# CS 464P/564 – Front End Web Technologies
# Homework 3

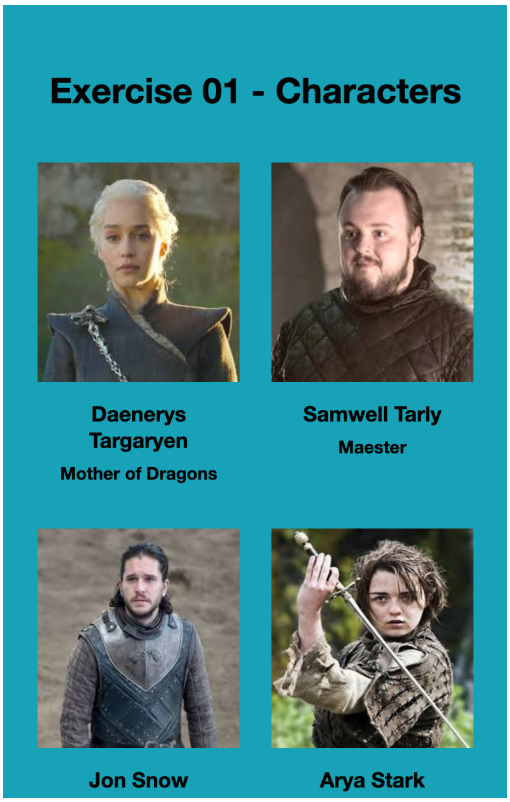
## Instructions

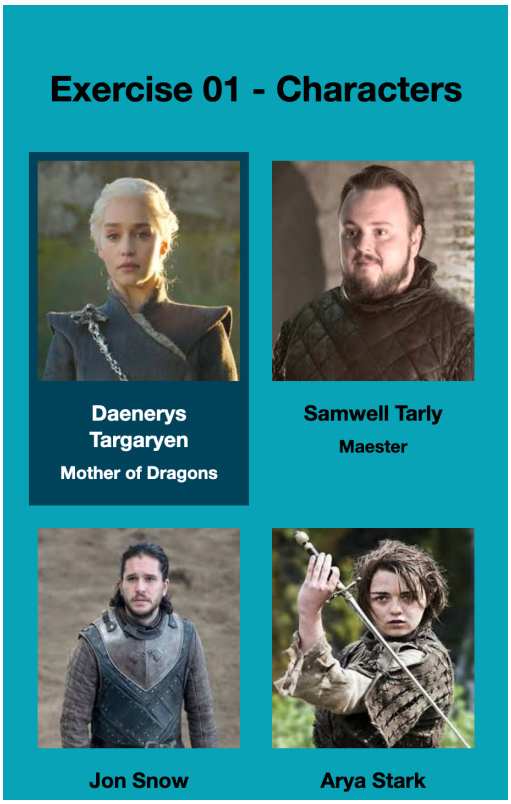
- ❏ In this assignment, you will practice using JavaScript, work with data from an API, charting libraries such as Chart.js, React, and React libraries.
- ❏ As always, make sure to check the solutions in more than one browser.

## Exercises

- ❏ **Exercise 01 - Characters**
  - ❏ Use the files: **01-characters.html, 01-characters.js,** and **styles.css**
  - ❏ In this exercise, you will use the browser Fetch API, the Thrones API, and promises to retrieve all the characters in that API and output the name and picture of each character on the page.
    - ❏ Loop through the response and create the appropriate DOM elements to show an image of every character along with the full name and the title.
    - ❏ Implement a hover behavior for each element, as shown in the screenshots below. (The first screenshot shows the page without any hover and in the second screenshot, the first character element is being hovered over.)
    - ❏ Note: The images in this API have many different sizes. Automatically resizing the images will lead to some distortion, as can be seen below. This is an important issue in front-end development, but it is not one of the primary concerns of this exercise.
  - ❏ The screenshots below were taken at 414px and 1024px in Firefox.

- Resources:
  - Fetch API
  - Thrones API
  - Using Promises | MDN



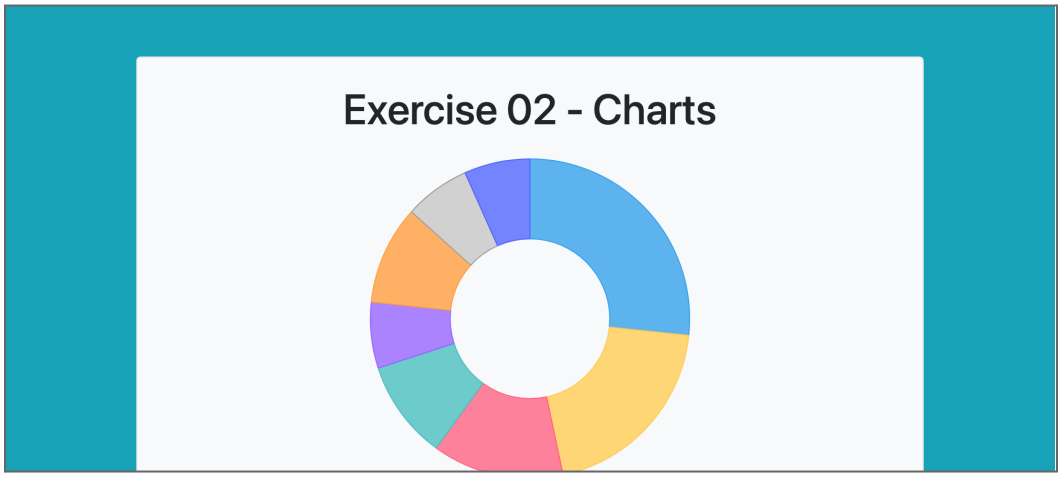




- ❏ **Exercise 02 - Charts**
  - ❏ Use the files: **02-charts.html, 02-charts.js,** and **styles.css**
  - ❏ In this exercise, you will use the browser Fetch API, the Thrones API, and async/await to retrieve all the characters in that API. Based on that data, you will calculate the number of characters in each house. You will then output that data in a donut chart from Chart.js.
    - ❏ You will first need to fetch all the characters and then calculate how many characters are in a given house.
    - ❏ Note: The Thrones API contains imperfect data, just as most APIs do. In this case, the API contains ambiguous organization as well as some incorrect data. You will need to investigate the data and make decisions about how to proceed. For example, one character's last name is listed as Targaryen and another character's last name is listed as Targaryn. This is a typo, as both of those characters belong to the same house. I chose this API knowing the

data is less than ideal, because it's important to check and validate the data before proceeding.

- ❏ There are 53 characters in this API and all that data should be included in the visualization. That being said, you can group data together.
- ❏ Make sure you add the algorithm you use to group the data in the comments of the code.

- ❏ The screenshot below was taken at 1024px in Firefox.
- ❏ Resources:
  - ❏ Fetch API
  - ❏ Thrones API
  - ❏ async function | MDN
  - ❏ Chart.js




- ❏ **Exercise 03-05 - React**
  - ❏ Create your own folder for this exercise.
    - ❏ You will need to create a React project using create-react-app, codesandbox.io, Next.js, or another framework.
  - ❏ In this exercise, you will use the Thrones API and npm modules for routing, making HTTP requests, adding styles, and creating a chart to complete this exercise. You can use the npm modules suggested below or you can choose other modules from the npm registry.

- ❏ Routing:
  - ❏ Ex: react-router-dom
- ❏ Making HTTP Requests:
  - ❏ Ex: axios
- ❏ Adding styles:
  - ❏ Ex: bootstrap
  - ❏ Ex: react-bootstrap
  - ❏ Ex: reactstrap
- ❏ Charting libraries:
  - ❏ Ex: chartjs
  - ❏ Ex: react-chartjs-2

❏ You will create a navbar with these three routes:

- ❏ **/home**
  - ❏ This is the main route that should contain a welcome message.
- ❏ **/search**
  - ❏ In this route, the user can search for a character. Create an input element allowing the user to search for a specific character. Depending on the user input, return the full name and image of that character from the Thrones API.
  - ❏ Consider what happens when that character is not found or when the user input matches more than one character.
- ❏ **/houses**
  - ❏ In this route, your solution should have a donut chart with information on how many characters belong to a given house.
  - ❏ You can reuse the logic used in exercise 02, but the code must be updated to be in React and JSX.

❏ Add styles to this solution using Bootstrap or another CSS framework. There are no specific requirements for this exercise, but all the components in your application should be styled consistently.

- ❏ Selected Resources:
  - ❏ Doughnut | react-chartjs-2
  - ❏ Doughnut Chart | react-chartjs-2

- ❏ **Code Quality**
  - ❏ Check all JavaScript code follows the rules outlined in the Airbnb JavaScript Style Guide and the Airbnb React/JSX Style Guide.
  - ❏ Check the solution is responsive by using the Device Toolbar or Design View in DevTools.
  - ❏ Check the solution is accessible by using the Wave AIM extension.
  - ❏ Check the code is formatted and consistent throughout all exercises.
  - ❏ Make sure to delete unnecessary code from the submission.
  - ❏ Cite any outside sources used to come up with the solution.

## Submission Guidelines

- ❏ **Due date:** Week 6 on Wednesday at 5:00 pm.
- ❏ Submit the URL to the pull request in the **Discussion: Code Review for Homework 3** as part of the initial post on Wednesday.
- ❏ Add all discussion groups members as reviewers.
- ❏ You will need to review all submissions as part of the discussion follow up post on Friday.