IS216 Midterm Assignment [30 marks]

General Instructions:

- This is an open-book, take-home and **individual** test.
- You must test your web pages using **Google Chrome Web Browser** (Version 85.0.4183 or later). Your graders will be using only **Google Chrome Web Browser** (Version 85.0.4183 or later) to test your web pages.
- No questions will be entertained by the IS216 teaching team (faculty/instructor/Teaching Assistants) during the test period. If necessary, make your own assumptions.
- You are allowed to use only standard HTML5, CSS (Version 3), Bootstrap (Version 4.5) and JavaScript (do NOT use jQuery) in your solutions. Do not use any other third party libraries. Do not use JavaScript frameworks (e.g. Vue, Angular, React or similar).
- Use meaningful names for HTML class/id and JavaScript variables and functions. You must indent your code (HTML/CSS/JavaScript) correctly. Use 4 spaces for indentation. Failure to do so will attract a penalty of up to **20%** of your score for the corresponding question.
- You **MUST** include your name as author in the comments of all your submitted source files. Failure to do so will attract a penalty of up to **20%** of your score for the corresponding question.
  For example, if your registered name is "KIM Jong Un" and email ID is kim.jongun.2019, include the following comment at the beginning of each source file you write.

HTML files	CSS, JavaScript files

`<!--`
`Name: KIM Jong Un`
`Email: kim.jongun.2019`
`-->`
	`/*`
	`Name: KIM Jong Un`
	`Email: kim.jongun.2019`
	`*/`

- You may wish to comment out the parts in your code which cause errors. But commented code will not be marked.
- All student submissions will be thoroughly checked by an SMU-approved source code plagiarism checker software program AND an additional external software program. The source code checking will be conducted across all submissions from all 9 sections of IS216. Suspected plagiarism cases will be communicated immediately to the IS216 faculty in charge and SIS Dean's Office for further investigation. Students in the suspected cases will be informed accordingly by their section faculty, and the incident will be escalated to the SMU University Council of Student Conduct. More information about the SMU Student Code of Conduct can be found **HERE**.
- **Resources:** Merged IS216 eLearn → Content ---> Midterm Assignment → midterm.zip
- **Appendices**: Click **HERE**

Submission Instructions

- **Due Date**
  - **14 October 2020 (Wednesday) 11:59 PM Singapore Time**
  - Late submission policy is as follows:

Submit **within 1 hour** of set deadline	10% penalty (of your score for the entire midterm assignment)
Submit **within 3 hour** of set deadline	20% penalty (of your score for the entire midterm assignment)
Submit **within 5 hour** of set deadline	30% penalty (of your score for the entire midterm assignment)
Submit **within 7 hour** of set deadline	50% penalty (of your score for the entire midterm assignment)
Beyond 7 hours, 0 mark (submission will NOT be accepted)	

- **Zip** up all of your files in **q1/q2/q3 folders** into **midterm_<YOUR_SMU_ID>.zip**
  - For example, **midterm_kim.jongun.2019.zip**
  - When unzipped, it must contain:

```
|-- midterm
|       |
|       |-- q1 (all files)
|       |-- q2 (all files)
|       |-- q3 (all files)
```

  - **Incorrect submission file name** WILL attract a penalty of up to **20%** of your score for the entire **midterm assignment**.

- Only **zip** format is accepted.
  - **.7z**, **rar** or other **compression formats** are ***NOT*** accepted.
  - Until the correct **zip** format is submitted again by the student, it will be assumed that the student has NOT made the submission and late submission policy will apply.

- Submit the **zip** file to the following location:
  - **eLearn** page: https://elearn.smu.edu.sg/d2l/home/274235

    IS216-Web Application Development II-Merged Section - 2020-211IS216_MERGED_SECTION

  - **eLearn → Assignments → Midterm Assignment**

Question 1: Savings (Difficulty Level: */)** **[8 marks]**

Given:

```
webroot
    |
    |-- midterm
    |      |
    |      |--- q1
    |      |      |-- savings.html
    |      |      |-- savings.js
```

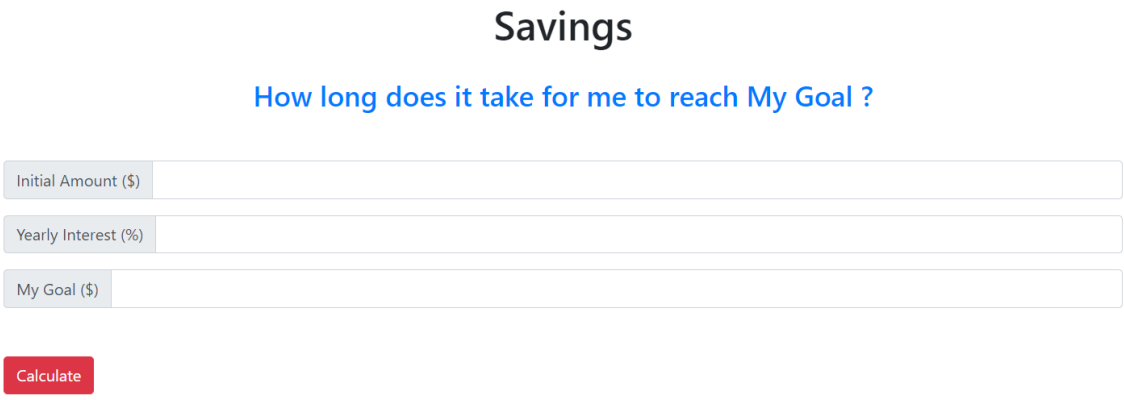
IMPORTANT
You are free to define the HTML elements and CSS style required to match the screenshot to the best of your interpretation ***where the requirement is not stated explicitly***. This includes margin, padding, font style, font size, etc. **`savings.html`** must only contain HTML and CSS code. You must write **ALL** JavaScript code inside **`savings.js`**. JavaScript code written inside **`savings.html`** will ***NOT*** be considered for grading.

Part A: Complete `savings.html` (3 marks)

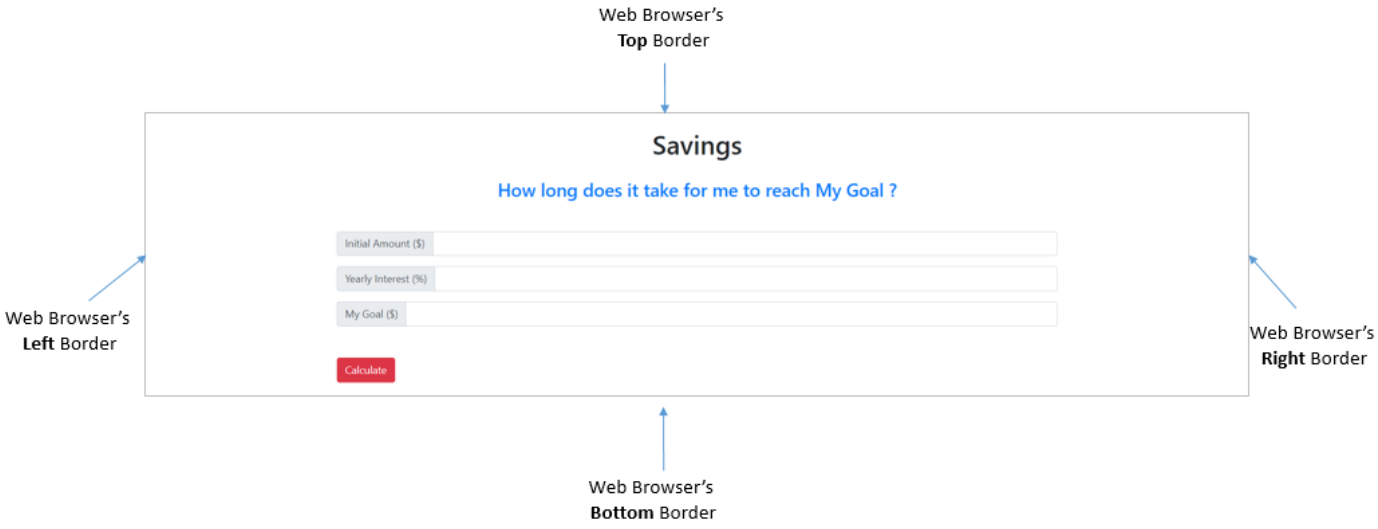
Complete implementation of **`savings.html`** such that when rendered in a web browser, **`savings.html`** displays the following.

NOTE: The three **input fields** must only allow for **numeric values** (not text). You may **ASSUME** that the user will always enter numeric values greater than 0 (thus, no negative values).

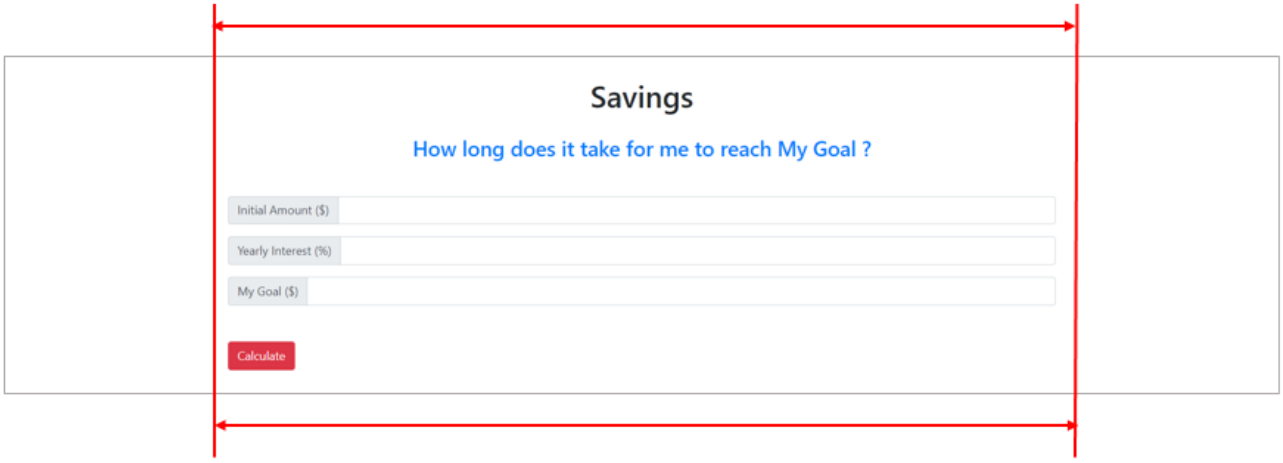
`savings.html`

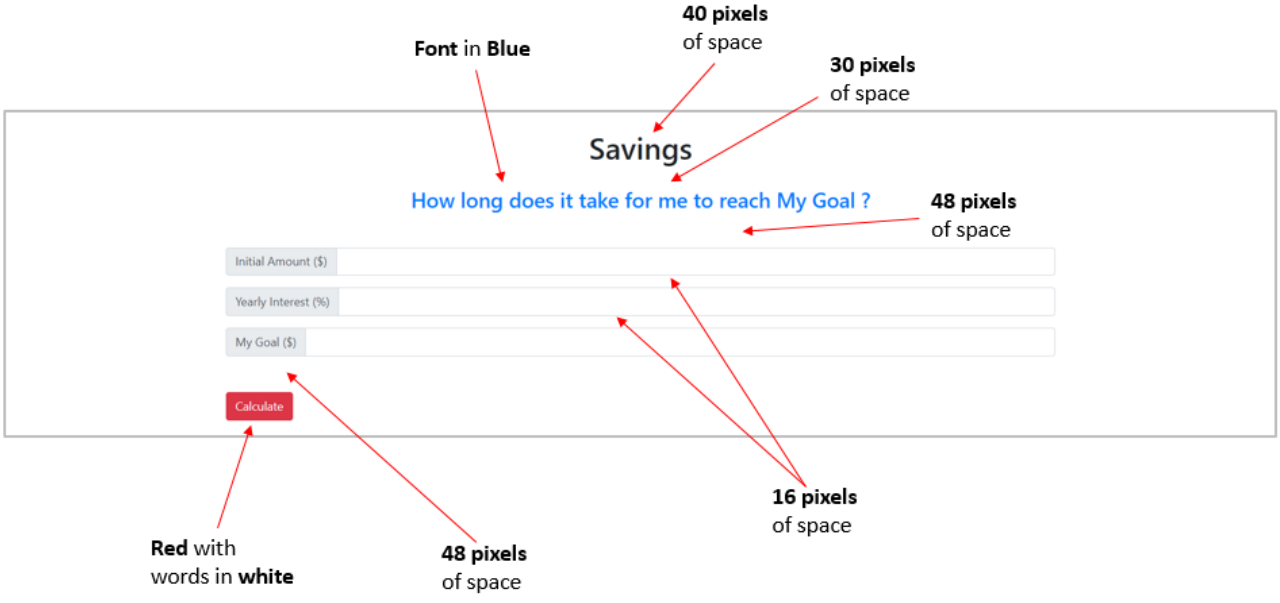


Please take note of the following **user interface layout** guidelines:



The **main content** is centered and spans **70%** of the web browser **width.**

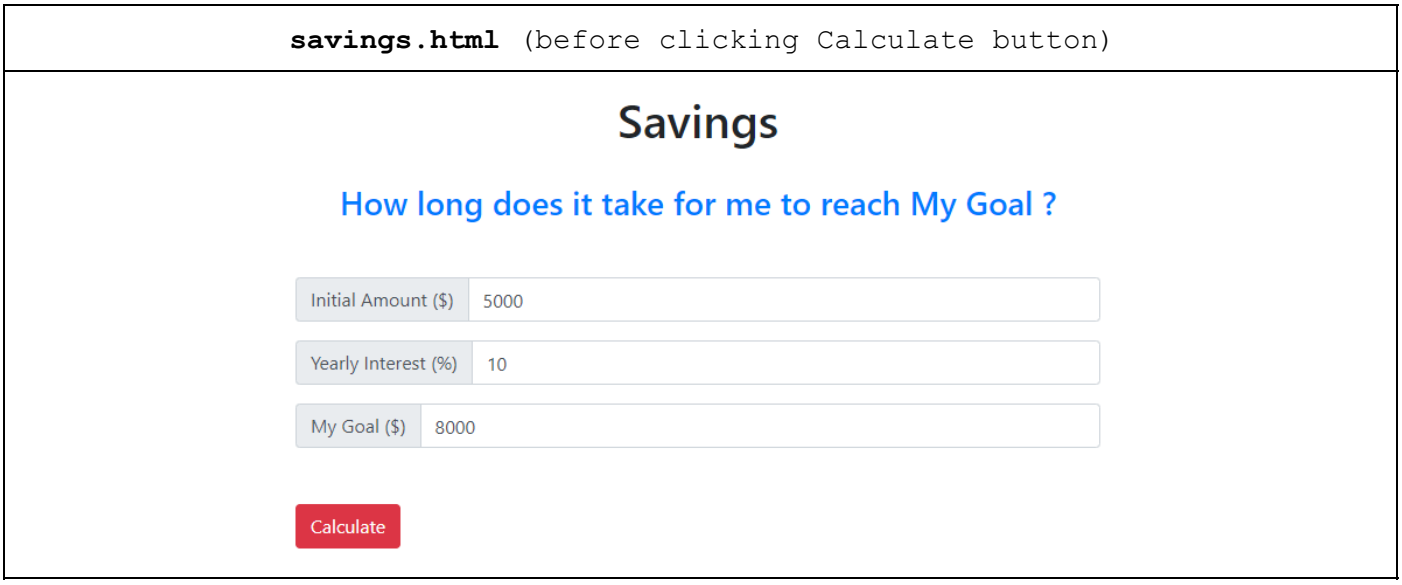




Part B: Complete function `calculate()` (5 marks)

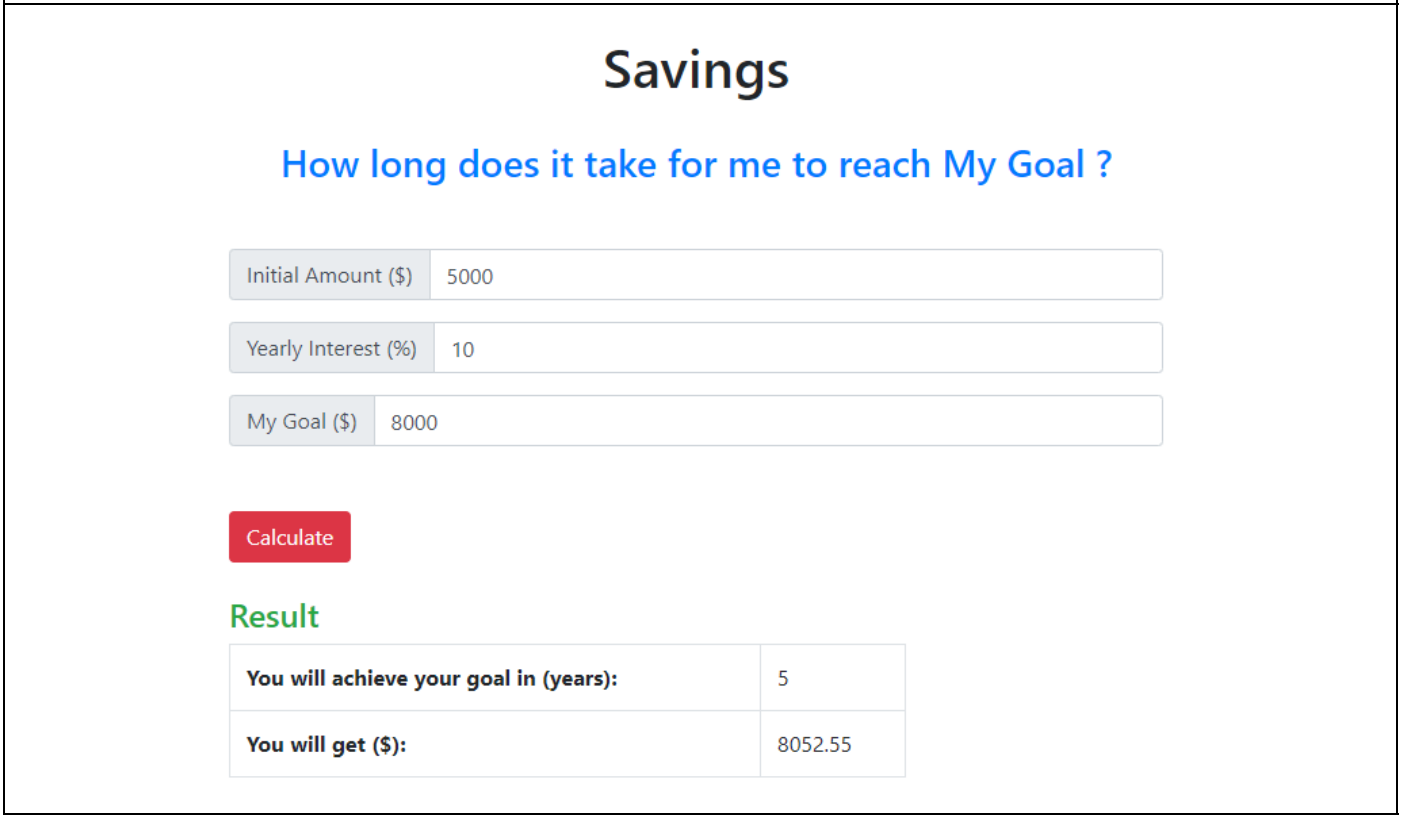
Complete the function **`calculate()`** in **`savings.js`** such that it calculates the **number of years** it will take for the user to achieve **My Goal** (user's goal) given 1) an **Initial Amount ($)** and 2) **Yearly interest (%)**.

`savings.html` (before clicking Calculate button)



1. The user enters Initial Amount = **5000**, Yearly Interest = **10** and a Goal of **8000** in the three input text fields.
2. The user clicks the **Calculate** button.
3. The **results** will be displayed below the **Calculate** button. It will display the **number of years** to achieve the goal (***as an Integer***) and the **amount** you will get in **two decimal places**.

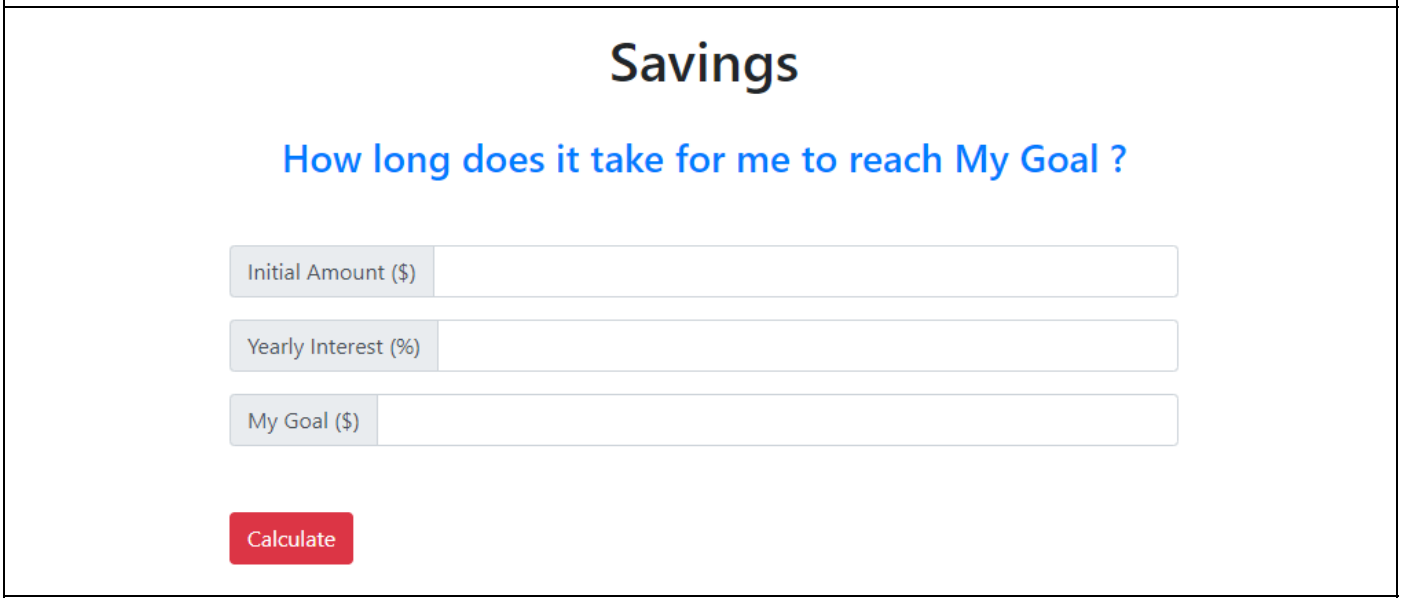
`savings.html` (after clicking Calculate button)



For the **"My Goal" calculation** to work, all three **input field values** are necessary.

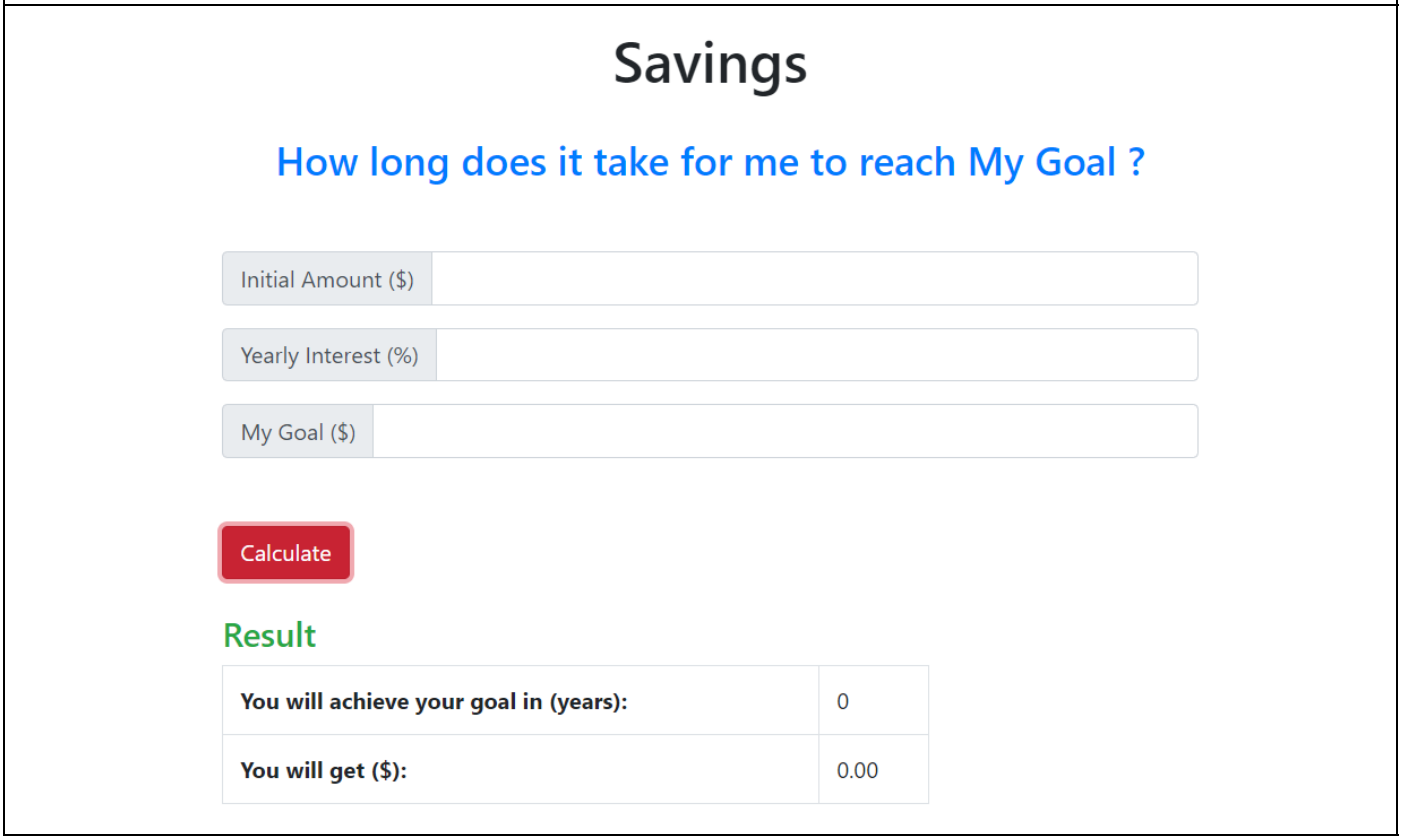
For example, below, we demonstrate a scenario where the user leaves all three input fields EMPTY.

savings.html (before clicking Calculate button)



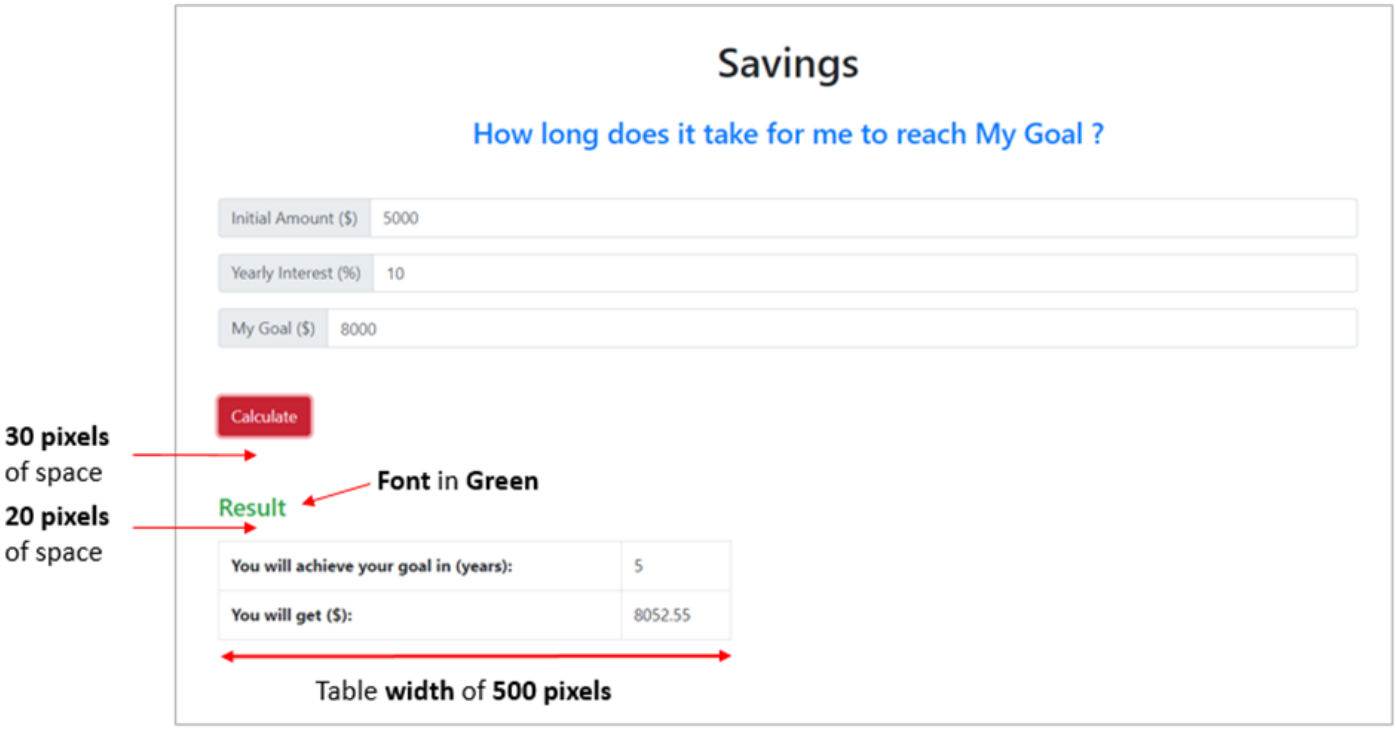
1. The user leaves all three **input fields** empty (no values).
2. The user clicks the **Calculate** button.

savings.html (after clicking Calculate button)



3. The **results** will be displayed below the **Calculate** button (indicating zero).

Please take note of the following **user interface layout** guidelines:



You may refer to the following **test cases** to test your code.

	User Input	Result after clicking Calculate button
1	Initial Amount: **1000** Yearly Interest: **10** My Goal: **1100**	Result You will achieve your goal in (years): 1 You will get ($): 1100.00
2	Initial Amount: **1500** Yearly Interest: **5.2** My Goal: **2000**	Result You will achieve your goal in (years): 6 You will get ($): 2033.23
3	Initial Amount: **1000.50** Yearly Interest: **7.8** My Goal: **800**	Result You will achieve your goal in (years): 0 You will get ($): 1000.50
4	Leaving **any** of the three input fields **EMPTY**	Result You will achieve your goal in (years): - You will get ($): 0.00

Question 2: Grocery (Difficulty Level: */**/**) [11 marks]

Given:

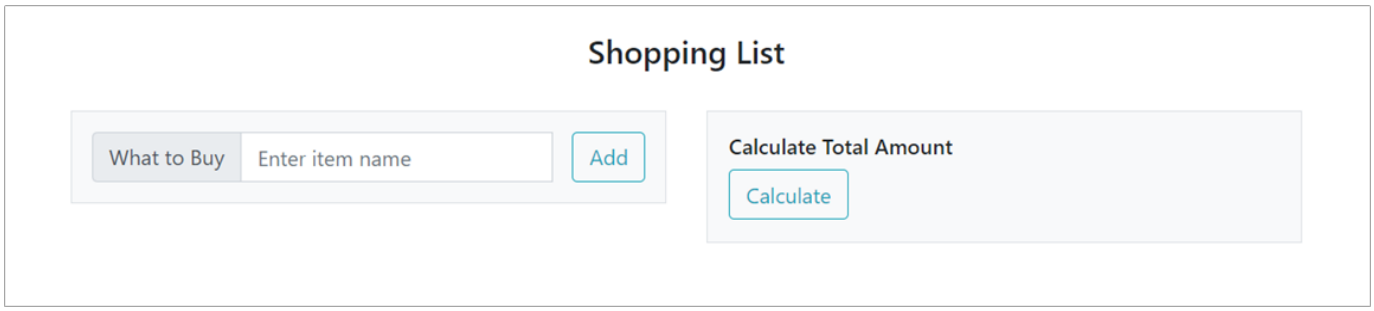
```
webroot
    |
    |-- midterm
    |      |
    |      |--- q2
    |      |      |-- grocery.html
    |      |      |-- grocery.js
```

IMPORTANT
You are free to define the HTML elements and CSS style required to match the screenshot to the best of your interpretation ***where the requirement is not stated explicitly***. This includes margin, padding, font style, font size, etc. **`grocery.html`** must only contain HTML and CSS code. You must write **ALL** JavaScript code inside **`grocery.js`**. JavaScript code written inside **`grocery.html`** will ***NOT*** be considered for grading.

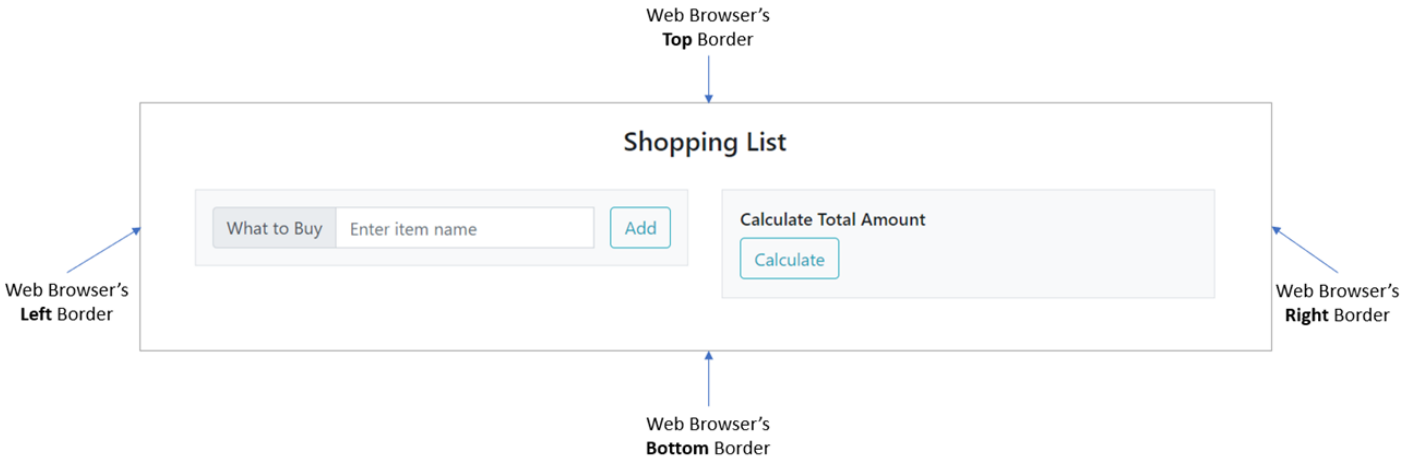
Part A: Complete `grocery.html` (3 marks)

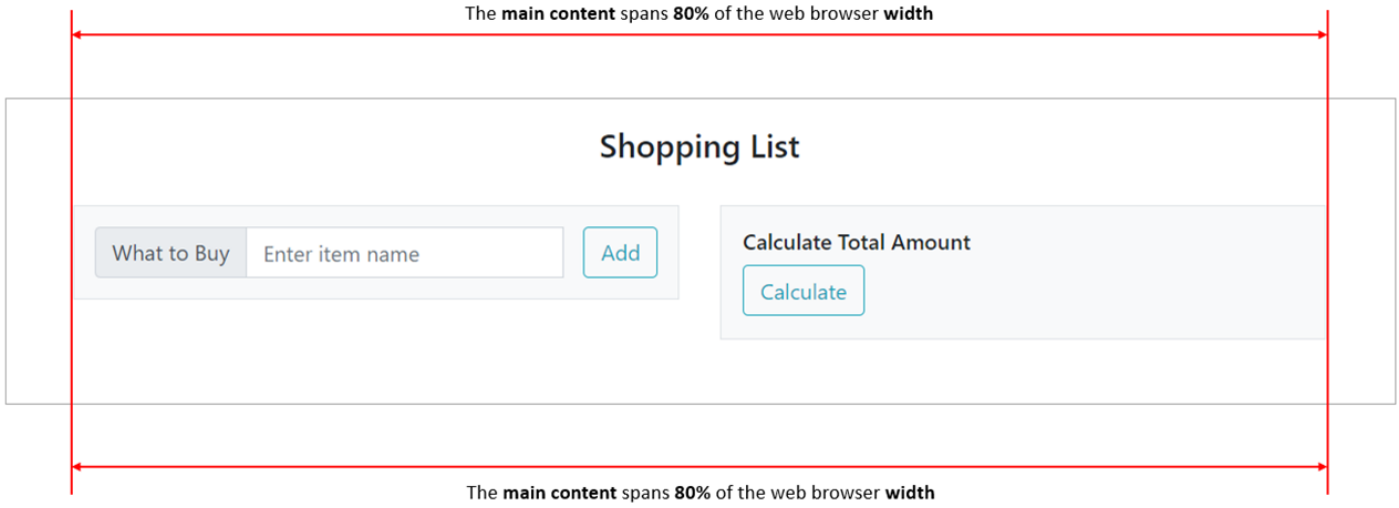
Complete the implementation of **`grocery.html`** such that when rendered in a web browser, **`grocery.html`** displays the following:

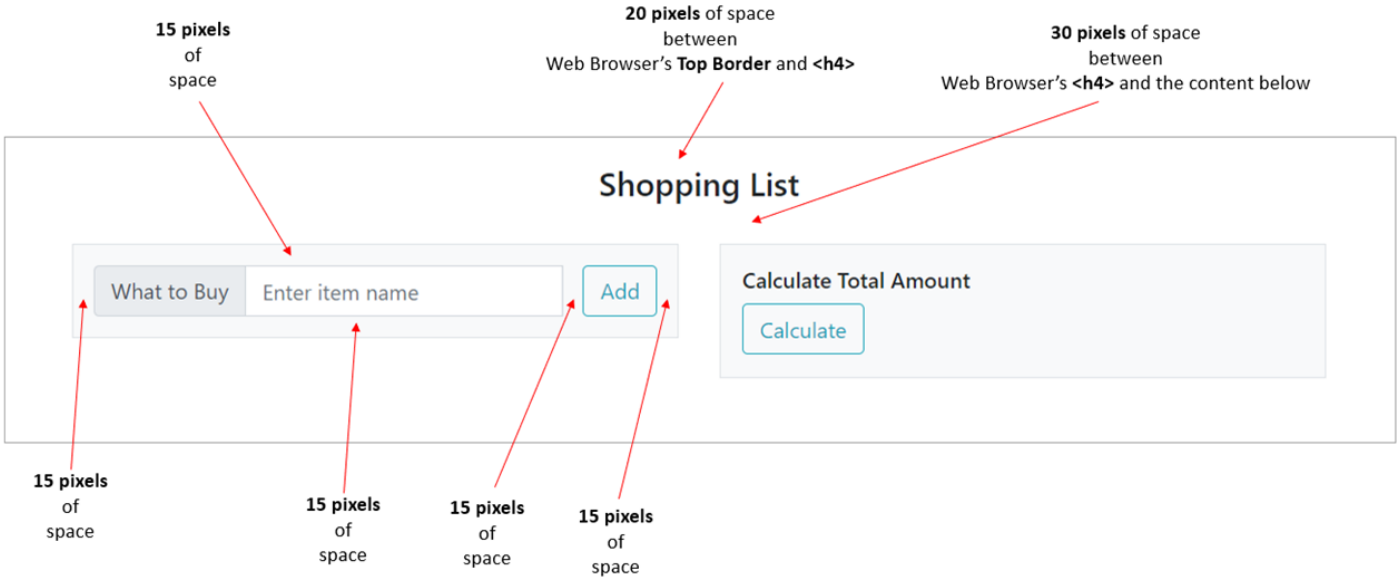
`grocery.html`

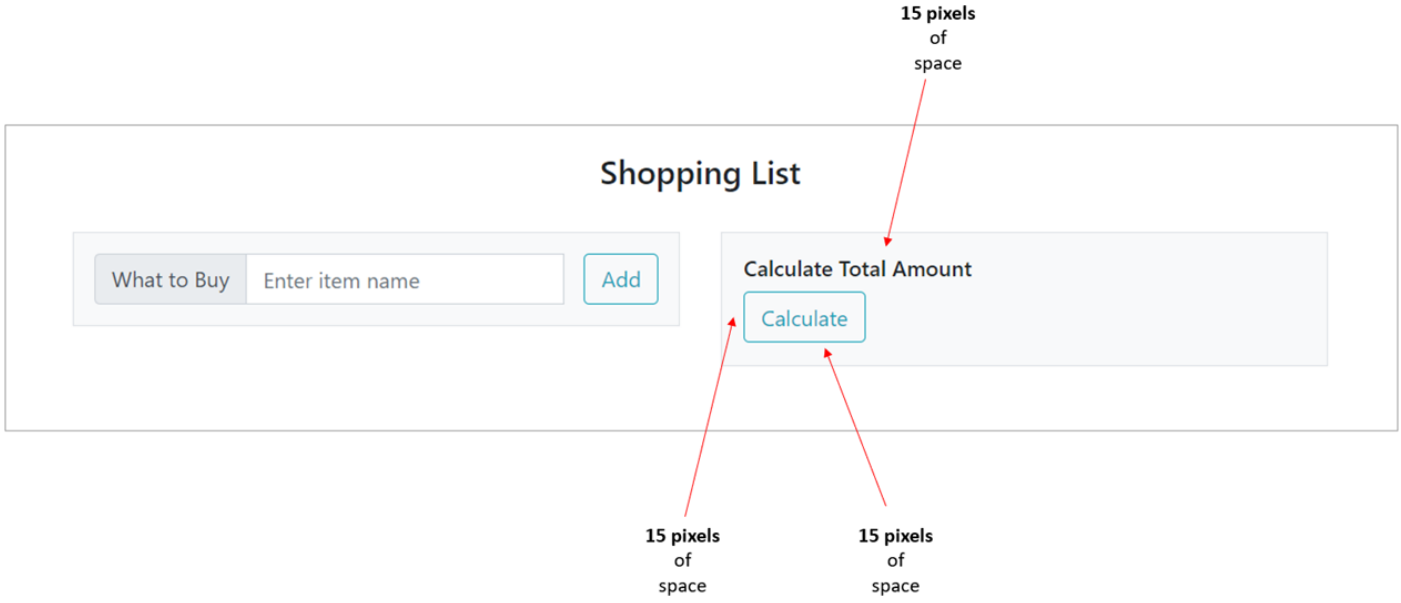


Please take note of the following **user interface layout** guidelines:




The **main content** spans **80%** of the web browser **width**
Shopping List
What to Buy
Enter item name
Add
Calculate Total Amount
Calculate
The **main content** spans **80%** of the web browser **width**

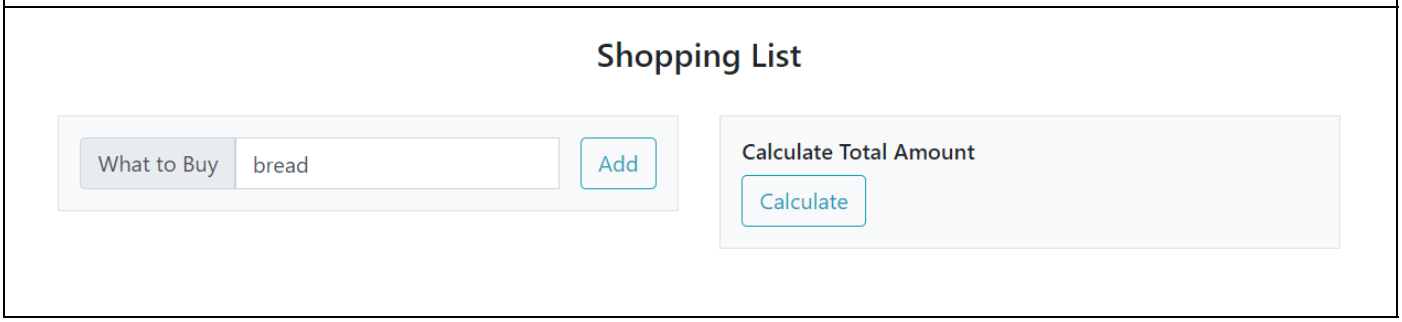

15 pixels of space
20 pixels of space between Web Browser's **Top Border** and **<h4>**
30 pixels of space between Web Browser's **<h4>** and the content below
Shopping List
What to Buy
Enter item name
Add
Calculate Total Amount
Calculate
15 pixels of space
15 pixels of space
15 pixels of space
15 pixels of space


15 pixels of space
Shopping List
What to Buy
Enter item name
Add
Calculate Total Amount
Calculate
15 pixels of space
15 pixels of space

Part B: Complete `grocery.js` (4 marks)

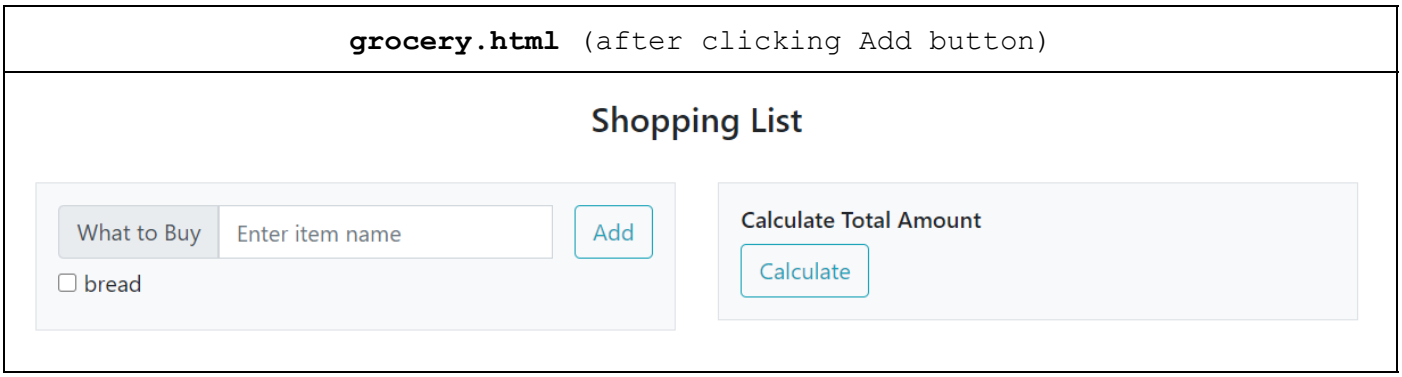
Implement **`addItem()`** and **`processItems()`** such that the following **user scenario** is fulfilled.

`grocery.html` (before clicking Add button)



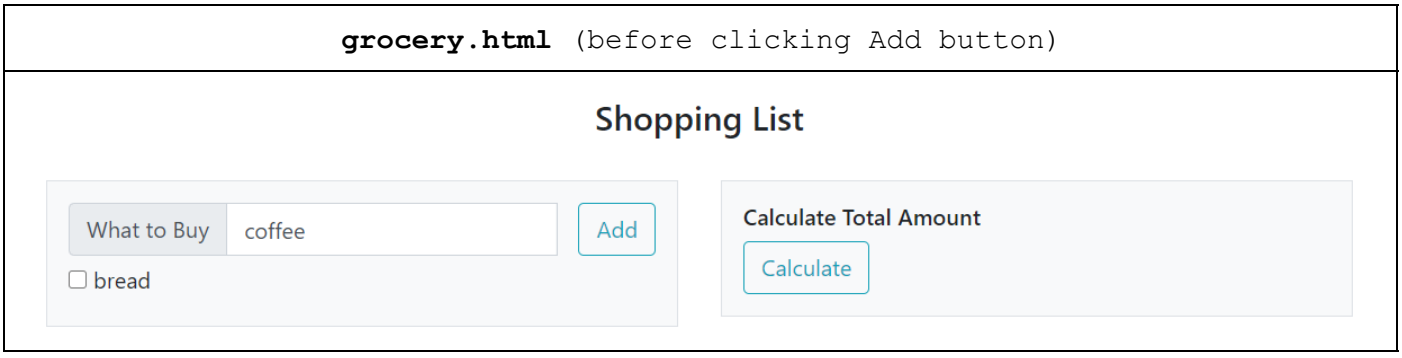
1. The user keys in string/text **bread** in the input text field.
2. The user **clicks** the **Add** button.

`grocery.html` (after clicking Add button)



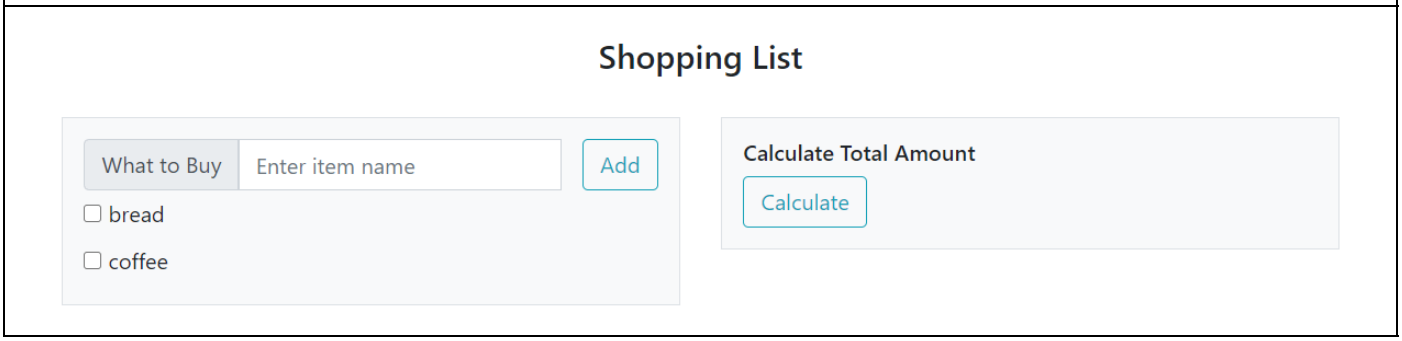
Item **bread** has been **added** with a **checkbox**.
Next, the user will add **one more item**.

`grocery.html` (before clicking Add button)



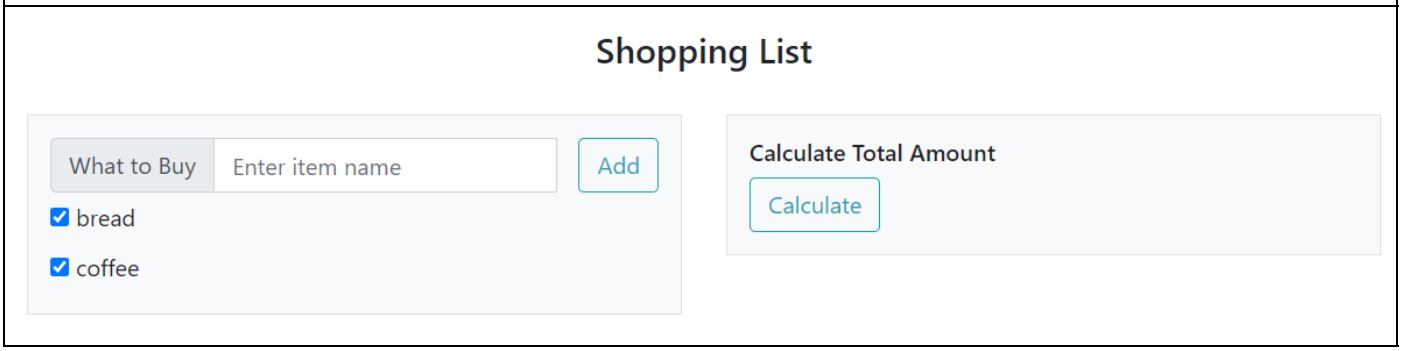
3. The user keys in string/text **coffee** in the input text field.
4. The user **clicks** the **Add** button.

`grocery.html` (after clicking Add button)



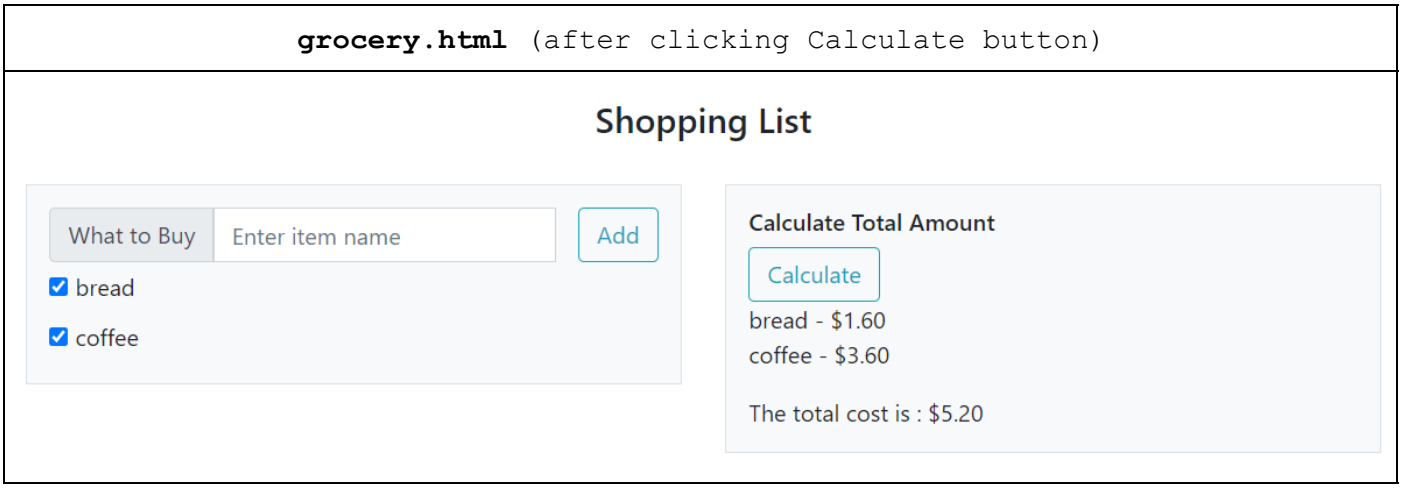
Item **coffee** has been **added** with a **checkbox**.

`grocery.html` (before clicking Calculate button)



5. The user selects both **bread checkbox** and **coffee checkbox**.
6. The user **clicks** the **Calculate** button.

`grocery.html` (after clicking Calculate button)



Below the **Calculate** button, each item's name and its price are listed.

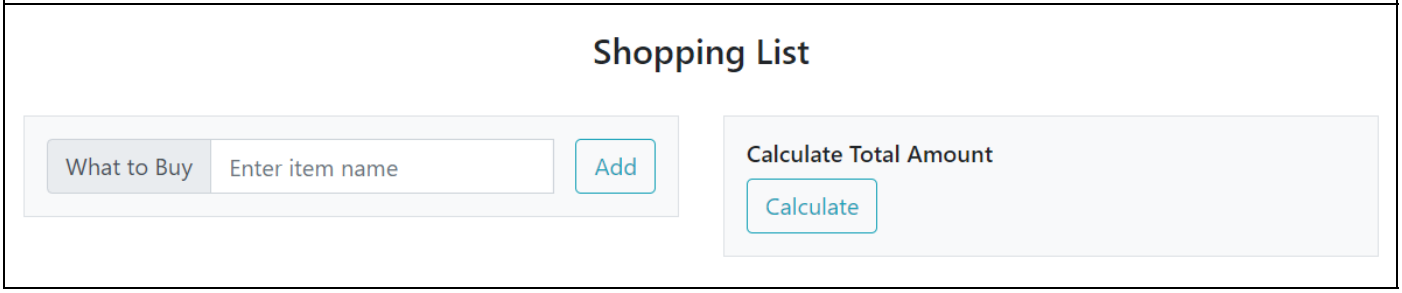
- Each item's **price** information can be found in the **`shopList`** array in **`grocery.js`**.

Furthermore, the **total cost** is calculated and displayed below the item list.

Part C: Complete `grocery.js` (4 marks)

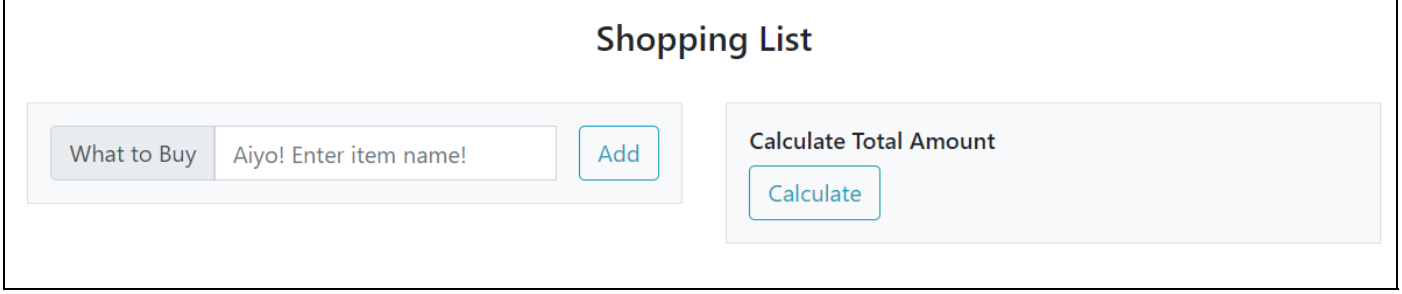
Modify **`addItem()`** and **`processItems()`** such that the following **user scenario** is fulfilled.

`grocery.html` (before clicking Add button)



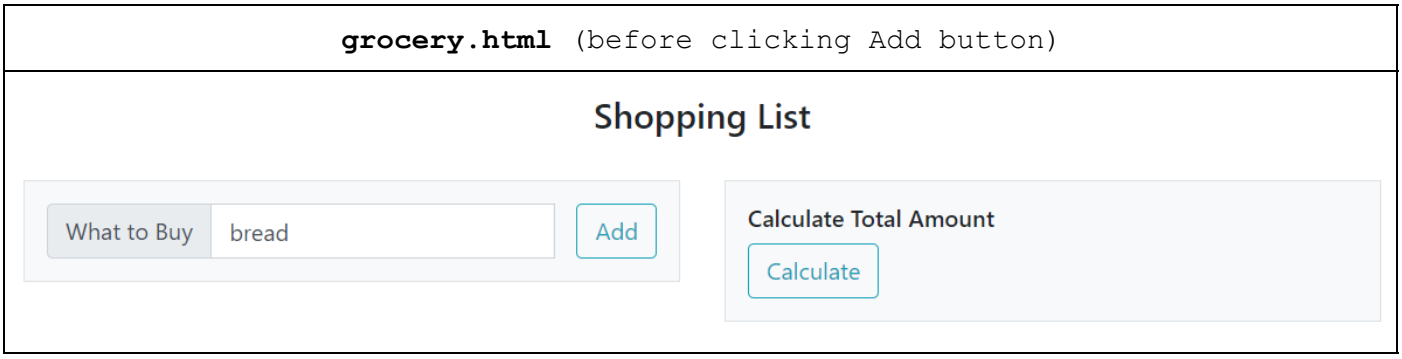
1. The user does NOT key in anything in the input text field (leaving it empty).
2. The user **clicks** the **Add** button.

`grocery.html` (after clicking on Add button)



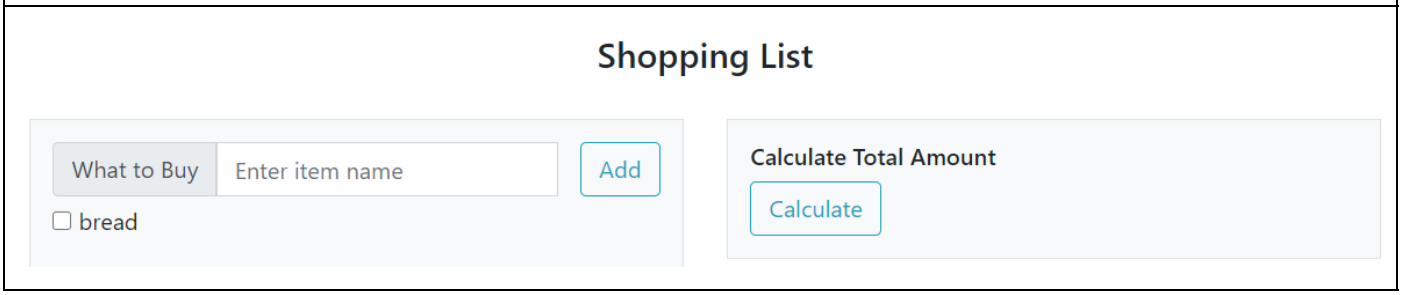
The input text field's **placeholder** value has been updated to **Aiyo! Enter item name!**
Next, the user will add **one item**.

`grocery.html` (before clicking Add button)



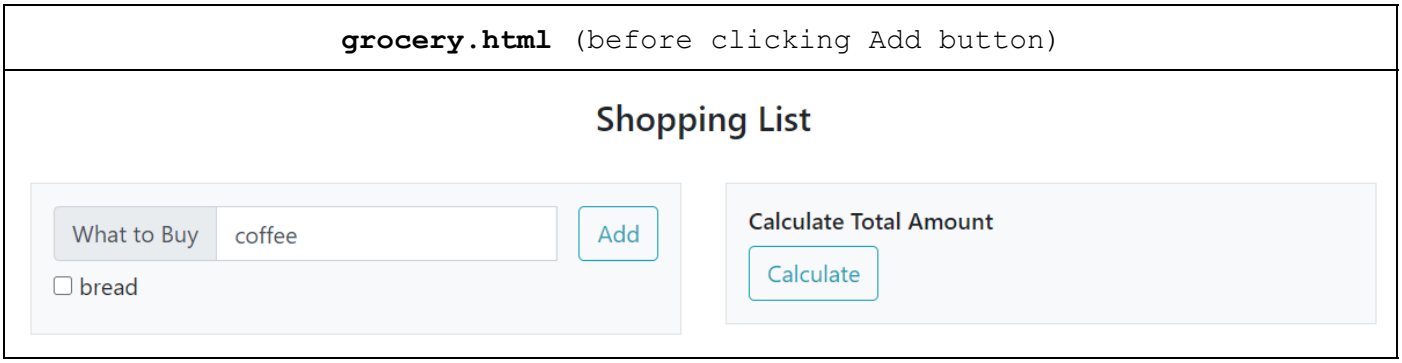
3. The user keys in string/text **bread** in the input text field.
4. The user **clicks** the **Add** button.

grocery.html (after clicking Add button)



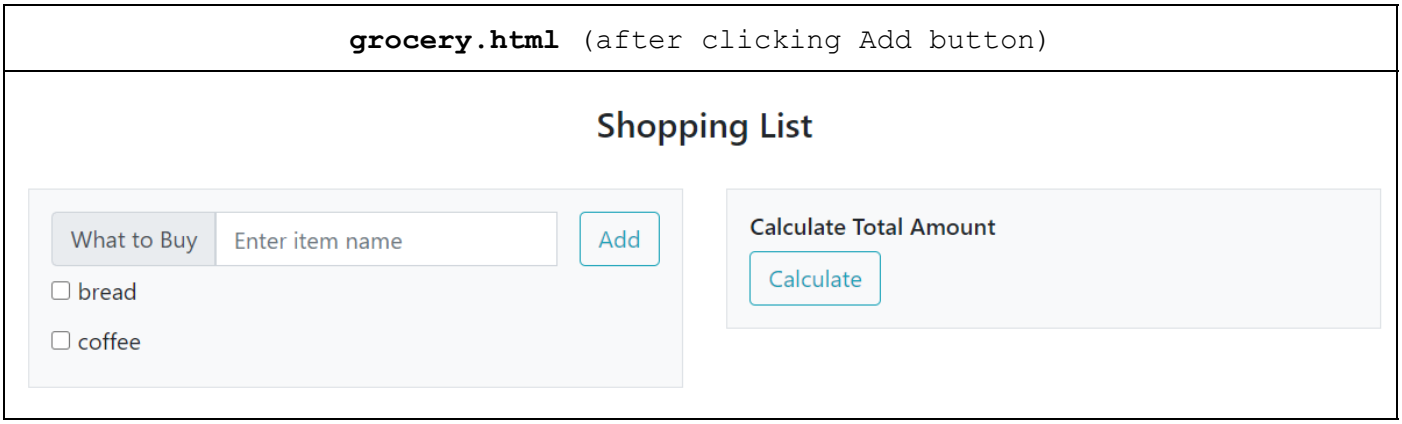
Item **bread** has been **added** with a **checkbox** button.
Next, the user will add **one more item**.

grocery.html (before clicking Add button)



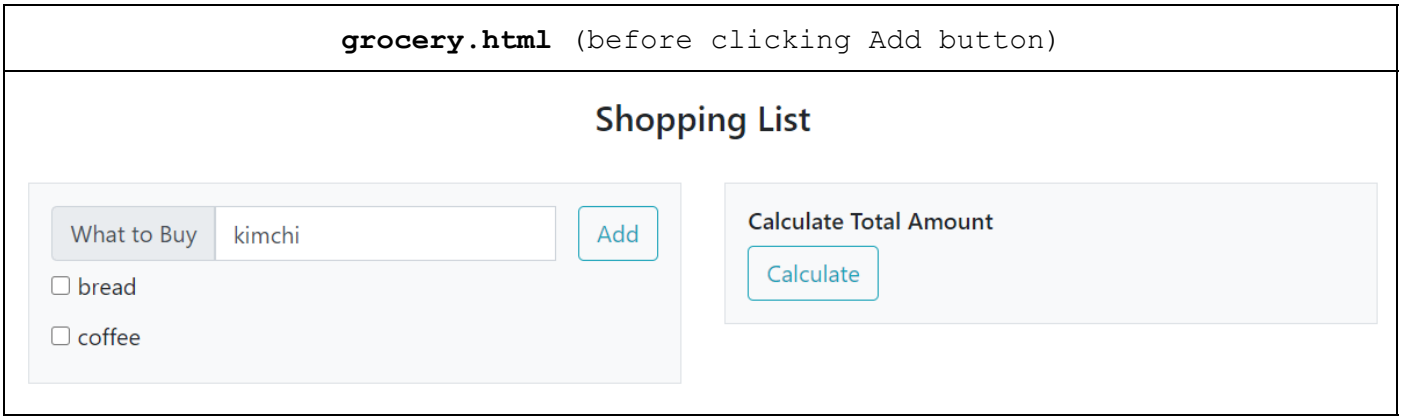
5. The user keys in string/text **coffee** in the input text field.
6. The user **clicks** the **Add** button.

grocery.html (after clicking Add button)



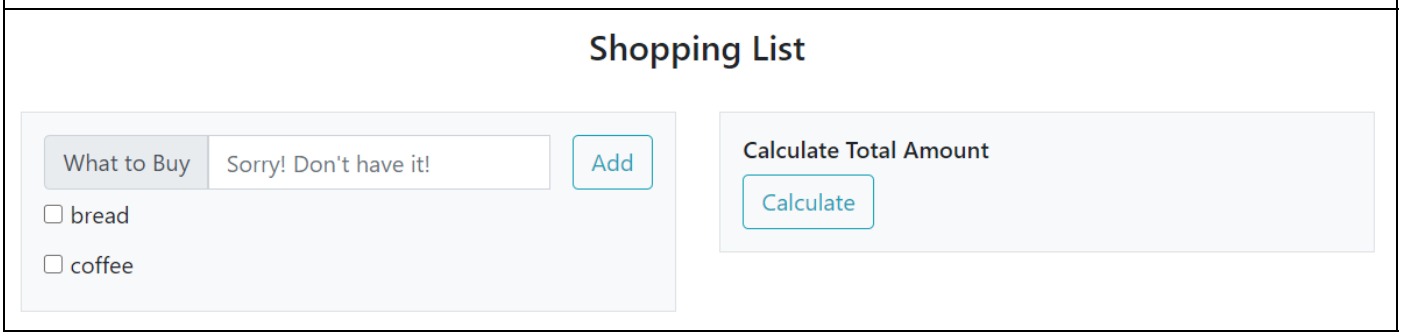
Item **coffee** has been **added** with a **checkbox** button.

grocery.html (before clicking Add button)



7. The user keys in string/text **kimchi** in the input text field.
8. The user **clicks** the **Add** button.

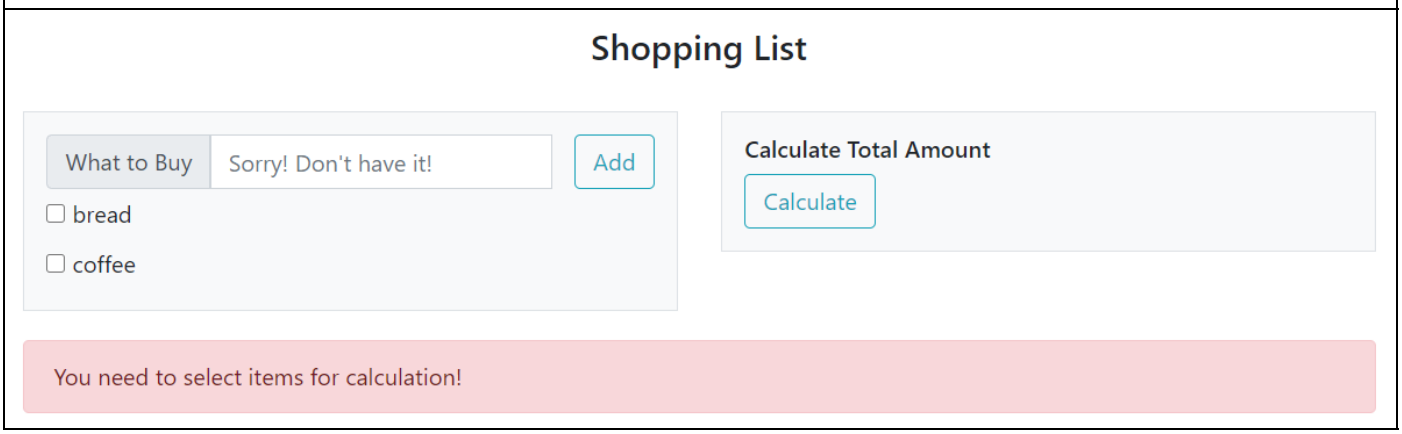
`grocery.html` (after clicking Add button)



Item **kimchi** is ***NOT*** available in this shop (refer to the variable **`shopList`** in **`grocery.js`** file).
The input text field's **placeholder** value has been updated to **Sorry! Don't have it!**

9. The user does NOT select any item **checkboxes**.
10. The user **clicks** the **Calculate** button.

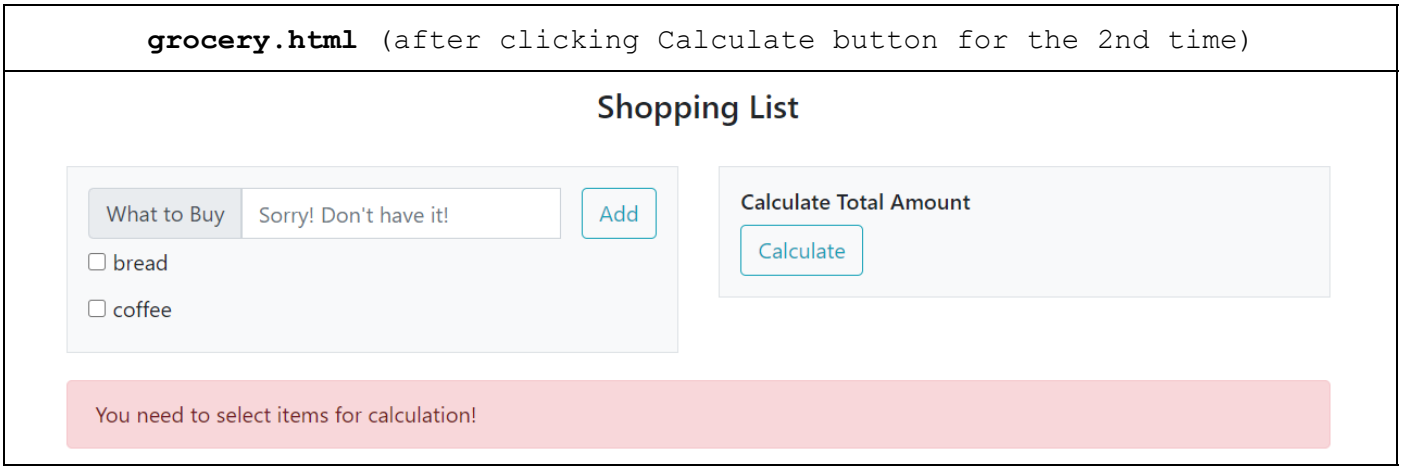
`grocery.html` (after clicking Calculate button)



At the bottom of the webpage, an **alert** message is displayed with text **You need to select items for calculation!**

11. Again, the user does NOT select any item **checkboxes**.
12. The user **clicks** the **Calculate** button.

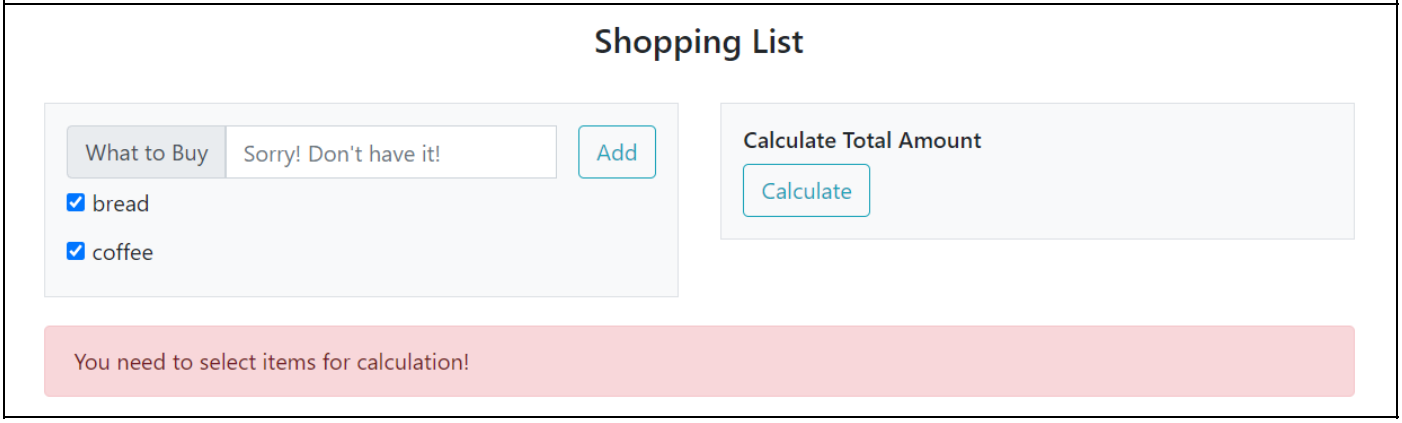
`grocery.html` (after clicking Calculate button for the 2nd time)



At the bottom of the webpage, an **alert** message is displayed with text **You need to select items for calculation!**

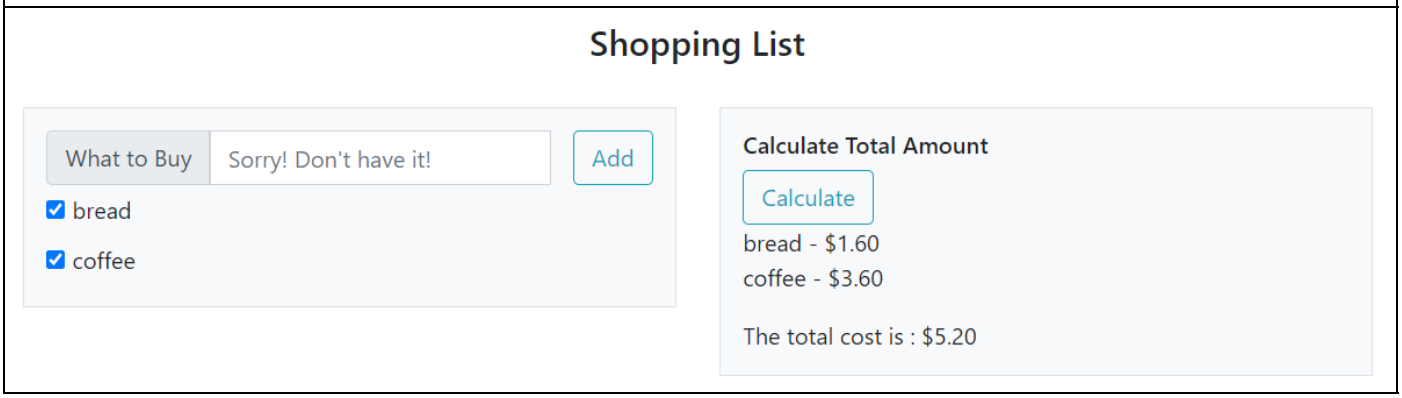
- If the user repeatedly **clicks** on the **Calculate** button without any items selected, **`grocery.html`** must display only **ONE (1) alert notification** at all times.

`grocery.html` (before clicking Calculate button)



13. Finally, the user selects both **bread checkbox** and **coffee checkbox**.
14. The user **clicks** the **Calculate** button.

`grocery.html` (after clicking Calculate button)



Below the **Calculate** button, each item and its price are listed. Furthermore,

- The **total cost** is calculated and displayed, and;
- The **alert notification** is removed from the web page.

Question 3: Oscars (Difficulty Level: */***/***) [11 marks]

Given:

```
webroot
    |-- midterm
    |      |--- q3
    |      |      |-- oscars.html
    |      |      |-- oscars.js
    |      |
    |      |--- api
    |      |      |-- index.html (API Documentation)
    |      |      |-- config
    |      |      |      |-- database.php
    |      |      |
    |      |      |-- db
    |      |      |      |-- load.sql
    |      |      |
    |      |  (and other folders)
```

Configure API's Connection Manager

- In **database.php**, verify that the username, password, and port number are correct for your local computer setup.
- Import **load.sql** into your local MySQL database (via PHPMyAdmin, WorkBench or by other means).
- In your **Google Chrome Web Browser**, go to: http://localhost/midterm/api/ (*if you downloaded the source files elsewhere, you need to go to the correct folder to access the API documentation*).
- You should be able to see the **Oscars API documentation** web page as shown below.

IMPORTANT
You are free to define the HTML elements and CSS style required to match the screenshot to the best of your interpretation ***where the requirement is not stated explicitly***. This includes margin, padding, font style, font size, etc. **`oscars.html`** must only contain HTML and CSS code. You must write **ALL** JavaScript code inside **`oscars.js`**. JavaScript code written inside **`oscars.html`** will ***NOT*** be considered for grading.

This web application (inside **`q3`**) interacts with the API (inside **`api`**). It sends **requests** to the API, retrieves information, and displays the information.

Please check out the **Oscars API documentation** in **api/index.html**. In particular, please see what each **endpoint** returns to the caller. You can use **Postman** to test each API **endpoint** and observe the API **response**. Also, you can visually inspect using the following online **JSON Viewer** apps:

- http://jsonviewer.stack.hu/
- https://codebeautify.org/jsonviewer

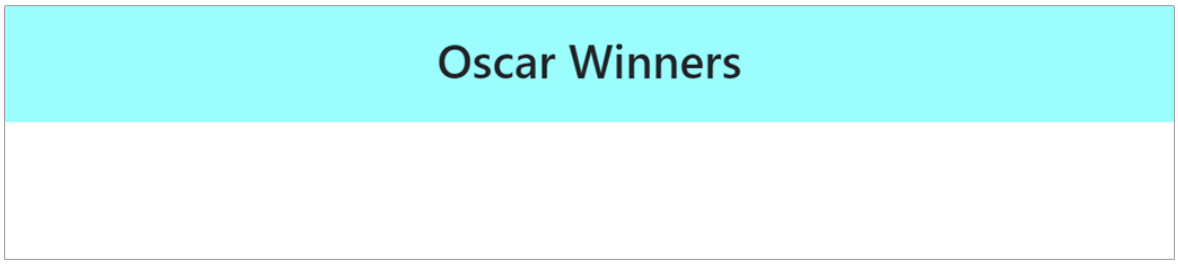
Below is a **snippet** of an **API response** from the API's **"read"** endpoint:


```
▼ object {2}
  ▼ records [104]
    ▼ 0 {4}
        id : 1
      ▼ bio {2}
          name : Emil Jannings
          gender : M
      ▼ movie {3}
          title : The Last Command
          year : 1928
          description : At the first Oscar ceremony, Jannings
                        won two performances. In 'The Last
                        Command,' Jannings plays a former
                        Russian general who ends up as a
                        Hollywood extra. And in 'The Way of
                        All Flesh,' he takes on the role of a
                        bank clerk who is separated from his
                        family for over two decades.
      ▼ others {1}
          image : emil_jannings.jpg
    ▶ 1 {4}
    ▶ 2 {4}
```

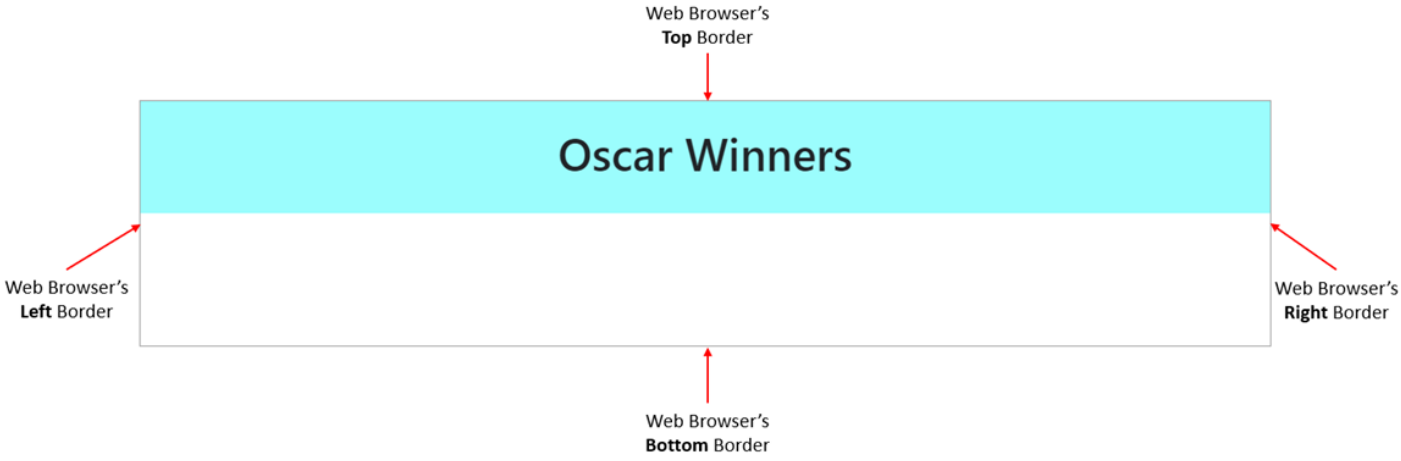

- There are **104 Oscar winners** in the API's database. Each item in the object's **records** is an **Oscar winner**.
- For each **winner**, we can find details such as the **movie** for which the **Oscar award** was given (e.g. movie title, description, and the year in which the actor/actress won the Best Actor/Actress Oscar award).
- Under "others," we can also find the actor/actress' image file name.
    - **NOTE**: The **image files** are stored in the **api**'s **images** folder.

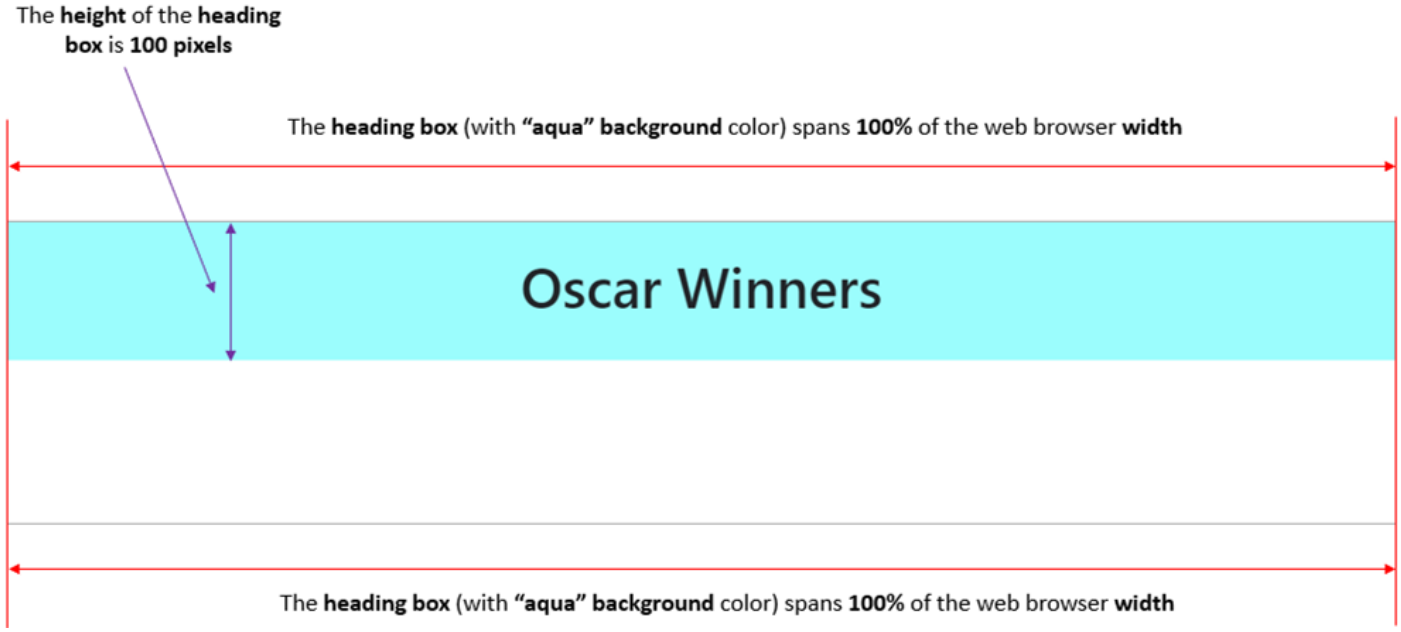
Part A: Complete the top heading in `oscars.html` (1 mark)

Complete implementation of **`oscars.html`** such that when rendered in a web browser, **`oscars.html`** displays the following at the top of the page:

`oscars.html`


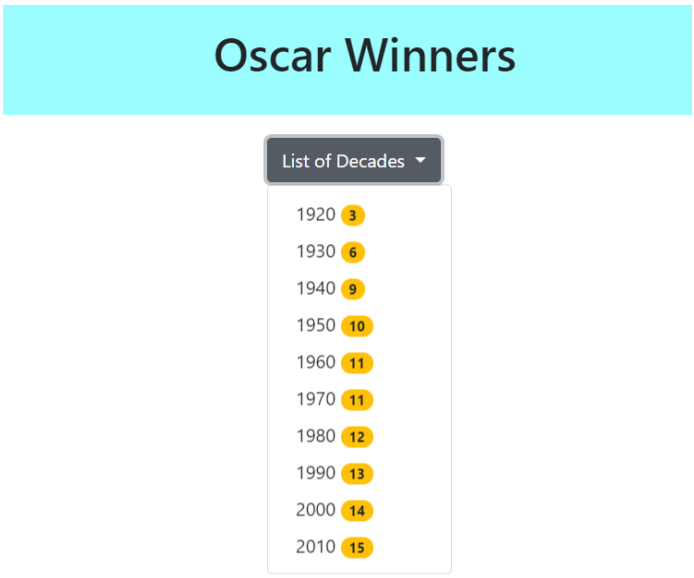
Please take note of the following **user interface layout** guidelines:



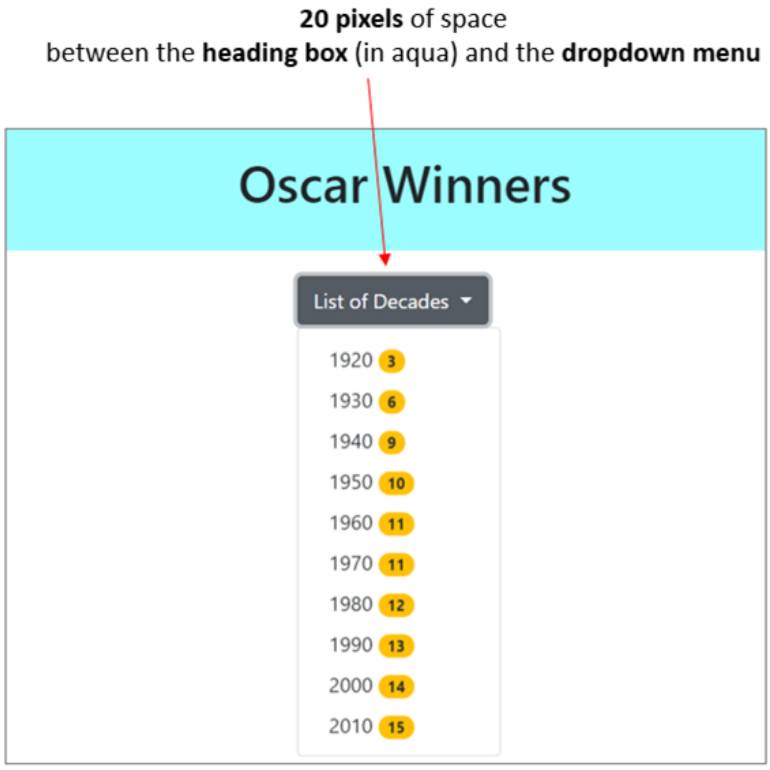


Part B: Complete the dropdown menu in `oscars.html` (3 marks)

Complete implementation of **`oscars.html`** such that when rendered in a web browser, **`oscars.html`** displays the following:

`oscars.html`


Please take note of the following **user interface layout** guidelines:



1. The **dropdown menu** lists different **decades** (1920, 1930… and so on) during which Oscar awards were given.
    a. For example, the **decade** of "1920" encompasses years **1920**, **1921**… all the way to **1929**.
2. The yellow circle on the right hand side of each **decade** denotes the total number of **Oscar winners** in that **decade**.

Make appropriate changes in one or more **source file(s)** so that **`oscars.html`** displays the dropdown menu as shown above.

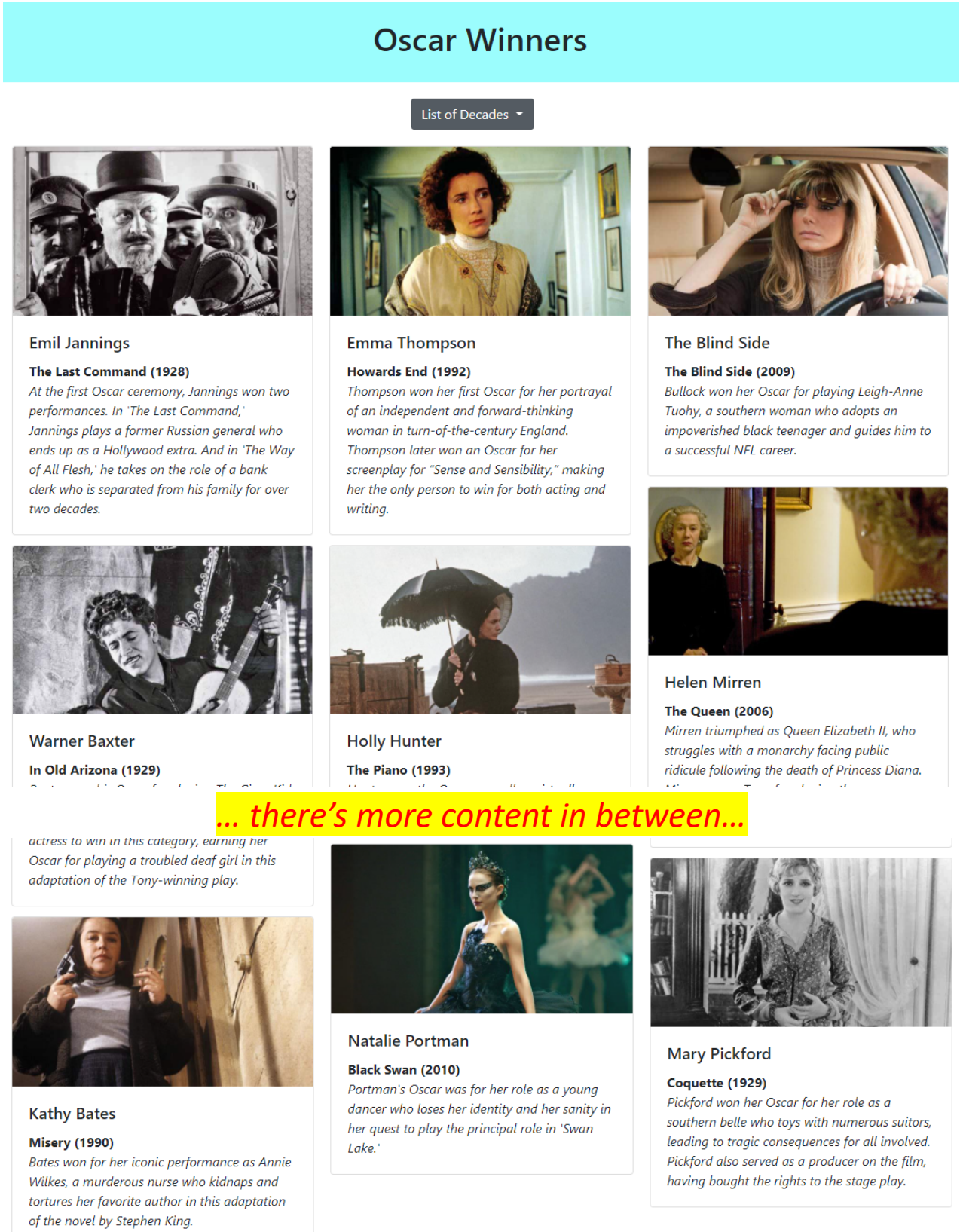
IMPORTANT

1. Do **NOT** hardcode the **decades** (e.g. 1920, 1930, etc.) in the dropdown menu. Do **NOT** hardcode the total number of Oscar winners in the dropdown menu.
    - Your code must dynamically retrieve the values from API response(s).
2. Your code **MUST** use **relative URLs** for **API endpoints** and **images**.
    - For example, an example of a **relative URL** (relative to the current file, e.g. **oscars.html**) is:
        - <img src="../api/images/cher.jpg"> ***---> Use this!!!***
        - <img src="http://localhost/midterm/api/images/cher.jpg"> ***---> DO NOT USE THIS!!!***
    - Failure to do so will attract a penalty of up to **20%** of your score for the entire **Question 3**.

Part C: Complete the main body in `oscars.html` (3 marks)

Complete implementation of **`oscars.html`** such that when rendered in a web browser, **`oscars.html`** displays the following:

`oscars.html`



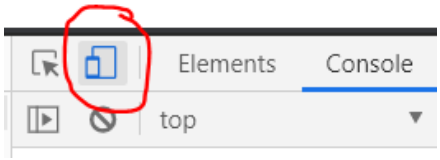
1. When the web page is loaded for the first time, it must display **ALL** available/retrievable **Oscar winners** and display each **winner** in a **Bootstrap card**. There are **104 Oscar winners** in the API's database.

2. Note that not all **cards** have the same **height**. What Bootstrap components/utilities allow for this?
   HINT: You need to check the latest **Bootstrap 4 Documentation** in its **Cards** section.

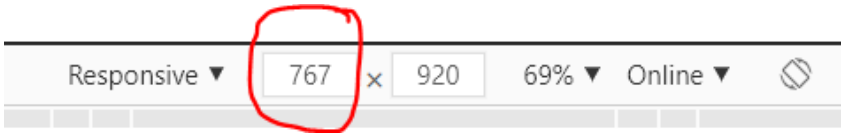
3. Each **card** shows the **details** of an **Oscar winner**. Inside each card, style the content (e.g. Oscar winner's name, movie title, year, and description) appropriately. Here is an example:

*For **Steps 4 and 5**, please take note of the following:*

NOTE: In Google Chrome Web Browser, go to **Developer Tools →** click on the **Toggle Device Toolbar** icon:

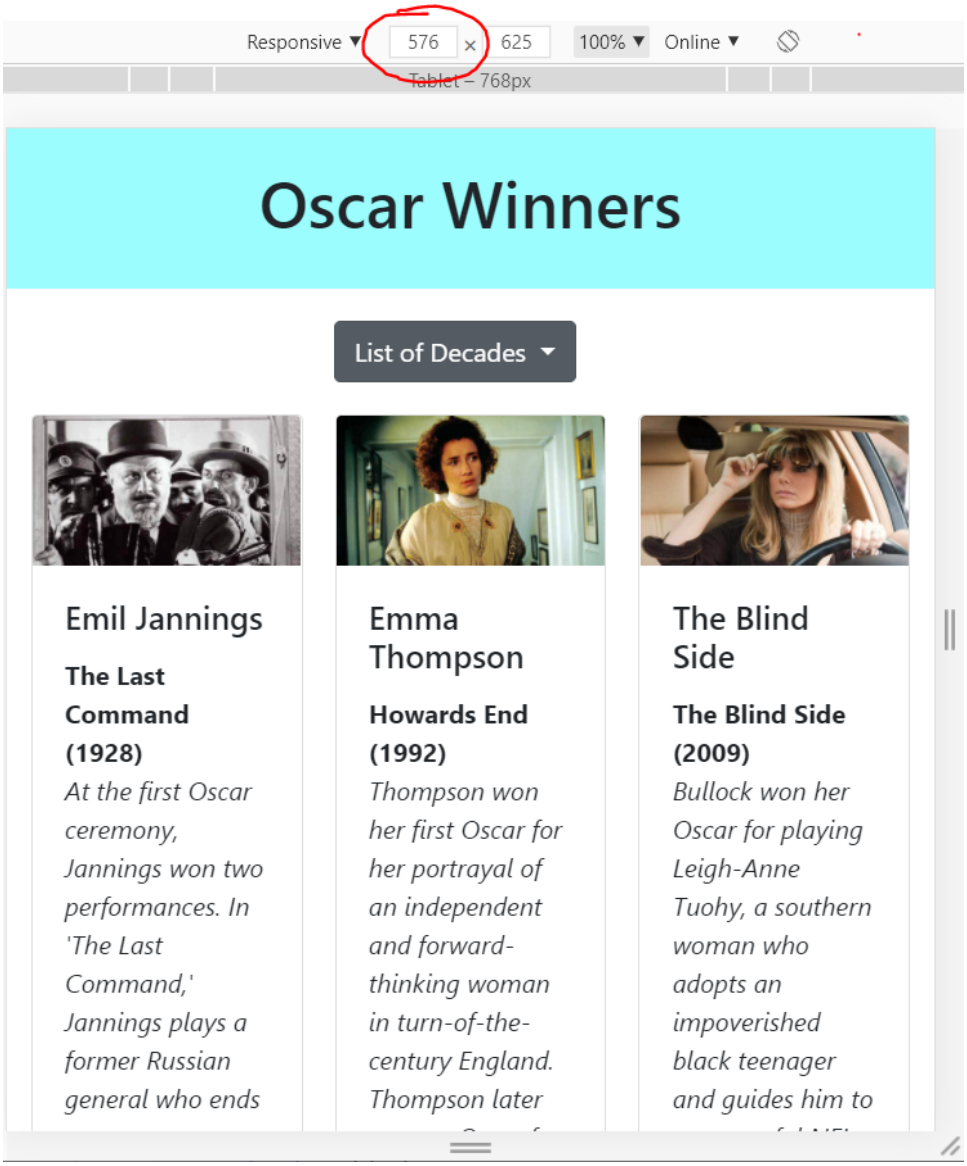


You can test **varying screen dimensions** by modifying the numeric values inside **width** (and height):

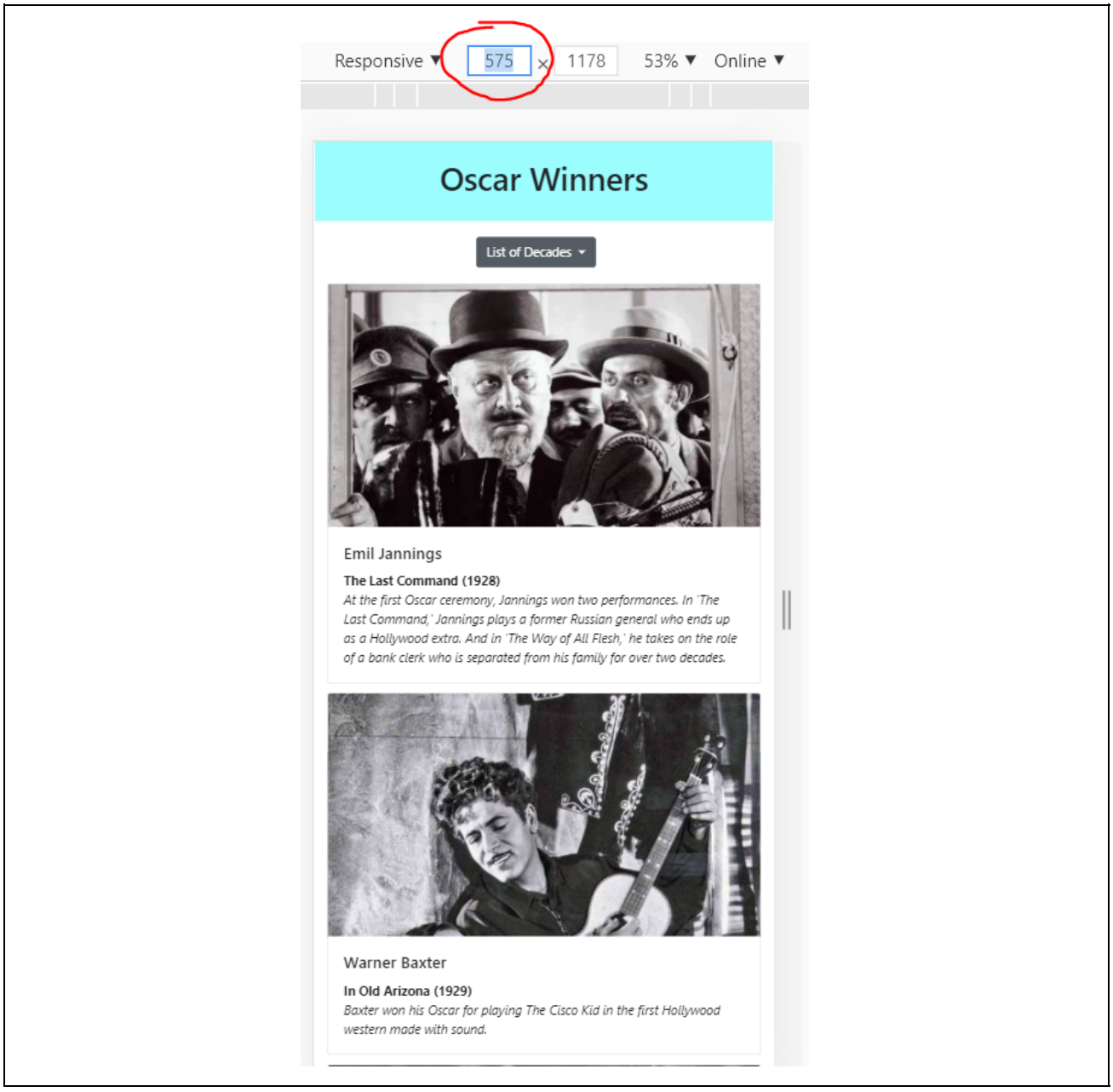


---

4. At screen width of **576 pixels and ABOVE**, the web page displays as follows.



5. At screen width of **575 pixels and BELOW**, the web page displays as follows.



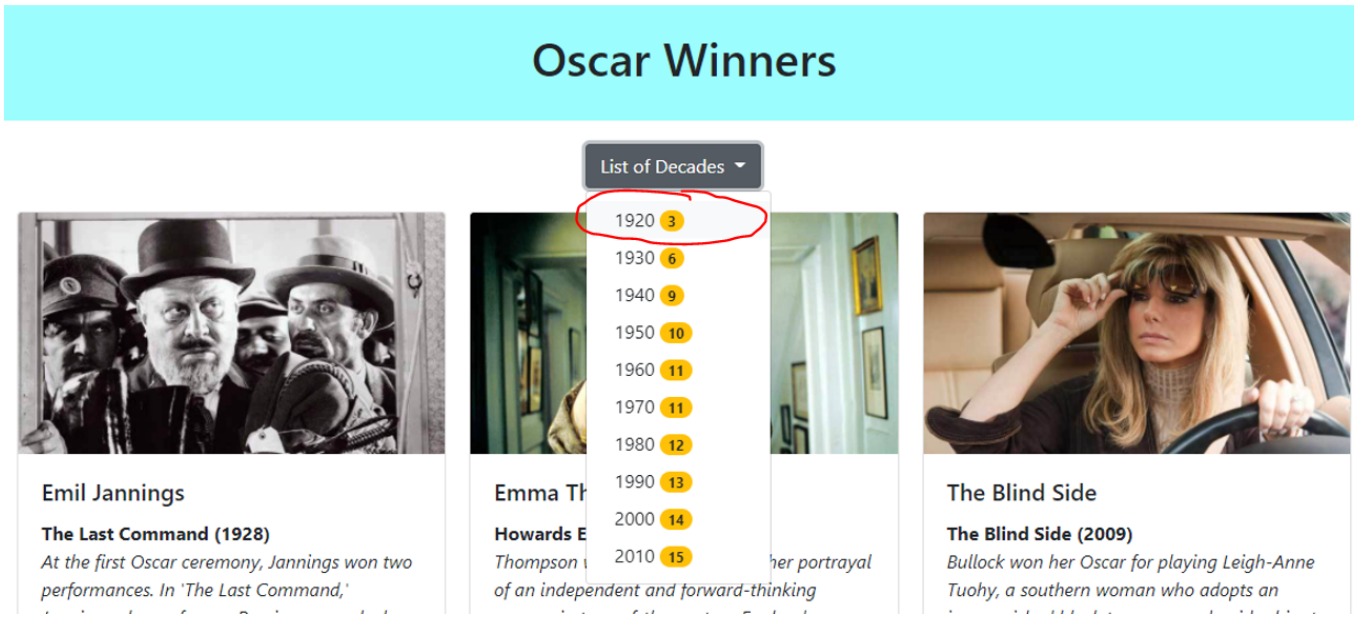
6. Make appropriate changes in one or more **source file(s)** so that `oscars.html` displays as described above.

7. **IMPORTANT**: Your code **MUST** use **relative URLs** for **API endpoints** and **images**.
    - Failure to do so will attract a penalty of up to **20%** of your score for the entire **Question 3**.

Part D: Complete the main body in `oscars.html` (4 marks)

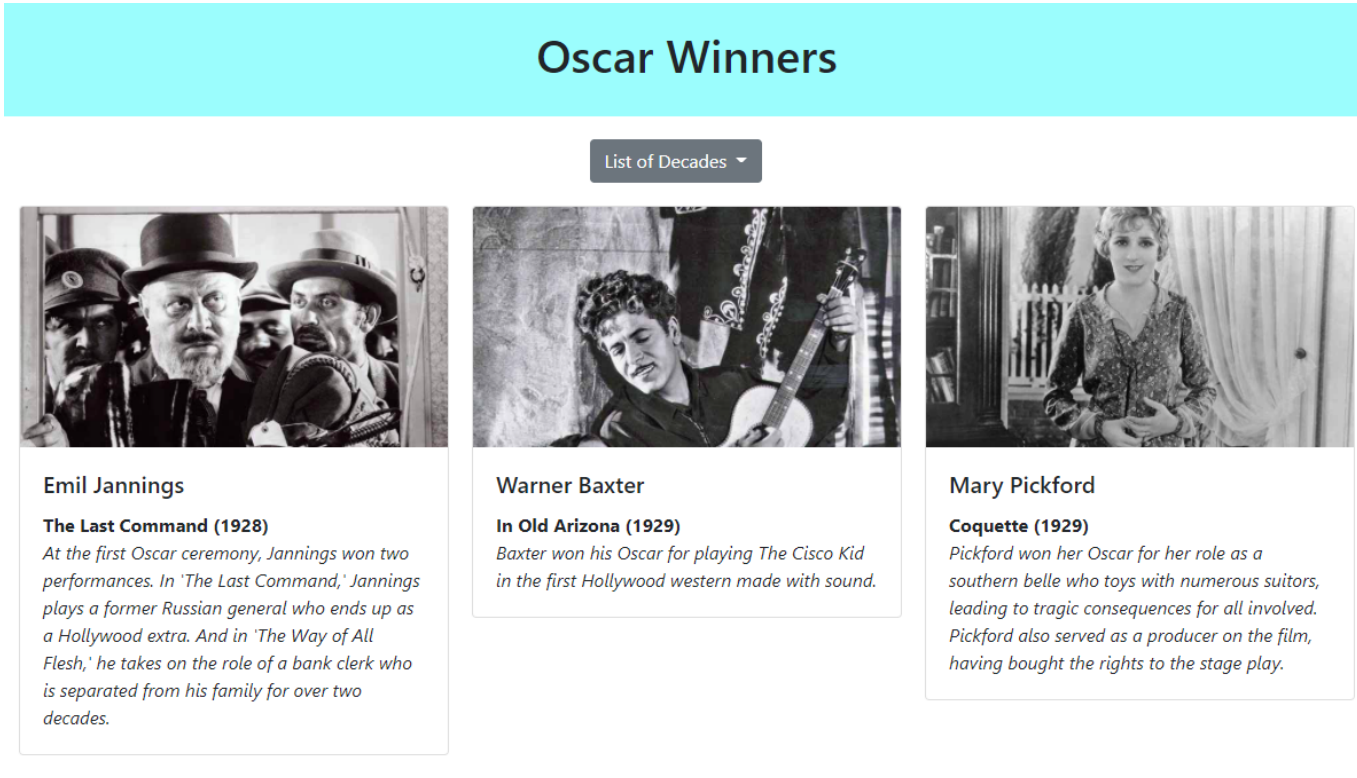
When the user clicks on a **decade** in the **dropdown menu, `oscars.html`** must display only those **Oscar winners** who received the award in **that decade**. For example, below, the user clicks on **1920** (it appears that there are **3 Oscar winners** in that decade). The **red circle** is an *annotation used for illustration* - it is NOT part of the web page.

`oscars.html`



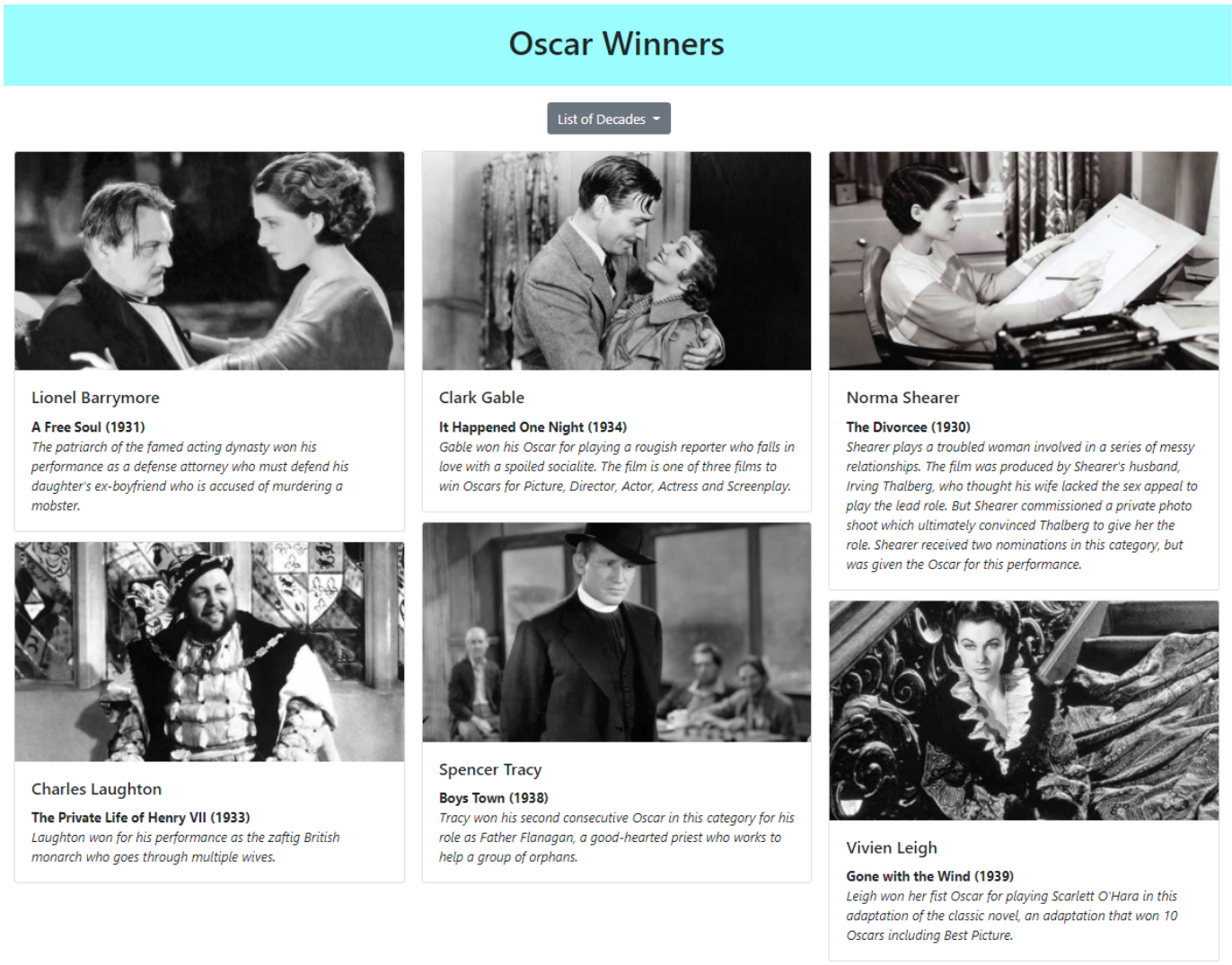
`oscars.html` must display as shown below. Only those **Oscar winners** who received the award in the **1920s** (decade) are displayed.

`oscars.html`



When **1930** is selected in the **dropdown menu**, **`oscars.html`** must display as shown below. Only those **Oscar winners** who received the award in **1930's** (decade) are displayed.

`oscars.html`



Make appropriate changes in one or more **source file(s)** so that **`oscars.html`** displays as described above.

IMPORTANT: Your code **MUST** use **relative URLs** for **API endpoints** and **images**.

3. Failure to do so will attract a penalty of up to **20%** of your score for the entire **Question 3**.

- END -

Appendix A. HTML Cheat Sheet

Adapted from http://www.simplehtmlguide.com/cheatsheet.php

Document outline

<html> ... </html>	HTML document
<head> ... </head>	Page information
<body> ... </body>	Page contents

Section Divisions

...	Division or Section of Page Content
...	Section of text within other content
...	Paragraph of Text
 	Line Break
<hr>	Basic Horizontal Line

Links

 link name 	Create a link to another page or website
<a name="anchor">	Anchor
<a href="#anchor">	Link to anchor

Text Formatting

<h?> ... </h?>	Heading (?= 1 for largest to 6 for smallest, eg h1)
...	Bold Text

** ... **	**Italic Text**
<u> ... </u>	**Underline Text**
<strike> ... </strike>	**Strikeout**
^{...}	**Superscript - Smaller text placed below normal text**
_{...}	**Subscript - Smaller text placed below normal text**
** ... **	**Small - Fineprint size text**
<pre> ... </pre>	**Pre-formatted Text**
** ... **	**Strong - Shown as Bold in most web browsers**
** ... **	**Emphasis - Shown as Italics in most web browsers**

Lists

** ... **	**Ordered List**
** ... **	**Un-ordered List**
** ... **	**List Item (within ordered or unordered)**

Forms

<form> ... </form>	**Form input group declaration**
<input> ... </input>	**Input field within form**
<label> ... </label>	**Input label**
<select> ... </select>	**Select options from drop down list**
<option> ... </option>	**Option (item) within drop down list**
<textarea> ... </textarea>	**Large area for text input**

Tables

<table> ... </table>	**Define a Table**
<tr> ... </tr>	**Table Row within table**
<th> ... </th>	**Header Cell within table row**
<td> ... </td>	**Table Cell within table row**

Image

<img src="filename.jpg">	**Show an image**

Appendix B. CSS Cheat Sheet

Adapted from https://courses.cs.washington.edu/courses/cse154/14sp/cheat-sheets/css-cheat-sheet.pdf

Selectors

div	All DIV tags
div, span	All DIV tags and all SPAN tags
td, th	All table data cells and header cells
#name	Element with ID "name"
.class	All elements with CLASS "class"

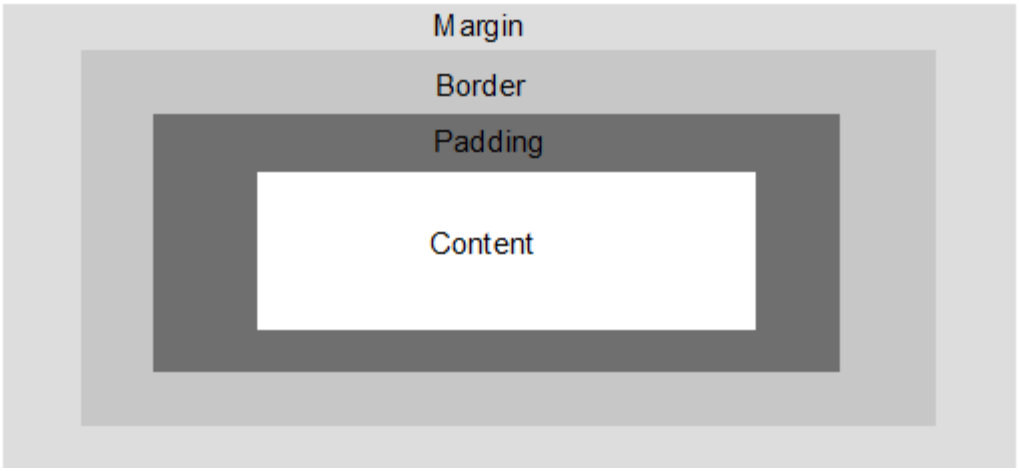
Text

font-family	Font used, e.g., Helvetica, Arial, etc.
font-size	Text size, e.g., 60px
color	Text color, e.g., black, red, #abcdef, etc.
font-weight	How bold the text is, e.g. bold
font-style	What style the text is, e.g., italic, normal
text-decoration	Sets a variety of effects on text, e.g., underline, none, etc.
text-align	How the text is aligned, e.g., center, right
text-indent	Indent of the first line, e.g., 2em
text-transform	Format the text, e.g., uppercase, lowercase, capitalize
vertical-align	Align relative to baseline, e.g., text-top

Borders and Lists

border	**Sets border style for all borders, in the format: border: (solid, dashed, dotted, double) (width) (color), e.g., border: solid 1px black**
border-top	**Sets border style for a specific border (same property syntax used for padding and margin, e.g., margin-top, margin-left, padding-right, padding-bottom, etc.)**
border-bottom	
border-left	
border-right	
list-style-type	**Sets style of bullets, e.g., square**
list-style-position	**Sets how text wraps when bulleted, e.g., outside, inside**
list-style-image	**Sets an image for a bullet e.g., list-style-image:url(img.png)**

Box Model



Positioning

position	**Places elements on screen, e.g., absolute, fixed, relative**
float	**Stacks elements horizontally in a particular direction, e.g., left**
top, left, right, bottom	**Specifies the offsets used in absolute, fixed, and relative positions, e.g., top:10px; left:10px;**
display	**Sets how the element is placed in the document flow, e.g., block, inline, none**
z-index	**Sets the stacking order of elements, e.g., z-index of 1 is below z-index of 2**
overflow	**Sets what happens to content outside of container, e.g., auto, hidden**

Background

background-color	**Sets background color of an element, in the format: background-color: (color), e.g., background-color: blue;**
background-image	**Sets background image to use as the background of an element, e.g., background-image: url("img.png")**
background-position	**Sets the position of the background image, e.g., background-position: right top**

Width and Height

width	**Sets the width and height of an element. These properties may have the following values:** **· auto – default. The browser calculates the height and width**
height	**· length – defines the width/height in px, cm, etc.** **· % - defines the width/height in percentage of the containing block**

Appendix C. Bootstrap Cheat Sheet

Adapted from https://bootstrapcreative.com/resources/bootstrap-4-css-classes-index/

Utility

.bg-*	Background color utility classes: .bg-(light, dark primary, secondary, transparent, white, warning, success, info, danger)
.clearfix	Clear the floats of any child elements. Add this class to the parent element wrapping the floating elements
.d-block	Display the element as a block element
.d-inline	Display the element as an inline element
.d-inline-block	Display the element as an inline element. This class also allows to set a width and height on the element.
.d-none	Hide the element
.flex-*	Change the flexbox items alignment: .flex-(row, row-reverse, column)
.float-*	Float an element. .float-(none, left, right)
.font-*	Style the font. .font-(italic, weight-bold, weight-light, weight-normal, monospace)
.h-*	Height utility class that makes the element a percentage height of its parent element: h-(25,50,78,100, auto)
.justify-content-*	Class specifies where the flex items will be positioned inside the container: .justify-content-(start, end, center, between, around)
.m*-*	Apply margin to an element {property}{sides}-{size}: m(t,b,r,l,x,y)-(0, 1, 2, 3, 4, 5)
.p*-*	Apply padding to an element {property}{sides}-{size}: p(t,b,r,l,x,y)-(0, 1, 2, 3, 4, 5)

.text-*	Align text left, right or center: .text-(left, right, center, justify)
.w-*	Width utility class that makes the element a percentage width of its parent element: w-(25,50,78,100, auto)

Grid

.col	This class is used when you want your columns to be equal width.
.col-*	This class is used for grid columns to determine the column width: .col-(1-12)
.container	Basic layout element in Bootstrap. It is required when using Bootstrap's default grid system.
.container-fluid	For a full width container, spanning the entire width of its parent element.
.row	A parent wrapper of columns

Jumbotron

.jumbotron	Class for showcasing/highlighting key messages in the web page
.jumbotron-fluid	A default jumbotron is not full width but adding this class removes the rounded corners and makes it extend to 100% of its parent element.

Forms

.form-check	The parent class of form checkboxes and radio buttons
.form-check-inline	Class used for a horizontal group of checkboxes or radio buttons
.form-check-input	This class is added to the input tag for checkboxes and radio buttons. Adds styles for positioning and margins.
.form-check-label	This class is added to checkbox and radio button labels

.form-control	This class is used to style textual form controls like <input>, <select>, and <textarea>
.form-group	A div wrapper class used to wrap each form control element for proper margins
.form-inline	This class is added to the form tag for displaying the form elements inline.

Typography

.lead	Increase the font size and line height of a paragraph. Good to use on the first paragraph of an article to improve readability.
.list-inline	Make a list inline
.list-inline-item	Specify the item of an inline list
.list-unstyled	Remove bullet points from a ul or ol list

Tables

.table	Class for styling the table
.table-*	Add or remove table borders. .table-(bordered, borderless)
.table-striped	Adds a light background color to every other table row for a striped effect

Navs

.nav	Base class for styling navigation components
.nav-item	If your nav uses a list, add this class to each list item for proper spacing
.nav-link	Each anchor link inside your nav is given this class in order to have the proper styling
.nav-tabs	Create tab-able regions for navigation components

Navbar

.navbar	Base class for styling a responsive navigation header
.navbar-brand	Class for styling navbar brand
.navbar-collapse	The nav links that are collapsed and shown when toggled on mobile widths.
.navbar-expand-*	Since the navbar is displayed collapse on mobile first, this class specifies what breakpoint you want the navbar to not be collapsed: .navbar-expand-(sm, md, lg, xl)
.navbar-light	Add this class to your navbar if you would like it to have a light background and dark text
.navbar-text	Vertically centers text inside a navbar
.navbar-toggler	For use with collapse plugin and other navigation toggling behaviors

Images

.img-fluid	This class is applied to images you would like to be responsive or fluid width across various screen sizes.
.rounded	Primarily used with images. Add a border radius to the element.

Buttons

.btn	Base class for setting the spacing and size of the button
.btn-*	Buttons with different colored themes: .btn-(light, link, dark primary, secondary, warning, success, info, danger)
.btn-group	Group a series of buttons together horizontally and super-power them with JavaScript
.btn-group-vertical	Group a series of buttons together vertically and super-power them with JavaScript
.btn-lg	Increase the default button size

.btn-outline-*	Transparent background with colored text and border: .btn-outline-(danger|info|primary|secondary|success|warning)

Cards

.card	The class added to the div that wraps each individual card
.card-body	This class is added to the first child div inside the div.card parent
.card-columns	The .card-columns class is added to the wrapping div of of masonry-like collection of cards
.card-img-top	Cap the top of a card with an image
.card-text	Wrap the container around card text
.card-title	This class is added to titles inside cards. It applies the proper spacing.

Carousel

.carousel	Parent class of a carousel
.carousel-caption	Caption for each slide item
.carousel-control-*	When you have an image carousel with pagination you will use this class on the previous and next anchor links: .carousel-control-(prev, next)
.carousel-item	he wrapper class applied to each individual carousel item

Dropdowns

.dropdown	This class gives you the ability to add a dropdown to navbar, tabs, and pills so you can display a dropdown of additional navigation
.dropdown-divider	Add a horizontal line between dropdown link items
.dropdown-item	This class is added to each link item shown in a dropdown menu

.dropdown-menu	Add the default styles for the dropdown menu container
.dropdown-toggle	This class is added to the button that will have the toggle action applied that will hide and show the dropdown menu

Appendix D. JavaScript cheat sheet

Adapted from https://websitesetup.org/javascript-cheat-sheet/

Data Types

Number	`var age = 23`
Variable	`var x`
String	`var a = "string"`
Operation	`var b = 1 + 2 + 3`
Boolean	`var c = true`
Constant	`const MAX = 10`
Object	`var person = {firstName:"John", lastName:"Doe", age:20, isMarried: false}`
Array	`var fruit = ["Banana", "Apple", "Pear"];`

Operators

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (remainder)
++	Increment number by 1
--	Decrement number by 1
==	Equal to

===	Equal value and equal type
!=	Not equal
!==	Not equal value or not equal type
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
?	Ternary operator
&&	Logical and
||	Logical or
!	Logical not

Outputting Data

alert(message)	Outputs data in an alert box in the browser window
confirm(message)	Opens up a yes/no dialog and returns true/false depending on user click
console.log(value)	Writes information to the browser console, good for debugging purposes
document.write(value)	Writes directly to the HTML document
prompt(message)	Creates a dialogue for user input

Global Functions

decodeURI(encodedURI)	**Decodes a Uniform Resource Identifier (URI) created by encodeURI**
encodeURI(uri)	**Encodes a URI into UTF-8**
eval(string)	**Evaluates JavaScript code represented as a string**
isNaN(value)	**Determines whether a value is NaN or not**
Number(value)	**Returns a number converted from its argument**
parseFloat(value)	**Parses an argument and returns a floating point number**
parseInt(value)	**Parses its argument and returns an integer**

Loops

for	**The most common way to create a loop in JavaScript**
for … in	**loops through the properties of an object**
for … of	**Loops through the values of an iterable variable**
while	**Sets up conditions under which a loop executes**
do while	**Similar to the while loop but it executes at least once and performs a check at the end to see if the condition is met to execute again**
break	**Used to stop and exit the cycle at certain conditions**
continue	**Skip parts of the cycle if certain conditions are met**

Number Methods

toFixed([digits])	**Returns the string of a number with a specified number of decimals. If no argument, 0 decimal places.**
toString()	**Returns a number as a string**

Math Methods

Math.ceil(x)	The value of x rounded up to its nearest integer
Math.floor(x)	The value of x rounded down to its nearest integer
Math.max(x,y,z,...,n)	Returns the number with the highest value
Math.min(x,y,z,...,n)	Same for the number with the lowest value
Math.pow(x,y)	x to the power of y
Math.random()	Returns a random number between 0 and 1
Math.round(x)	The value of x rounded to its nearest integer

String Methods

charAt(index)	Returns a character at a specified position inside a string
charCodeAt(index)	Returns the unicode of a character at that position
concat(s1,s2,…)	Concatenates (joins) two or more strings into one
fromCharCode(num)	Returns a string created from the specified sequence of UTF-16 code units
includes(search[, start])	Checks if the string contains a specified string. True/False
indexOf(search[, start])	Provides the position of the first occurrence of a specified text within a string. Return index or -1 if not found.
lastIndexOf(search[, startIdx])	Same as indexOf() but with the last occurrence, searching backward. Return index or -1 if not found.
match(regex)	Retrieves the matches of a string against a search pattern. Return an Array of matches. If no match, null.

replace(pattern, newStr)	**Finds and replaces specified text in a string.** **Return a new string.**
search(regex)	**Executes a search for a matching text and returns its position. Return index or -1 if not found.**
slice(start[, endExclude])	**Extracts a section of a string and returns it as a new string**
split([separator])	**Splits a string object into an array of strings at a specified position**
substr(startIdx[, length])	**Similar to slice() but extracts a substring depending on a specified number of characters**
substring(start[, endExclude])	**Also similar to slice() but can't accept negative indices**
toLowerCase()	**Converts string to lower case**
toUpperCase()	**Converts string to upper case**

Array Methods

concat(arr1,arr2,…)	**Joins several arrays into one**
indexOf(element[,start])	**Returns the first position at which a given element appears in an array. Return index or -1 if not found.**
join([separator])	**Combines elements of an array into a single string and return the string**
pop()	**Removes the last element of an array**
push(element1,element2,…)	**Adds a new element at the end**
reverse()	**Sorts elements in a descending order**
shift()	**Removes the first element of an array**

slice(start[,endExclude])	Pulls a copy of a portion of an array into a new array. Does not affect the original array.
sort()	Sorts elements alphabetically
splice(start[,delCount[,item1,…]])	Adds/removes elements in a specified way and position. Returns array of deleted elements.
toString()	Converts elements to strings. E.g. [1,2,3].toString() gives "1,2,3"
unshift(element1,element2,…)	Adds a new element to the beginning

DOM Methods

appendChild(child)	Adds a new child node to an element as the last child node
createElement(tagName)	Creates an HTML element
remove()	Removes a node
removeChild(child)	Removes a child node from an element
getAttribute(attrName)	Returns the specified attribute value of an element
getAttributeNode(attrName)	Gets the specified attribute node
removeAttribute(attrName)	Removes a specified attribute from an element
removeAttributeNode(attrNode)	Removes a specified attribute node and returns the removed node
setAttribute(attrName,value)	Sets or changes the specified attribute to a specified value
setAttributeNode(attrNode)	Sets or changes the specified attribute node
getElementById(id)	Gets the element with the specified ID

getElementsByTagName(tagName)	**Provides a collection of all child elements with the specified tag name**
getElementsByClassName(className)	**Gets a collection of nodes with the specified class**
addEventListener(eventName,function)	**Attaches an event handler to the specified element**

Accessing and Modifying HTML Element's Properties

element.innerHTML = new content	**Changes the inner HTML of an element**
element.attribute = new value	**Changes the attribute value of an HTML element**
element.style.property = new style	**Changes the style of an HTML element**
document.forms[name]	**Gets the form element with the specified name**
document.forms[name].elements[field]	**Gets the form field element with the specified name**

Event Handlers

onchange	**The event occurs when the content of a form element changes (for <input>, <select> and <textarea>)**
onclick	**The event occurs when the user clicks on an element**
onfocus	**The event occurs when the element gets focus**
oninput	**The event occurs when the user input on an element**
onkeydown	**The event occurs when the user is pressing a key down**
onkeypress	**The event occurs the moment the user starts pressing a key**
onkeyup	**The event occurs when the user releases a key**
onload	**The event occurs when the HTML element (e.g. body) has been loaded**

onmouseover	The event occurs when the pointer is moved onto an element
onmouseout	The event occurs when the user moves the mouse pointer out of an element
onselect	The event occurs when the user selects some text (for <input> and <textarea>)
onsubmit	The event occurs when the form is submitted

JSON

JSON.parse(text)	Convert JSON into JavaScript Object
JSON.stringify()	Convert JavaScript object into JSON

AJAX – code template

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange = function() { // callback aka anonymous function
      if (this.readyState == 4) {
      if (this.status == 200) {
            // process response
            console.log( this.responseText );
        }
     }
  };



xhr.open("POST", "getData.php", true);
xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhr.send("x=1&y=2"); // query parameters



// xhr.open("GET", "getData.php?a=4&b=5", true);
// xhr.send();
```