Provide some visual cue that a sort change has occurred, such as an icon or styling changing in the header. The list must be an unordered list.

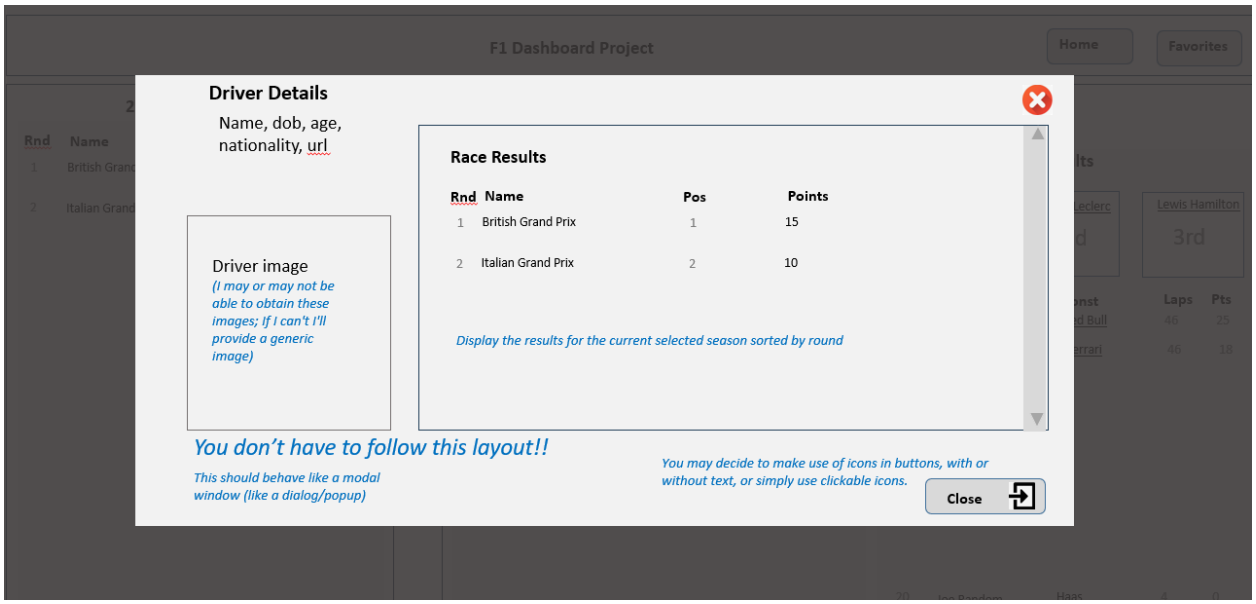


Note: while you don't have to follow this layout, from a usability perspective, it is important that the user can see the qualifying results and race results together without having to scroll. Ignore the hearts shown above.

10. **Constructor Popup.** Display details on the selected constructor in a modal-style popup. This typically involves displaying some type of element (div/aside/section/etc) that was previously hidden. It will stay displayed until the user close it (that is, it will become hidden again). You don't have to worry about letting the user change the sort order of race data in these popups: simply sort on Rnd (round).

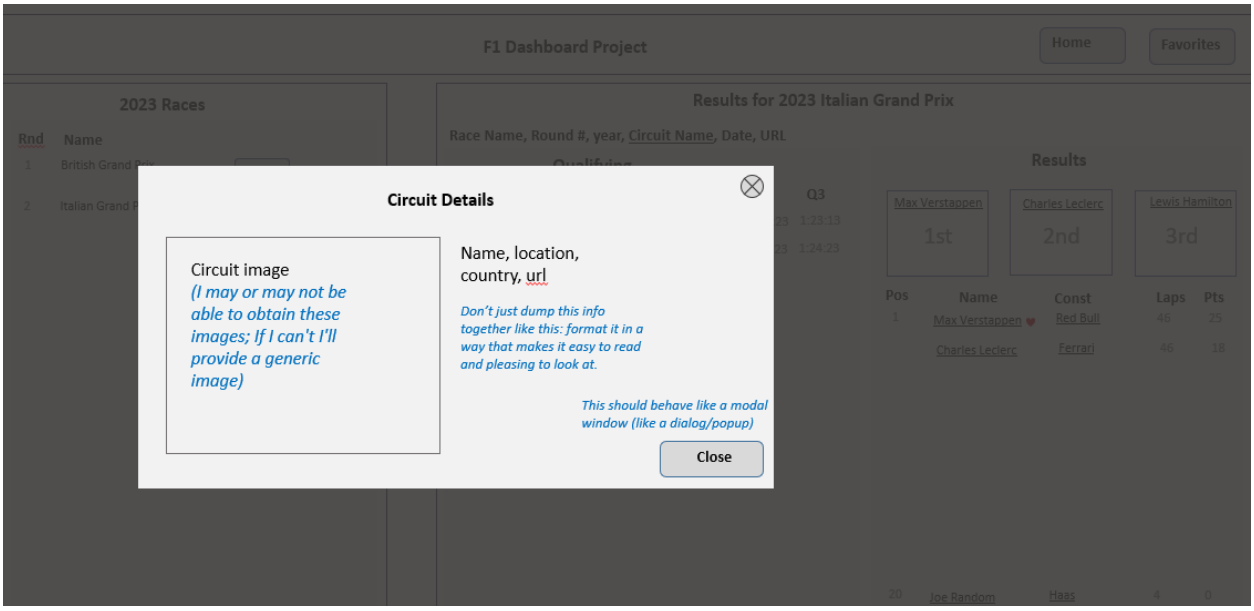


11. **Driver Popup.** Display details for a driver in a modal-style popup. It works in the same way as the **Constructor Popup**. Here I've included icons to show you that that is a possibility.



For the driver and circuit images, use a placeholder service such as https://placehold.co/.

12. **Circuit Popup.** Display details for a circuit in a modal-style popup. It works in the same way as the **Constructor Popup**.



Testing

Every year students lose many easy marks because they didn't read the requirements carefully enough. When your assignment is getting close to done, I would recommend going through each requirement step and carefully evaluate whether your assignment satisfies each specified requirement.

API [this entire page is new]

I will be providing the API calls on my own server. The domain for the api is:

`[domain] = https://www.randyconnolly.com/funwebdev/3rd/api/f1`

Note: my https certificate is timing out in November, so you may need to use http for a few days.

The full URL includes the domain, the file, and often a query string. For instance:

`[URL] = https://www.randyconnolly.com/funwebdev/3rd/api/f1/races.php?season=2023`

Sample URL	Description
`[domain]/circuits.php`	Returns all circuits.
`[domain]/circuits.php?id=1`	Returns single circuit specified by the passed circuitId value.
`[domain]/constructors.php`	Returns all constructors.
`[domain]/constructors.php?id=1`	Returns single constructor specified by the passed constructorId value.
`[domain]/constructors.php?ref=mclaren`	Returns single constructor specified by the passed constructorRef value.
`[domain]/constructorResults.php?` `constructor=mclaren&season=2023`	Returns the race results for a specified constructor (constructorRef value) and season.
`[domain]/drivers.php`	Returns all drivers
`[domain]/drivers.php?id=857`	Returns single driver specified by the passed driverId value.
`[domain]/drivers.php?ref=piastre`	Returns single driver specified by the passed driverRef value.
`[domain]/driverResults.php?` `driver=piastri&season=2023`	Returns the race results for a specified driver (driverRef value) and season.
`[domain]/races.php?season=2023`	**Recommended.** Returns all the races for the specified season. **This data should be stored in localStorage.**
`[domain]/races.php?id=1100`	Returns just the specified race (raceId value).
`[domain]/results.php?race=1100`	Returns all the results for the specified race (raceId value).
`[domain]/results.php?season=2023`	**Recommended.** Returns all the results for all the races in the season. **This data should be stored in localStorage.**
`[domain]/qualifying.php?race=1100`	Returns all the qualifying results for the specified race (raceId value).
`[domain]/qualifying.php?season=2023`	**Recommended.** Returns all the qualifying for all the races in the season. **This data should be stored in localStorage.**

I have also provided csv files and a database diagram that illustrates how the different data tables are related.

Submitting and Hosting

Your assignment source code must reside on GitHub and reside on a working public server (in this case GitHub Pages).

GitHub Pages works in conjunction with git and github. You push your html/css/images to github repo; and then push to the host. The instructions for doings so can be found at:

`https://docs.github.com/en/pages/getting-started-with-github-pages/creating-a-github-pages-site`

It is up to you whether you want your pages to be public (available to the world) or private (available to only those with access to your github repo).

If you are using a private repo, you must add me as a collaborator!

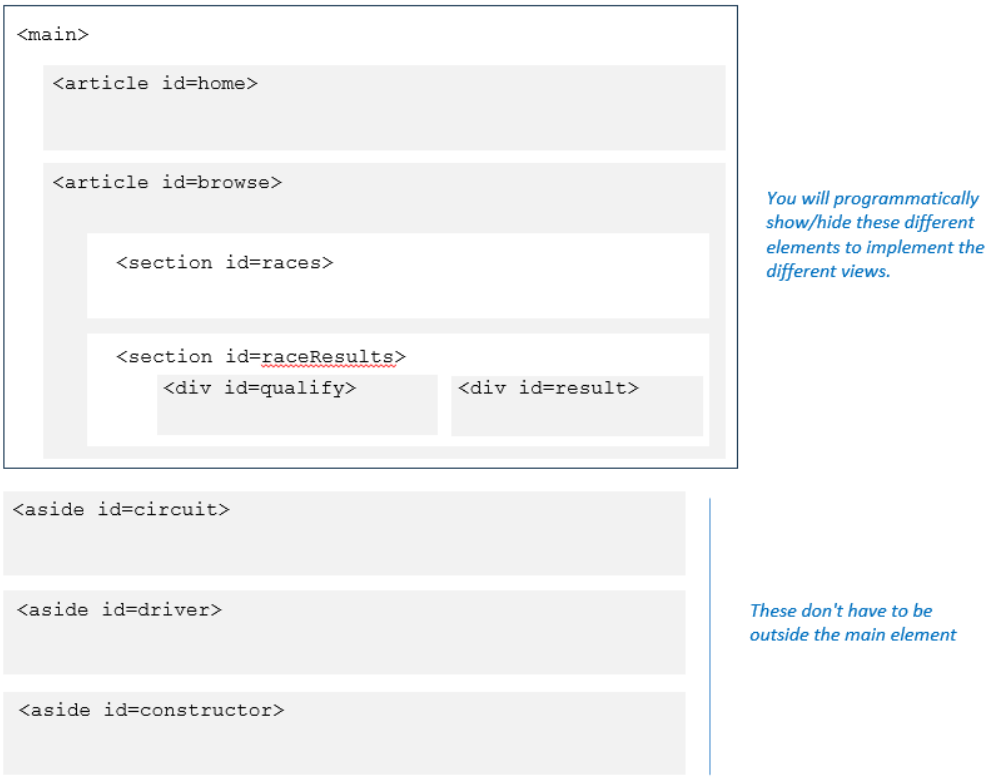
I would strongly recommend getting your hosting to work a few days before the due date. It's okay if your assignment is still not complete at that point: the idea here is to make sure hosting works ahead of time!

When your hosting is working and the assignment is ready to be marked, then send me an email with the following information:

- The URL of the home page of the site on github pages.
- The URL of the github repo so that I can mark the source code. If your repo is private, then add me as a collaborator.

Hints

1. This assignment requires swapping different views. You should implement each view in its own container, e.g.,



Instead of using an <aside> element, I would instead recommend using the <dialog> element in HTML, which now (as of Fall 2024) has new universal capabilities that make it easier to create and style model content.
See https://developer.mozilla.org/en-US/docs/Web/HTML/Element/dialog.

1. Early on, try consuming the API in the browser. Simply copy and paste the URL into the browser address bar. The JSON can be hard to understand because it doesn't have white space, so use an online JSON formatter (such as https://jsonformatter.curiousconcept.com) via copy and paste to see the JSON structure in an easier to understand format. Alternately, install a json viewer extension in Chrome.
2. Recommended program flow *after* user selects season:
    a. Check if data for that season already exist in localStorage. If so, load it into memory and then skip to step g.
    b. Display loading animation
    c. Fetch races for the selected season. Save in localStorage.
    d. Fetch results for the selected season. Save in localStorage.
    e. Fetch qualifying for the selected season. Save in localStorage.
    f. Hide loading animation
    g. Display races and then hide results and qualifying area
3. Recommended program flow *after* user selects race:
    a. Retrieve the relevant race object from the race data in memory (from steps 2a or 2c). Find will be your friend.
    b. Retrieve the relevant results for that race from the results data in memory (from steps 2a or 2d). Filter will be your friend.
    c. Retrieve the relevant qualifying for that race from the qualifying data in memory (from steps 2a or 2e). Filter will be your friend.
    d. Display the race, results, and qualifying data. This will likely require clearing existing data.
4. Recommended program flow *after* user selects a new sort criteria:
    a. Determine what the sort order should be. Bracket notation can be helpful in avoiding multiple if…else structures.
    b. Sort the data
    c. Display the race, results, and qualifying data.
5. Remember that JSON and JavaScript are case sensitive. For instance, it is easy to type in `Name` and it won't work because the JSON field is actually `name`.
6. Your `index.html` file will include the markup for all your views. Your JavaScript code will programmatically hide/unhide (i.e., change the `display` value) the relevant markup container for these views.

7. Most of your visual design mark will be determined by how much effort you took to make the views look well designed. Simply throwing up the data with basic formatting will earn you minimal design marks.
8. Most years, students tend to get low marks on the design side of the assignment. I would recommend looking at other sites on the internet and examine how they present data. Notice the use of contrast (weight, color, etc) and spacing.
9. Most of your programming design mark will be based on my assessment of your code using typical code review criteria. For instance, did you modularize your code using functions? Are your functions too long? Is the code documented? Do you have a lot of code duplication (you shouldn't … if you are copy+pasting code, that should tell you that you are doing things wrong from a design standpoint)? Did you make use of object-oriented techniques? Are variables and functions sensibly named? Is your code inefficient (e.g., fetching the same data repeatedly)? Are you using outdated JavaScript techniques (e.g., inline coding, var, `XmlHttpRequest`, etc)? Is the code excessively reliant on found code from Stack Overflow, ChatGPT, etc?