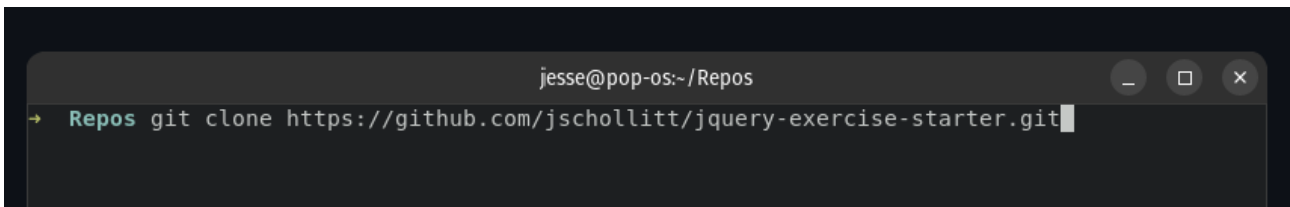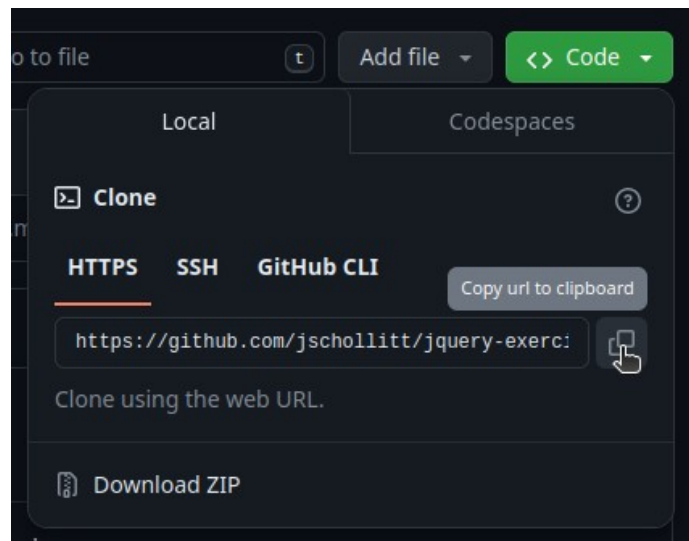# ISCG6420 Semester 1 2024

Exercise: jQuery fundamentals

These instructions will take you through the process setting up using jQuery in a few use-cases.

Open Exercise Starter

Clone the content from the starter project repository:

Open the project folder in VSCode, and open the week8.html file for editing.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Week 8</title>

    <!-- Link external CSS File   -->

    <!-- Add jQuery (from CDN) Script -->
```

Link the provided **week8.css** file as an external CSS file, and add jQuery as an external script using the copyable snippet from https://releases.jquery.com

jQuery Core

Showing the latest stable release in each major branch. See all versions of jQuery Core.

jQuery 3.x

- jQuery Core 3.7.1: uncompressed, minified, slim, slim minified

---

Code Integration   ✕

```
<script
  src="https://code.jquery.com/jquery-3.7.1.min.js"
  integrity="sha256-/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo="
  crossorigin="anonymous"></script>
```

Copy to clipboard!

The integrity and crossorigin attributes are used for Subresource Integrity (SRI) checking. This allows

---

```html
<title>Week 8</title>

<link rel="stylesheet" href="./week8.css" />

<script src="https://code.jquery.com/jquery-3.7.1.min.js"
    integrity="sha256-/JqT3SQfawRcv/BIHPThkBvs0OEvtFFmqPF/lYI/Cxo=" crossorigin="anonymous"></script>
```

Save the file.

Document Ready

Open the provided week8.js file for editing. Inside you will find the Document Ready function declaration. This function is run once the web page has completed loading synchronous content and is ready to begin running code. Inside are commented sections for each of the exercises in this activity:

```javascript
$('document').ready(function () {
    // exercise 1: jQuery Selectors


    // exercise 2: Selector filtering


    // exercise 3: Click events
```

Selectors
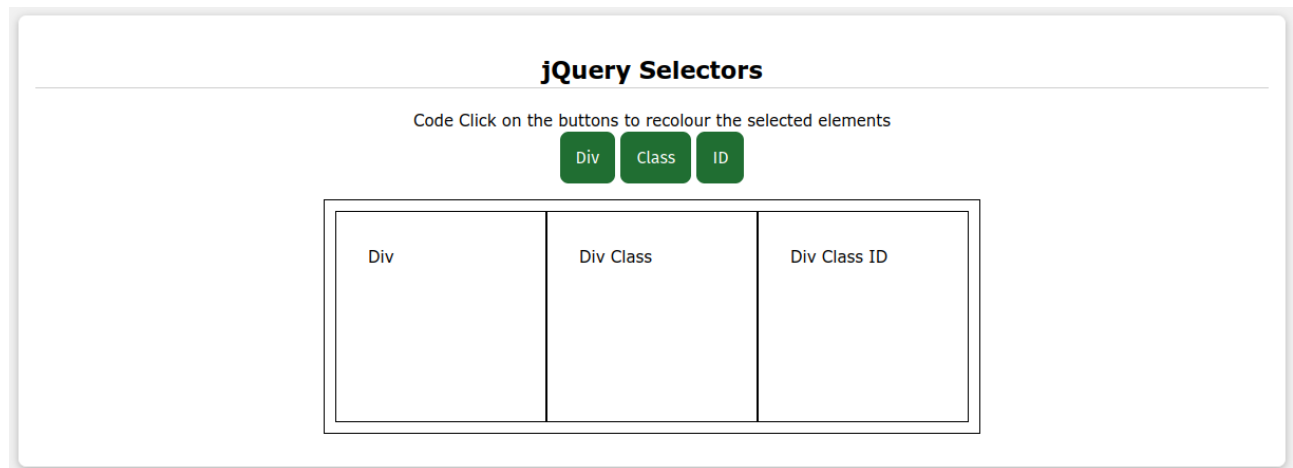
Exercise 1 demonstrates using three different jQuery selectors to target elements and change the background colour. Each button runs a function that selects the relevent elements are updates their CSS styles.

These are the button IDs:
- Div: "**ex1DivButton**"
- Class: "**ex1ClassButton**"
- ID: "**ex1IDButton**"

These are the element selector values:
- Div: All **<div>** tags that are children of ID "**demo1**".
- Class: "**ex1DemoClass**"
- ID: "**ex1DemoID**"

For each button, create a function for the click event. Inside the functions create a jQuery selector for the relevent element(s) and set the CSS **Background-color** to a colour:

```javascript
// exercise 1: jQuery Selectors
$('#ex1DivButton').click(function () {
    $('#demo1 div').css("background-color", "red");
});

$('#demo1ColourClass').click(function () {
    $('.ex1DemoClass').css("background-color", "green");
});

$('#demo1ColourID').click(function () {
    $('#ex1DemoID').css("background-color", "yellow");
});
```

Save your work and test the updates. The div button should set all boxes to red, the class button should set the two class boxes to green, and the ID button should set the ID box to yellow:
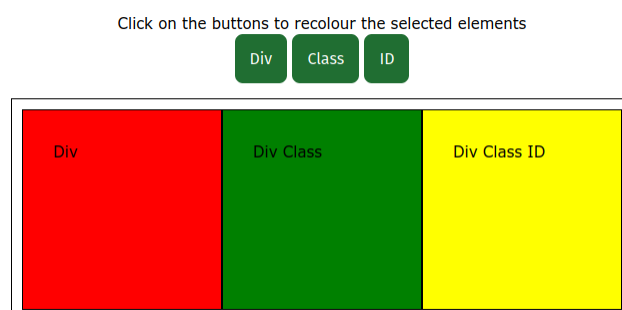
Next, we will create a function to fetch a pseudo-random colour for the demonstrations. To get different colours from the function we will use Math.random() to get a number between 0 and 1, multiply it by 10 to get a number between 0 and 10, and round the number down to an integer with Math.floor().

Math.floor(Math.random() * 10)

Outside the $('document').ready() function create a new function called 'getRandomColour()'. Inside the function create a switch statement that uses the above random number code as the assessed value:

```
function getRandomColour() {
    switch (Math.floor(Math.random() * 10)) {...
    }
}
```

Inside the switch statement create cases for numbers 0 to 9 and a default case. In each case, return a different colour name:

```
function getRandomColour() {
    switch (Math.floor(Math.random() * 10)) {
        case 0:
            return 'red';
        case 1:
            return 'orange';
        case 2:
            return 'yellow';
        case 3:
            return 'LightGreen';
        case 4:
            return 'Green';
        case 5:
            return 'LightBlue';
        case 7:
            return 'Blue';
        case 8:
            return 'indigo';
        case 9:
            return 'violet';
        default:
            return 'black';

    }
}
```
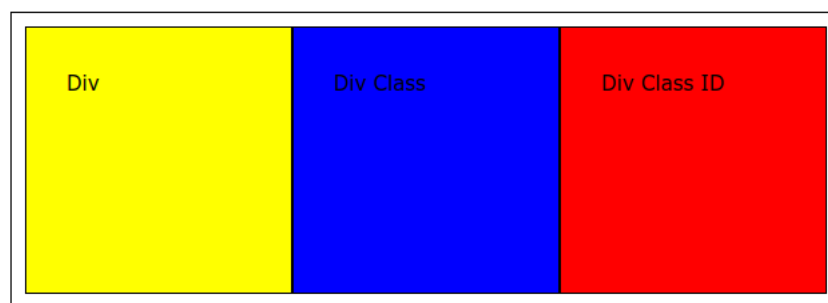
Update the exercise 1 button click functions to use this new function inplace of the static red, yellow, green values:

```javascript
// exercise 1: jQuery Selectors
$('#ex1DivButton').click(function () {
    let colour = getRandomColour();
    $('#demo1 div').css("background-color", colour);
});

$('#ex1ClassButton').click(function () {
    let colour = getRandomColour();
    $('.ex1DemoClass').css("background-color", colour);
});

$('#ex1IDButton').click(function () {
    let colour = getRandomColour();
    $('#ex1DemoID').css("background-color", colour);
});
```

Save your work and test the results. The buttons should now use a pseudo-random colour when updating the CSS of the boxes:

Selector Filter

Exercise 2 demonstrates filtering of selector results based on an attribute. When a radio button is selected, a function will run to filter the radio buttons for the selected value, and copy its value to the text line below.

In the JS file under the exercise 2 comment, create a selector for input of type radio with the attribute 'name' equal to 'radio-group'. Create a function for the 'change' event of this selector:

```
// exercise 2: Selector filtering
$('input:radio[name=radio-group]').change(function () {

});
```

Inside the function create a selector for the ID 'ex2Output' and set the html content of the selection to the value of the selected radio button.

Set the html content with: $('#ex2Output').html
Get the radio button value with: $('input:radio[name="radio-group"]:checked').val()

```
// exercise 2: Selector filtering
$('input:radio[name=radio-group]').change(function () {
    $('#ex2Output').html($('input:radio[name="radio-group"]:checked').val());
});
```
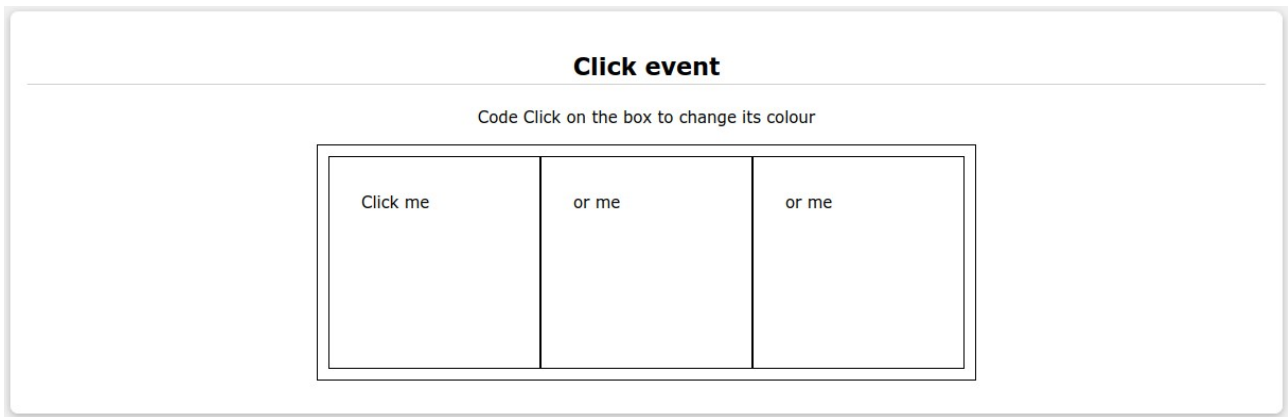
Save your work and test the output. The text line should update to complete with the text value of the selected radio button:

Change the radio button selection

○ Yes ○ No ● Maybe ○ I dunno

Selected option is maybe

Click Event

Exercise 3 demonstrates assigning a function to the click event of an element that isn't a button. jQuery API simplifies events of all elements by using the same naming scheme to access the events regardless of element type.

In the JS file under the exercise 3 comment, create a selector for class 'ex3Click' and assign a function to the click event:

```
// exercise 3: Click events
$('.ex3Click').click(function () {

});
```

Inside the function select the clicked element (using 'this' keyword) and set the **background-color** to a random colour from our function **getRandomColour()**:

```
// exercise 3: Click events
$('.ex3Click').click(function () {
    $(this).css("background-color", getRandomColour());
});
```

NOTE: Using 'this' keyword lets us refer only to the element that invoked the event. This is because at runtime the 'this' keyword will refer to the element that invoked the event function only. More information about using 'this' can be found at:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this

Be careful when using 'this' in combination with anonymous functions, as they do not get their own context. Instead the context is the parent of the anonymous function instead. More information about anonymous functions can be found at:
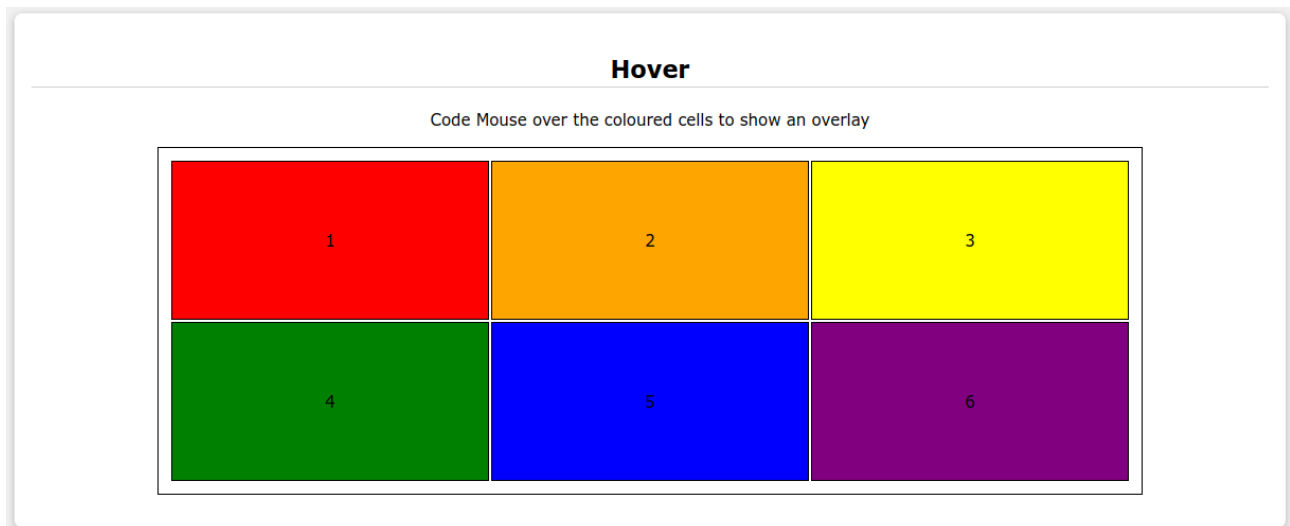https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Save your work and test the output. The three clickable elements should change colour when clicked without affecting the other elements:
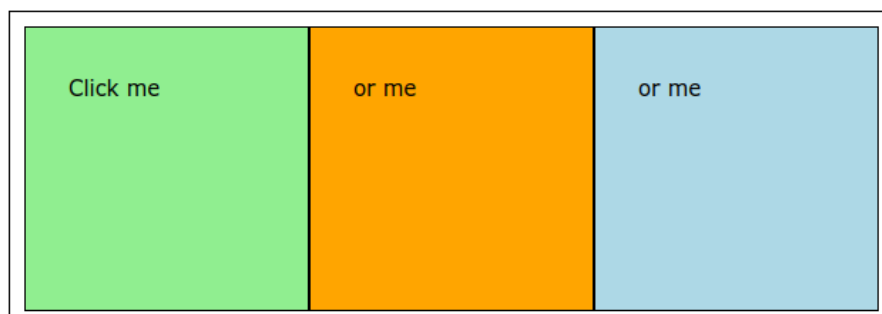
Hover Event

Exercise 4 demonstrates using hover event to respond to moving the mouse over elements.

In the JS file under the exercise 4 comment, create a selector for the ID 'ex4overlay' and hide the selected element using hide().

```
// exercise 4: Hover events
let overlay = $("#ex4overlay");
overlay.hide();
```

Select the class 'hover' and assign a function to the hover event and a separate function to the event callback:

```
overlay.hide();

$(".hover").hover(function () {
    // hover event
}, function () {
    // optional event complete callback
});
```

Inside the event function set the position of the overlay to the same position as the hovered element using the offset() function to get the position of the current element:

```
$(".hover").hover(function () {
    overlay.css($(this).offset());
```

Set the html content of the overlay element to the background colour (as a string) of the hovered element (this), and make the overlay visible by chaining show() at the end:

```
$(".hover").hover(function () {
    overlay.css($(this).offset());
    overlay.html($(this).css("background-color").toString()).show();
}, function () {
```

Finally, in the callback function hide the overlay:

```
$(".hover").hover(function () {
    overlay.css($(this).offset());
    overlay.html($(this).css("background-color").toString()).show();
}, function () {
    overlay.hide();
});
```

Save your work and test the output. When you mouse over the coloured cells the overlay should become visible, move over the cell, and display the RGB value as text:
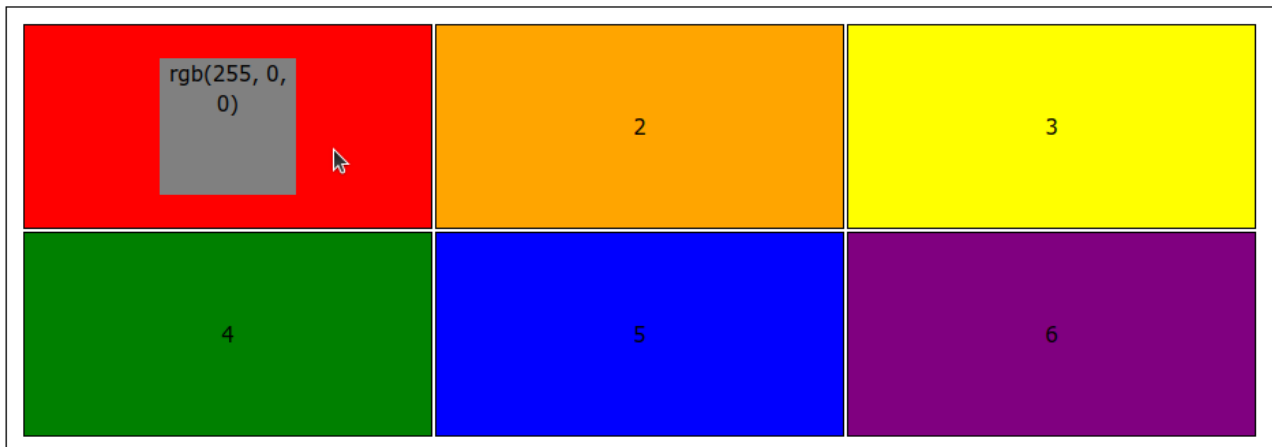
Slide

Exercise 8 demonstrates how to use **Slide** to expand and contract an element.

In the JS file under the exercise 8 comment, select the ID '**ex8**' and create a function for the **click** event.

```
// exercise 8: Slide
$('#ex8').click(function () {

});
```
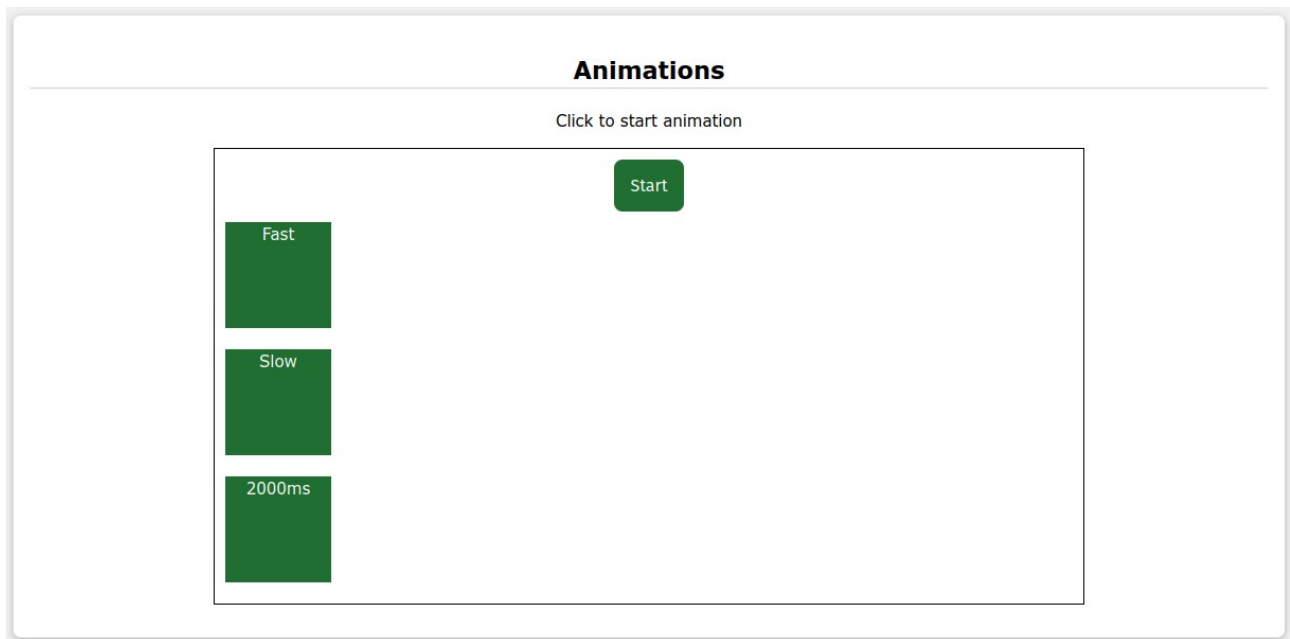
Inside the function select the ID '**lightbox**' and use **slideToggle**() with an optional parameter of 500 milliseconds:

```
// exercise 8: Slide
$('#ex8').click(function () {
    $('#lightbox').slideToggle(500);
});
```

Save your work and test the output. Clicking on the lightbox element should slide open the element to display a picture. Clicking on it while open should collapse the element:
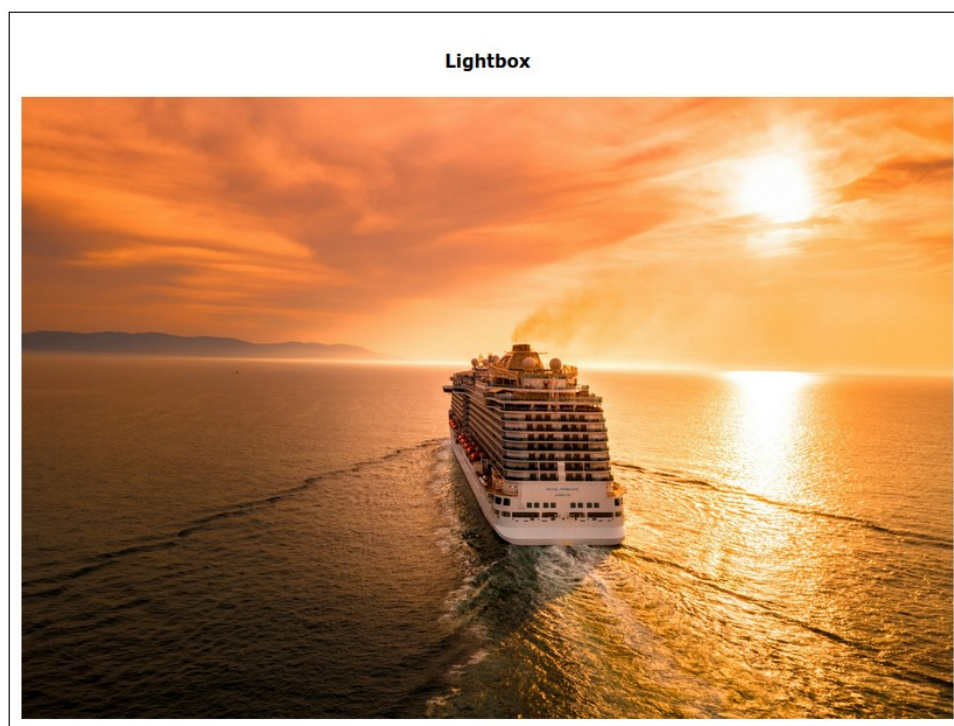
Animations

Exercise 9 demonstrates the animate function and the timing parameter.

In the JS file under the exercise 9 select the button child of ID 'ex9' and create a function for the click event:

```
// exercise 9: Animations
$('#ex9 button').click(function () {

})
```

Above the function create a boolean with the name 'ex9Animated' and set it to false. Inside the function create a variable with the name 'x' and set it equal to '0px' if ex9Animated is true, or '700px' if it's false. A ternary expression can consolidate this if-else statement:

- ex9Animated ? '0px' : '700px'

Ternary details are available here:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Conditional_operator

```
// exercise 9: Animations
let ex9Animated = false;
$('#ex9 button').click(function () {
    let x = ex9Animated ? '0px' : '700px';

})
```

Inside the function select each of the animating elements and use the .animate() function to set the 'left' CSS property to the value of 'x'. The animating elements are:
- ID: "ex91"
- ID: "ex92"
- ID: "ex93"

```
$('#ex9 button').click(function () {
    let x = ex9Animated ? '0px' : '700px';

    $('#ex91').animate({left: x});
    $('#ex92').animate({left: x});
    $('#ex93').animate({left: x});
})
```

Save your work and test the output. The animating elements should move from the left side of their container to the left:
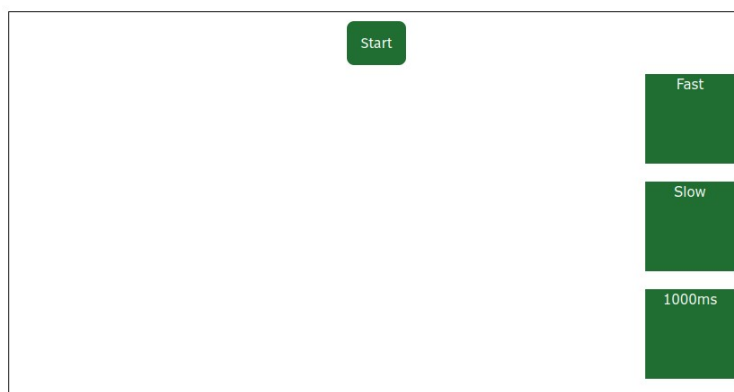
Inside the function after the animate lines, invert the value of 'ex9animate' by setting it equal to the inverse of its current value (! mean NOT, so NOT true is false, and NOT false is true):
- ex9Animate = !ex9Animate

```
    $('#ex91').animate({left: x});
    $('#ex92').animate({left: x});
    $('#ex93').animate({left: x});

    ex9Animated = !ex9Animated;
})
```

Add a value for the optional timing parameter of the animate() function for each element. Set them to the following values:
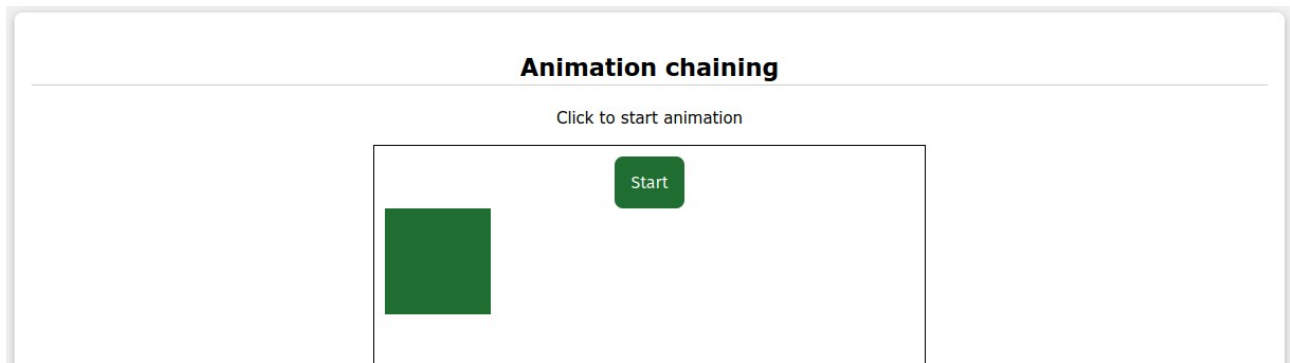- "fast"
- "slow"
- 1000

```
    $('#ex91').animate({left: x}, "fast");
    $('#ex92').animate({left: x}, "slow");
    $('#ex93').animate({left: x}, 1000);
```

Save your work and test the output. Each element should move at a different speed and alternate direction of animation.
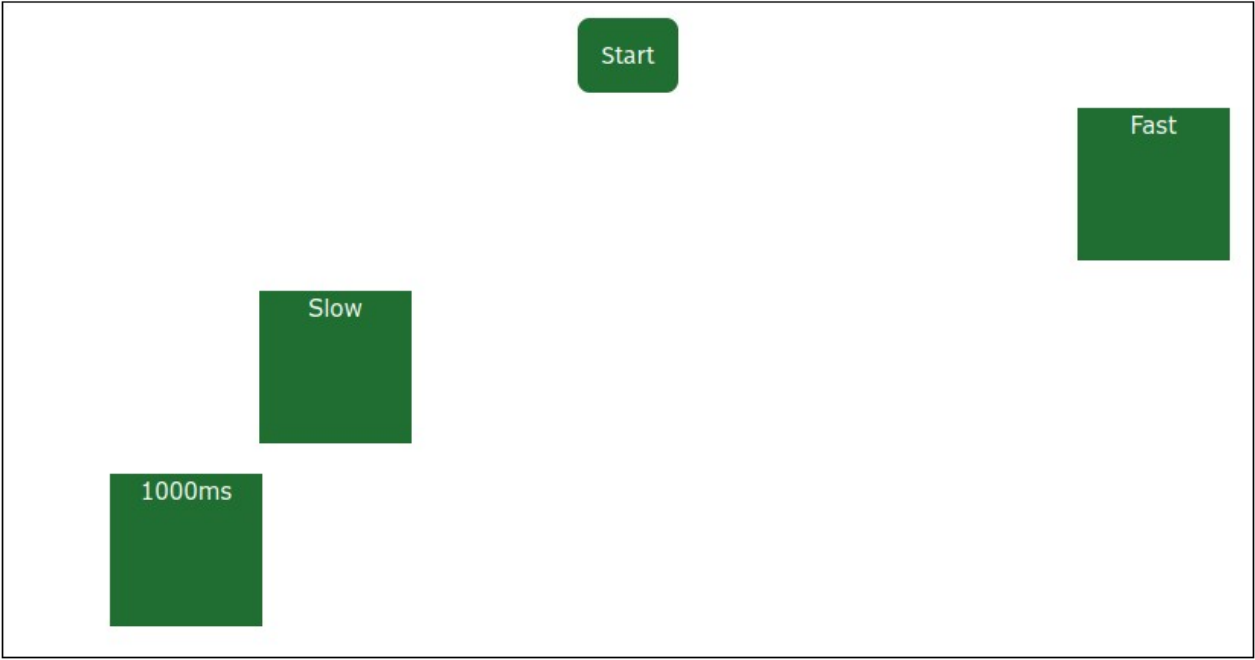
Animation Chaining

Exercise 10 demonstrates the practice of function chaining to append multiple animations into a single call.

In the JS file under the exercise 10 comment, select the button child of ID 'ex10' and create a function for the **click** event:

```
// exercise 10: Animation chaining
$('#ex10 button').click(function () {

});
```

Inside the function select the ID 'ex10animate' and use the **.animate()** function to animate the CSS properties { **height: '500px', opacity: '0.5'**} with a speed of '**slow**':

```
// exercise 10: Animation chaining
$('#ex10 button').click(function () {
    $('#ex10animate').animate({ height: '500px', opacity: '0.5' }, "slow");
});
```

Save your work and test the output. When the start button is clicked the box should increase in height and decrease in opacity during a short animation:
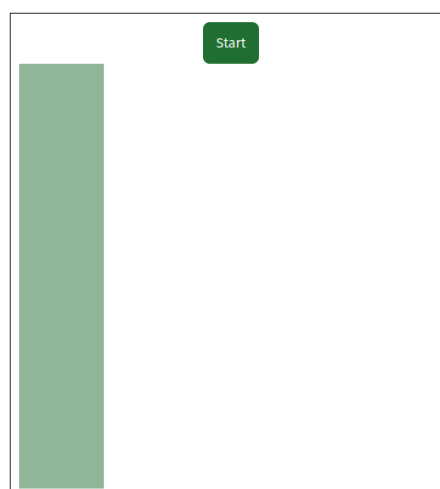
Back in the code, remove the semicolon from the end of the animate line and chain three additional animate function calls. Set the animate parameter values to:
{ height: '500px', width: '500px', opacity: '1' }, "slow"
{ height: '100px', opacity: '0.5' }, "slow"
{ width: '100px', opacity: '1' }, "slow"

To make the line more readable, move each chained function call to a new line and tab indent by one more than the first line:

```
// exercise 10: Animation chaining
$('#ex10 button').click(function () {
    $('#ex10animate').animate({ height: '500px', opacity: '0.5' }, "slow")
        .animate({ height: '500px', width: '500px', opacity: '1' }, "slow")
        .animate({ height: '100px', opacity: '0.5' }, "slow")
        .animate({ width: '100px', opacity: '1' }, "slow");
});
```

NOTE: Use **CTRL + SHIFT + I** to format your code using the document formatter. The document formatter can be set in the command pallette:
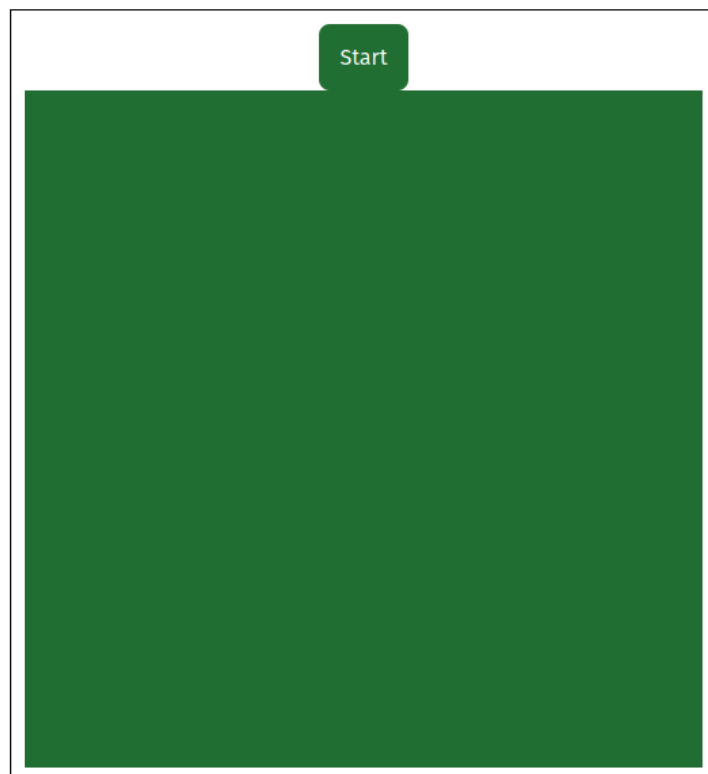**CTRL + SHIFT + P**
Search for "**Format Document With**"
Select "**Configure Default Formatter**"
Select "**prettier**" or your preferred formatter.

Save your work and test the output. The box should increase height, then increase width, then decrease height, then decrease width. Each animation should alternate increasing and decreasing opacity.

Element Creation

Exercise 11 demonstrates creating elements to store records of data captured from user input.

In the JS file under the exercise 11 comment, select the button child of ID 'ex11' and create a function for the click event:

```
// exercise 11: Element creation
$('#ex11 button').click(function () {

})
```

Inside the function create a chain of functions to do the following:
- select the table child of ID 'ex11'
- append a table row to the table
- append a table data to the table row
- set the text of the table data to the value of the input text field

```
// exercise 11: Element creation
$('#ex11 button').click(function () {
    $('#ex11 table').append($('<tr>').append($('<td>').text($('#ex11 input:text').val())));
})
```

This can also be separated for more clarity:

```
// exercise 11: Element creation
$('#ex11 button').click(function () {
    $('#ex11 table').append(                    // append table row
        $('<tr>').append(                       // append table data
            $('<td>').text(                     // set the text of td
                $('#ex11 input:text').val() // fetch input value from user
            )
        )
    );
})
```

Save your work and test the output. Data entered into the text input field should be added to the table as a new data record when the 'enter' button is clicked.

NOTE: If the table data is set using **.html()** instead of **.text()** the inputted text can be processed as page elements, as demonstrated here:

Next Steps

- Modify these exercises with different porperty and attribute values
- Explore the jQuery API and try implement some examples of more functionality
- Use the above exercises as a starting point for the group project assessment part 1.

Exercise complete.