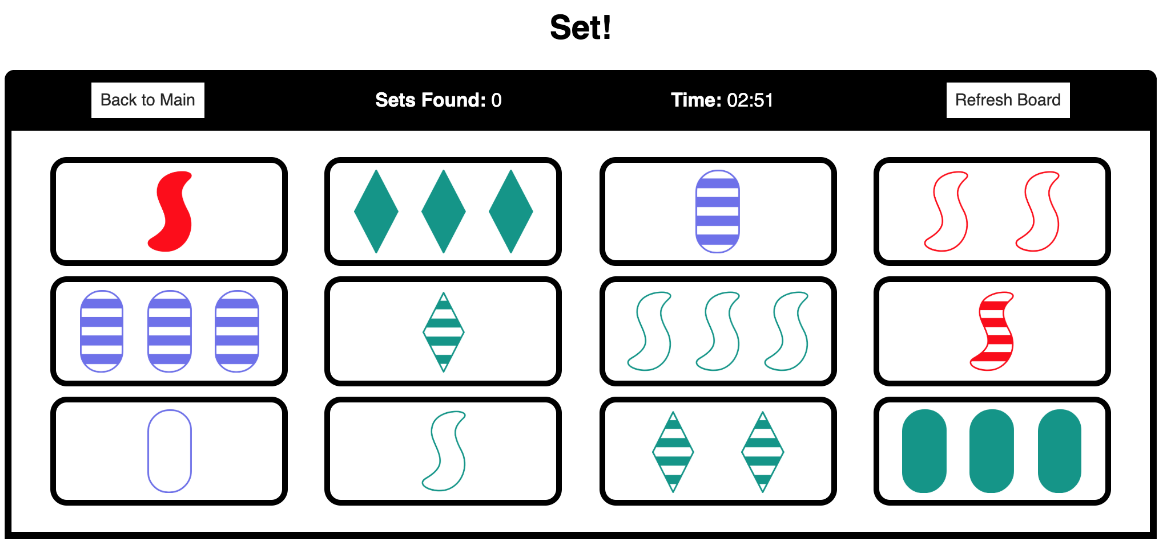
Homework 2: Set! Part A

Due: Saturday, April 24th 2AM PST (morning)



This assignment extends what we've learned in Module 1 (HTML/CSS) and puts into practice what you're learning in Module 2 (JavaScript, DOM, and Animations) to build an interactive webpage! A fully-functional UI (user interface) takes more time to plan and implement than just an HTML/CSS webpage (such as the one you created in HW1). As such, this HW is broken into two parts, based on the course schedule and the general process web developers take when implementing an interactive webpage like this one. Since this assignment primarily focuses on Module 2 material, we've provided all of the HTML for you as well as some CSS to get you started.

As a two-part assignment, it will be worth more points than HW1 and consists of two separate submissions:

1. Part A: Game View (CSS) and View-Switching (Start JS) - 10 points
2. Part B: Implementing the Game UI (JS DOM Manipulation and Timers) - 20 points

Note the point breakdown. You will have more time to work on Part B (do not treat Part A as "50%" of the work - Part B will take more time). The requirements for Part A are shorter, and will be due on Thursday so you can get feedback to implement in Part B.

Set! Overview

In this assignment you will use JS to add functionality to a basic webpage and use responsive CSS to layout two different views for the webpage. Specifically, you will implement a web-based version of the Set card game, providing features to generate new games for a user and to keep track of how many correct Sets a player has found. You will also implement a basic timer for the game, which increases/decreases depending on options a user selects.

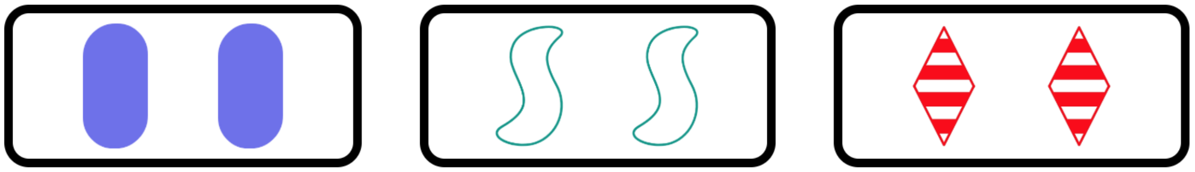
Rules of Set

While you will not implement the game logic until Part B, it can be helpful to have a bit of context of the finished product you will be creating.

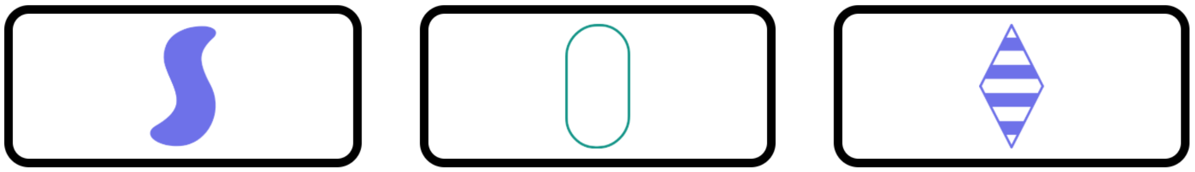
This assignment is inspired by the classic SET! Card game. A game consists of a board of cards (you will implement boards having 9 and 12 cards, and milestone.html is provided to test your styles.css with 9 cards). Each card has one of three options for 4 different "attributes":

Attribute	**Options**		
STYLE	solid	outline	striped
COLOR	green	purple	red
SHAPE	diamond	oval	squiggle
COUNT	1	2	3

The goal of the game is to find as many "Sets" of 3 cards such that for each attribute, all cards share the attribute or no cards share the attribute. For example, the following three cards build a Set because none share style, color, or shape attributes but they all share the count attribute.



However, the following three cards do not form a Set since the color attribute does not follow the "all or none" requirement (purple is shared by the first and third card, but not the second).



In Part B, you will be dynamically generating the board with JavaScript and use event listeners to support user interaction. You will also learn how to implement an increasing/decreasing game timer on this page, as well as how to show/hide feedback to users with a 1 second delay.

Starter Files and Final Deliverables

File/folders	Provided files to stay unchanged
`milestone.html`	A starter file for testing CSS styles related to the board and 9 cards (simulating a game for "Easy" difficulty). This **file will only be used to test your styles.css in Part A**. You will finish set.js to dynamically generate the cards in set.html in Part B.
`imgs`	A folder with 27 classic set cards, each named with the convention: *STYLE-SHAPE-COLOR.png* (replacing STYLE, SHAPE, COLOR with a value for that attribute as listed in "Rules of Set" section)
`styles.css`	Stylesheet for milestone.html (**required to finish for Part A and Part B**)

For Part A, your submission should be submitted with these starter files as well as the following **new** file you are to implement:

File	Files you will implement
`set.js`	For the Part A submission, this will implement functionality to toggle between `#menu-view` and `#game-view`. For Part B, you will finish the game UI implementation.

To summarize: **Your final solution for Part A will be graded only on styles.css and set.js** - any changes you make to the provided HTML or imgs folder will not be eligible for full credit. styles.css and set.js will also be completed and submitted in Part B (the final HW2 submission).

Part A Requirements (Game View with CSS and View-Switching with JS)

Part A will be due before Part B to give you an opportunity to get early feedback on your work, particularly relating to CSS. Although you are welcome and encouraged to develop more than is required for this Part A submission, any additional work related to Part B will not be graded until your submission for Part B.

The CSS for the `#menu-view` has been implemented for you in the styles.css file. You are responsible for ensuring the `#game-view` is stylistically complete.

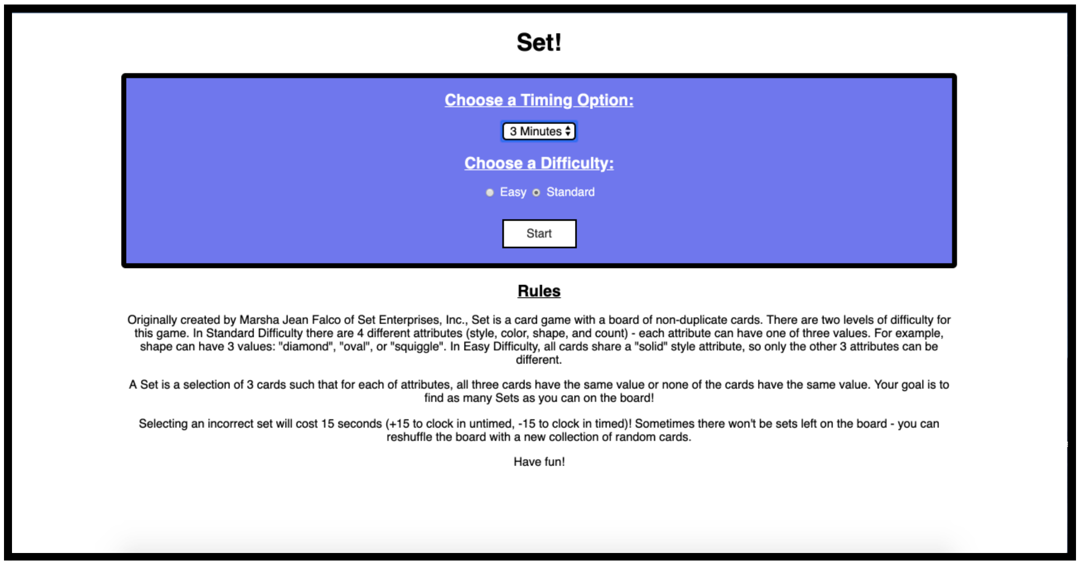
Note: button and select elements require selectwors that specifically include those tags for properties like `font-family` and `font-size` which you will see in the provided styles.css.

Main Menu View vs. Game View

There are two views in the game, representing the "Main Menu" view (elements of the `#menu-view` section) and the "Game" view (elements of the `#game-view` section). When the page loads, only the Main Menu view should be visible. The CSS and JS you will write in Part A will allow a user to show/hide ("toggle") between the two views when one of two buttons are clicked.

- The provided HTML gives the `#game-view` a `.hidden` class. You are to implement the styling for this class in CSS and use it to hide/show elements in JS appropriately.
- After writing the CSS needed for the .hidden class, we recommend temporarily unhiding the `#game-view` by commenting out the .hidden ruleset in styles.css so that the `#game-view` is shown and you can focus on the CSS for the Game View. Once you have the CSS done for the Game View, we recommend focusing on the JS to toggle between the two views using the `.hidden` class.

Main Menu View (Provided)



Rules

Originally created by Marsha Jean Falco of Set Enterprises, Inc., Set is a card game with a board of non-duplicate cards. There are two levels of difficulty for this game. In Standard Difficulty there are 4 different attributes (style, color, shape, and count) - each attribute can have one of three values. For example, shape can have 3 values: "diamond", "oval", or "squiggle". In Easy Difficulty, all cards share a "solid" style attribute, so only the other 3 attributes can be different.

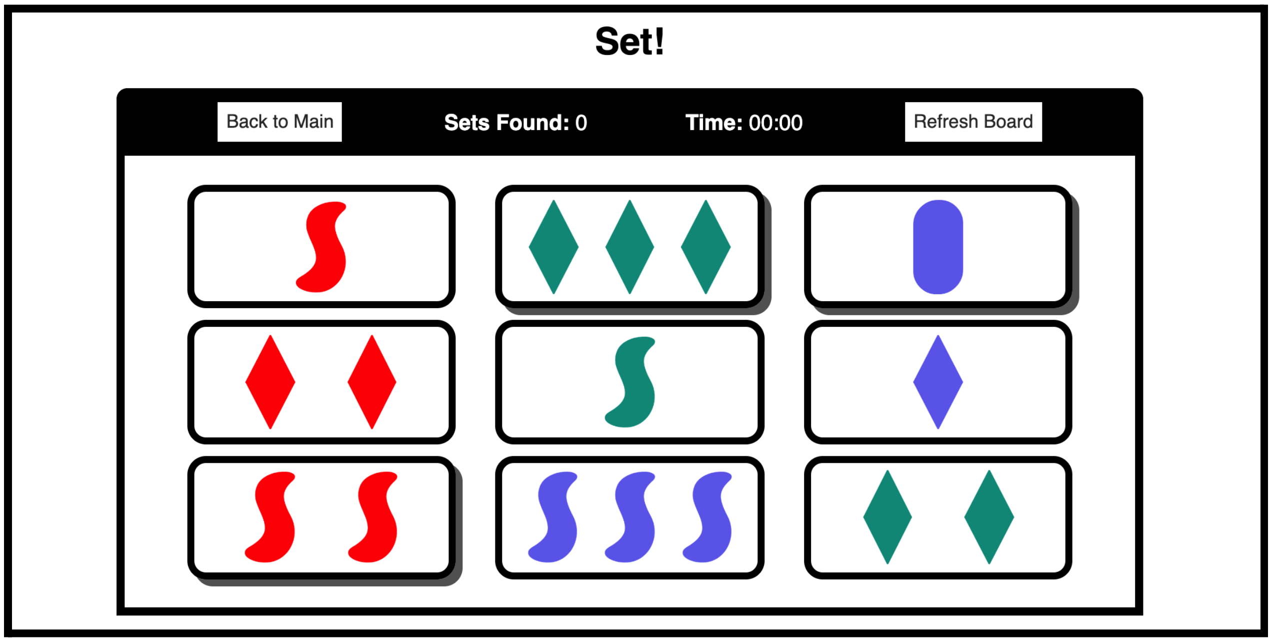
A Set is a selection of 3 cards such that for each of attributes, all three cards have the same value or none of the cards have the same value. Your goal is to find as many Sets as you can on the board!

Selecting an incorrect set will cost 15 seconds (+15 to clock in untimed, -15 to clock in timed)! Sometimes there won't be sets left on the board - you can reshuffle the board with a new collection of random cards.

Have fun!

- **Note**: The rendering of the `select` element is particularly different between browser/operating systems. Refer to the expected screenshots on your system, but if you are missing the "up/down" arrows don't worry about this.

Game View (To implement in styles.css)



In this part, you will use flex positioning where appropriate to layout the details bar, board and cards (do not use float or absolute positioning in this assignment). milestone.html "simulates" a randomly-generated board of 9 cards ("Easy" difficulty). Remember that in Standard difficulty, there are 12 cards on the board and cards may have the 4th STYLE attribute unique to Standard difficulty (refer to the screenshot at the start of this spec).

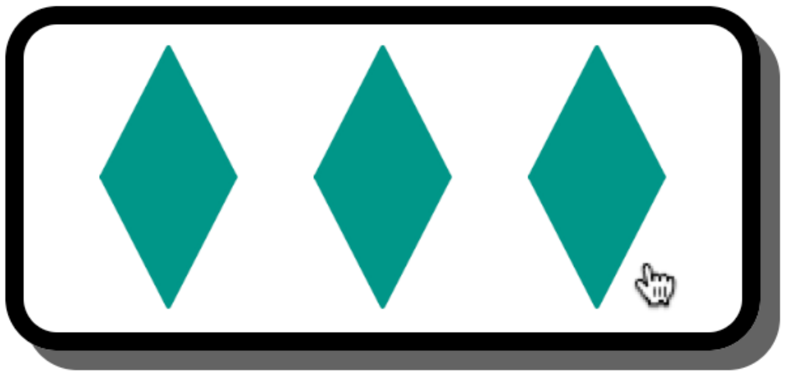
Details Bar

When the `#game-view` is visible, the user should see a "details" bar with a board of cards underneath. The `#details-bar` includes a "Back to Main" button, a set counter, a timer, and a "Refresh Board" button.

- The `background-color` of the `#details-bar` should be black.
- All four elements in this bar should have their horizontal distribution set to `space-evenly` and vertical distribution set to `center`.
- The top corners of the bar should be rounded with a border radius of 0.5em.
- Aside from the buttons which have black font, text in the details bar should be white.
- Any text in this `#details-bar`, besides the buttons that are already styled, should have a font size of 14pt.

Cards

- Each card should be a `div` element with the class `.card` that is 220px wide and 95px tall with a solid black border of 0.35em width and a border radius of 1em.
- Cards will have 1, 2, or 3 shape images (multiple images in the same card must be identical).
  - Each shape image should have a height of 85% relative to the card.
  - The images in the card should have their horizontal distribution set to `space-evenly` and vertical distribution set to `center`.
- When a user hovers over a card, the mouse cursor should change to a pointer icon. Use the `cursor: pointer;` rule on the cards to accomplish this.
- When a card has a `.selected` class during a game, it should have a #636363 "box shadow" shifted 6px right and 6px down. More details about when a card is "selected" will be relevant only to Part B, but three cards have this class to test your CSS in milestone.html. The card below is an example selected card appearance.
- Each card should have 5px of top and bottom margin.



Board

- The `#board` has a black border on the left, right, and bottom with a 5pt width.
- The cards of the `#board` element should be horizontally distributed using `space-evenly`.
- The cards should wrap naturally in the board as screen sizes adjust (hint: with appropriate flex properties, you should not need to define any rows or columns).
- There should be 20px of padding on all sides of the `#board` (this spacing will be fixed on the top/bottom but there may be extra spacing visible on the left/right depending on how cards fit for a particular screen width).
- Your layout styles in Part A should still meet the requirements listed here for a board in Standard Mode (12 cards) if using flex appropriately.
- **Note**: If the screen gets wide enough, the last row may have fewer cards than the earlier rows (you do not need to ensure all rows have the same number of cards in this case).

Step 2: Starting JS - Implement `toggleView()` Function

For both Part A and Part B you will be implementing functions in set.js to complete the game UI. We have required functions for you to implement (with clear specifications) to help you break down a JavaScript program into functions. You will be expected to put these functions together appropriately to build a working UI for your final submissions. For Part A, you will just be implementing one function: `toggleView()` (described below).

- Implement and use this function to allow a user to switch between the `#menu-view` and the `#game-view` when clicking the `#start-btn` and `#back-btn` buttons.
- For Part A you **do not** need to account for the difficulty or the timing option selected by the user when switching from `#menu-view` to `#game-view`.

`toggleView()`

- When this function is called, the `.hidden` class should be toggled for `#menu-view` and `#game-view` (note that there should always be exactly one view element having this class, starting with `#game-view`).

Debugging Tools

We strongly recommend that you use the Chrome DevTools on this assignment, or use the similar tool built into other browsers. This tool shows syntax errors in your JavaScript code. As we've demonstrated in class, you can use it as a debugger, set breakpoints, type expressions on the Console, and watch variables' values. This is an essential tool for efficient JavaScript programming.

The ES6 JSLint tool can help you find common JavaScript bugs. Since this is your first JavaScript assignment, you will probably encounter tricky bugs. If so, paste your code into JSLint to look for possible errors or warnings. To be eligible for full credit, your JavaScript code must pass the provided JSLint tool with no errors reported ("Warnings" are okay.)

External Requirements

For full credit, Your webpage should match the overall appearance of the provided screenshots and it **must** match the appearance specified in this document. We do not expect you to produce a pixel-perfect page that exactly matches the expected output image. However, your page should follow the specific style guidelines specified in this document and match the look, layout, and behavior shown here as closely as possible. Any properties unspecified in this spec or visually discernible in screenshots should be left to the default values for the respective page element (e.g the font size of the `h1` element on the page).

Internal Requirements

For full credit, your page must not only match the External Requirements listed above, you must also demonstrate that you understand what it means to write code following a set of programming standards. Your CSS and JS should also maintain good code quality by following the CS 101 Code Quality Guide. Make sure to review the section specific to JavaScript! We also expect you to implement relevant feedback from previous assignments. Some guidelines that are particularly important to remember for this assignment:

CSS:

- Use selectors appropriately to avoid needing to apply classes and ids to large numbers of HTML elements, and group selectors to reduce redundancy between shared styles.
- Express all stylistic information on the page using CSS defined in styles.css - do not set styles in JS (use `classList` instead).

JavaScript:

- Your .js file must be in the module pattern and run in strict mode by putting `"use strict";` within the module pattern.
- Do not use any global variables, and minimize the use of module-global variables. Do not ever store DOM element objects, such as those returned by the `document.querySelector[All]` functions, as global variables.
- Use camelCase naming conventions for variables and functions

Both:

- Format your CSS, and JS to be as readable as possible, similarly to the examples from class: Properly use whitespace and indent your CSS and JS code as shown in class. You can find information about JS conventions in the Code Quality Guide.
- You may not use any CSS/JS frameworks for this assignment.
- Your page must use valid CSS3 and JS and successfully pass both the W3C CSS3 validator and JSLint with no errors.

Documentation

Place a comment header **in each file** with your name, section, a brief description of the assignment, and the file's contents (examples have been given on previous assignments).

- You will be expected to properly document your functions in set.js using JSDoc with `@param` and `@return` where necessary. Refer to the Code Quality Guide for some examples.

Debugging Tools

We strongly recommend that you use the Chrome Inspector on this assignment. This will help you see the rendered styles of each element! You will also want to become familiar with it as early as possible, as it will become particularly useful for debugging JavaScript in future assignments!

Grading

Your Part A submission will be worth 10 points and due Saturday at 2AM. We strongly recommend you **do not** use a rework on this assignment in order to get feedback in time for CP2 and Part B of HW2.

The grading distribution for this assignment will be broken down as follows:

- External Correctness: 6pts
- Internal Correctness: 3pts
- Documentation: 1pt