Javascript 2

COSC3058 - Web Programming Bootcamp

Review Slides

Tom Huynh
School of Science, Engineering & Technology
RMIT University Vietnam

RMIT UNIVERSITY

Previously, we have learned:

- Javascript
- Run JS code directly on browser
- Attach JS to HTML document
- JS components
  - Logging and Alerting
  - Comments
  - Name Convention
  - Variables
  - Data Types
  - Operators
  - Control Structures
  - Functions
- JavaScript Style Guide

Agenda Today:

- Arrow Function
- Array
- Javascript Objects
- BOM (Browser Object Model)
- DOM (Document Object Model)
  - Selecting DOM elements
  - Manipulating DOM elements
- DOM Events

CES - Course Experience Survey

CES - Course Experience Survey

- **Please do it as soon as possible!**
- It is very important for you, me and RMIT as it is one of the best way to provide feedback for the course.
- **Please mention things you like so I will try to make it happen more often in the future courses or any of the course I will teach in the future semesters.**
- **Also, mention things you think it could be improved or changed so I will try my best to improve upon that!**
- I would like to **consider the survey is a way for us to build a long-term relationship as a lecturer and students** for many years to come!
- **Please go to this link: https://surveys.rmit.edu.au**

Arrow Function =>

Arrow Function

- **Arrow function** is a **shorthand**, **compact**, **convenient way** to define a regular function express.

```
function square (x){
    return x * x;
}
```

```
let square = (x) => {
    return x * x;
}
```

```
function sum (x, y){
    return x + y;
}
```

```
let sum = (x, y) => {
    return x + y;
}
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Round brackets of Arrow Function

- Round brackets are **optional** if there is only one parameter:

```
let square = x => {
    return x * x;
}
```

- Use empty round brackets for functions with **no parameters**:

```
let sayHello = () => {
    return "Hi there!";
}
```

Arrow Function Implicit Returns

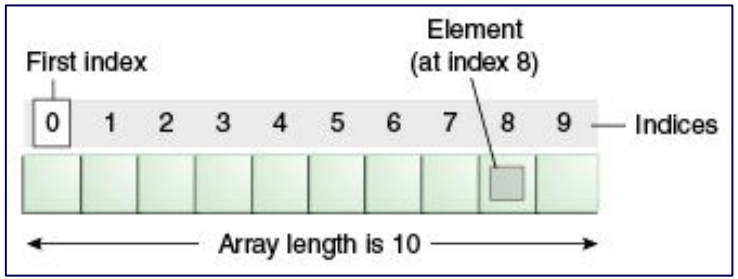
- **All of these functions are corrected and the same.**
- The code demonstrates different versions of a function that checks whether a number is even or not using regular function expression and arrow function syntax.
- The versions include different variations such as explicit/implicit return statements and usage of parentheses around the function parameter.

```
// Regular function expression
let isEvenVersion1 = function (number) {
    return number % 2 === 0;
}

// Arrow function with round brackets around parameters
let isEvenVersion2 = (number) => {
    return number % 2 === 0;
}

// No round brackets around parameters
let isEvenVersion3 = number => {
    return number % 2 === 0;
}

// Implicit return in one line
let isEvenVersion5 = number => number % 2 === 0;
```

Array 📚

Array

- An **array** is an **ordered collection** of **values**.
  - Examples:
    - List of comments on IG post.
    - Collection of levels in a game.
    - Songs in a playlist.
- You can treat an array of Javascript to be very similar to be ArrayList of Java or List of Python as their syntaxes and data structure are very similar.



https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Array Examples

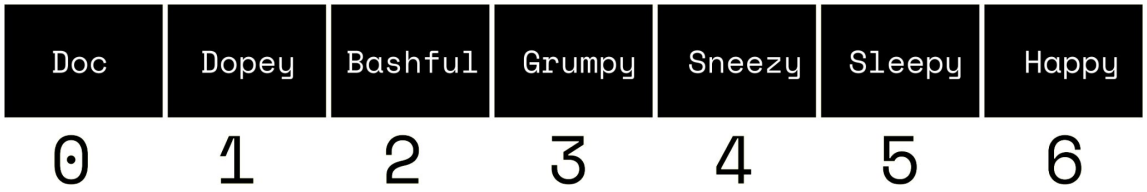
- Here are some examples how you can **initialize an array**.
- You can mix different value types in the same array.

```
// An array of strings
let colors = ['red', 'orange', 'yellow'];

// An array of numbers
let lotteryNumbers = [19, 22, 56, 12, 51];

// An mixed array with different value types
let stuffs = [true, 68, 'cat', null, 34.13, NaN, undefined];

let moreStuffs = [false, 42.3, [12, 43, 'yes'], function myFunction(a, b){ return a+b; }];
```

Array Indexes (Accessing)

- Each element of an array has a corresponding index.
- Counting of the index starts at 0.

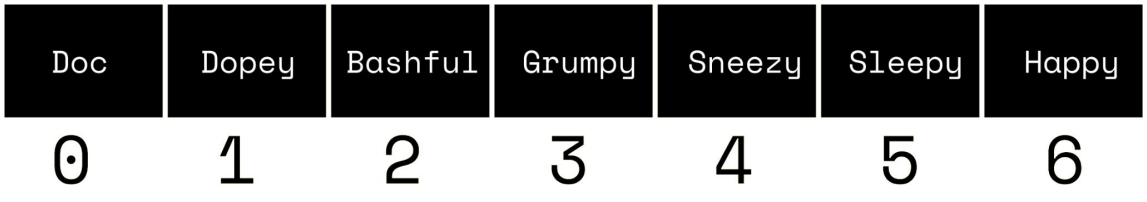
```
let colors = ['Doc', 'Dopey', 'Bashful', 'Grumpy', 'Sneezy', 'Sleepy', 'Happy'];

console.log(colors.length); // prints 7

// Accessing values of array elements
console.log(colors[0]); // prints 'Doc'
console.log(colors[2]); // prints 'Bashful'
console.log(colors[6]); // prints 'Happy'
console.log(colors[7]); // what will it print out?
```



Array Indexes (Modifying)

- An array elements can be modified via an index.

```
let colors = ['Doc', 'Dopey', 'Bashful', 'Grumpy', 'Sneezy', 'Sleepy', 'Happy'];

// Modifying values of array elements
colors[0] = 'Worry';
colors[3] = 'Sad';
console.log(colors[0]); // prints 'Worry'
console.log(colors[3]); // prints 'Sad'

colors[7] = 'Crazy';
console.log(colors[7]); // what will it print out?
```



Access and Modify Array Example

```
c = [-45, 6, 0, 72, 1543, -89, 0, 62, -3, 1, 6453, 78]
```

- An array element can be accessed via an index.

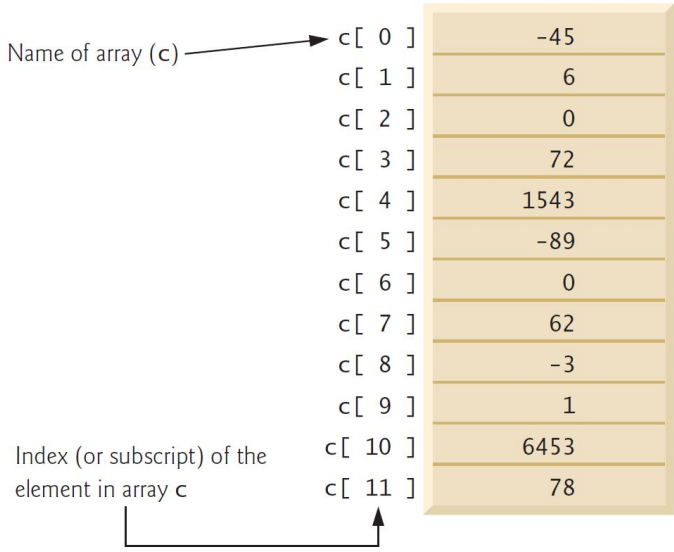
```
sum = c[0] + c[1] + c[2];
```

[Question] What is the output of sum?

- An array elements can be modified via an index.

```
a=5;
b=6;
c[a+b]+=2;
```

[Question] What will be changed in the array?



Push & Pop & Shift & Unshift

- **Push**: add to end.
- **Pop**: remove from end.
- **Unshift**: add to start.
- **Shift**: remove from start.

```
let movieLine = ['tom', 'nancy'];

movieLine.push('james'); // ['tom', 'nancy', 'james']
movieLine.push('charles'); // ['tom', 'nancy', 'james', 'charles']
movieLine.pop(); // returns 'charles' and remove him from moveLine
console.log(movieLine); // ['tom', 'nancy', 'james']

movieLine.unshift('john') // ['john, 'tom', 'nancy', 'james']
movieLine.unshift('adam') // ['adam', 'john, 'tom', 'nancy', 'james']
movieLine.shift(); // returns 'adam' and remove him from moveLine
console.log(movieLine); // ['john, 'tom', 'nancy', 'james']
```

More Array Methods

- **concat**: merge arrays.
- **includes**: look for a value.
- **indexOf**: just like string.indexOf.
- **join**: creates a string from an array.
- **reverse**: reverses an array.
- **slice**: copies a portion on an array.
- **splice**: removes or replaces elements.
- **sort**: sorts an array.

Slice

- **slice**: copies a portion on an array.
- slice(startIndex, endIndex)

```
let animals = ['ant', 'bison', 'camel', 'duck', 'elephant'];

console.log(animals.slice(2));     // ["camel", "duck", "elephant"]

console.log(animals.slice(2, 4));  // ["camel", "duck"]

console.log(animals.slice(-2));    // ["duck", "elephant"]

console.log(animals.slice(2, -1)); // ["camel", "duck"]
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/slice

Splice

- **splice**: removes or replaces elements.
- splice(startIndex, deleteCount, item1, item2, itemN)

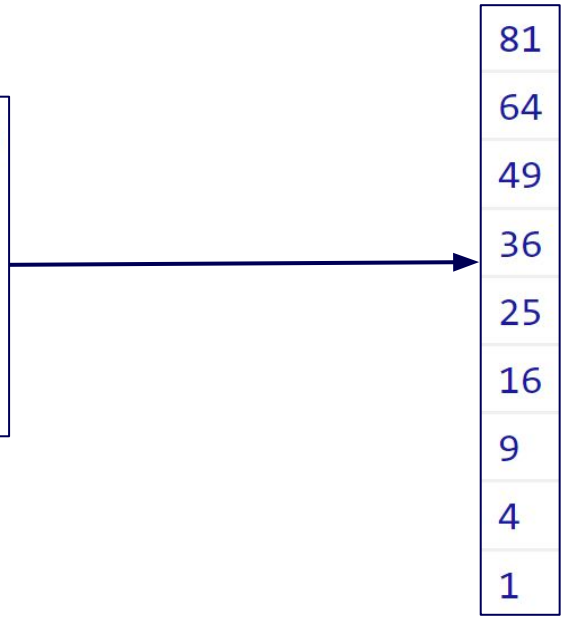
```
const months = ['Jan', 'March', 'April', 'June'];

months.splice(1, 0, 'Feb'); // Inserts at index 1
console.log(months); // ['Jan', 'Feb', 'March', 'April', 'June']

months.splice(4, 1, 'May'); // Replaces 1 element at index 4
console.log(months); // ['Jan', 'Feb', 'March', 'April', 'May']


months.splice(1, 3, 'ABC', 'DEF','GHK'); // Replaces 3 element at index 1
console.log(months); // ['Jan', 'ABC', 'DEF', 'GHK', 'May']
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/splice

forEach

- Accepts a callback function.
- Calls the function once per element in the array.

```
let nums = [9, 8, 7, 6, 5, 4, 3, 2, 1];

nums.forEach(n => {
    console.log(n*n);
});
```



https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/forEach

map

- Creates a new array with the results of calling a callback on every element in the array

```
let texts = ['tom', 'and', 'jerry', 'omg', 'lol'];
let caps = texts.map(t => t.toUpperCase());
console.log(caps);
```

```
['TOM', 'AND', 'JERRY', 'OMG', 'LOL']
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

find

- Returns the value of the first element in the array that satisfies the provided testing function.

```
let movies = [
    'The Fantastic Mr. Fox',
    'Mr. and Mrs. Smith',
    'Mrs. Doubtfire',
    'Mr. Deeds'
]

let movie = movies.find( movie => movie.includes('Mrs.'));
console.log(movie); // returns 'Mr. and Mrs. Smith'

let movie2 = movies.find( movie => movie.indexOf('Mrs.')===0);
console.log(movie2); // returns 'Mrs. Doubtfire'
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find

filter

- Creates a new array with all elements that pass the test implemented by the provided function.

```
let nums = [9, 8, 7, 6, 5, 4, 3, 2, 1];

let odds = nums.filter( n => n % 2 === 1);
console.log(odds); // returns [9, 7, 5, 3, 1]

let smallNums = nums.filter(n => n < 5);
console.log(smallNums); // returns [4, 3, 2, 1]
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

every

- Tests whether all elements in the array pass the provided function.
- It returns a Boolean value.

```
let words = ['dog', 'dig', 'log', 'bag', 'wag'];

words.every(word => word.length === 3);  // true

words.every(word => word[0] === 'd');  // false

words.every(w => {
    let last_letter = w[w.length - 1];
    return last_letter === 'g';
}) // true
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/every

some

- Similar to every, but returns true if ANY of the array elements pass the test function.

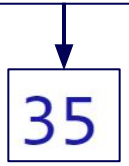
```
let words = ['dog', 'jello', 'log', 'cupcake', 'bag', 'wag'];

// Are there any words longer than 4 characters?
words.some(word => word.length > 4);  // true

// Does any word start with 'Z'?
words.some(word => word[0] === 'Z');  // false

// Does any word contain 'cake'?
words.some(w => w.includes('cake')); // true
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/some

reduce

- Executes a reducer function on each element of the array, resulting in a single value.
- Summing an array example:

```
[3, 5, 7, 9, 11].reduce((accumulator, currentValue) => accumulator + currentValue);
```



Callback	accumulator	currentValue	return value
first call	3	5	8
second call	8	7	15
third call	15	9	24
fourth call	24	11	35

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce

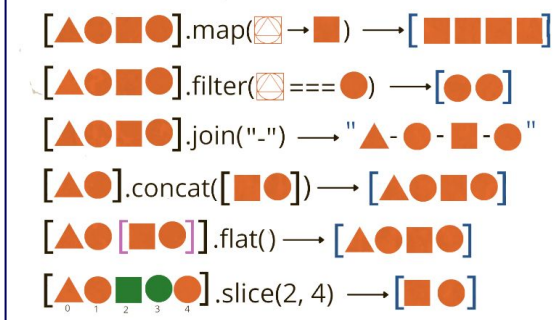
Finding max value using reduce example

```
let grades = [89, 96, 58, 77, 62, 93, 81, 99, 73];

let topScore = grades.reduce((max, currVal) => {
    if (currVal > max) return currVal;
    return max;
});
console.log(topScore); // returns 99

// A shorter version with Max.max & implicit return
let topScoreVer2 = grades.reduce((max, currVal) => (Math.max(max, currVal)));
console.log(topScoreVer2); // returns 99
```

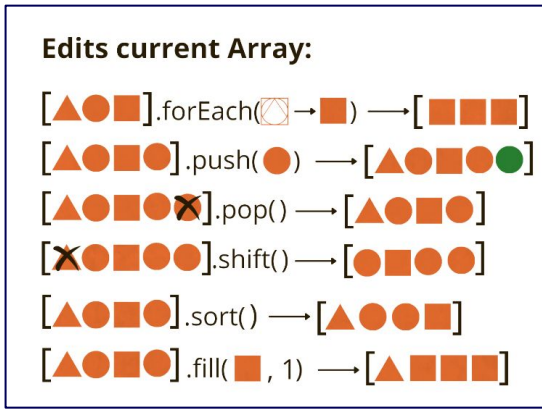
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/reduce

Array Methods Cheatsheet

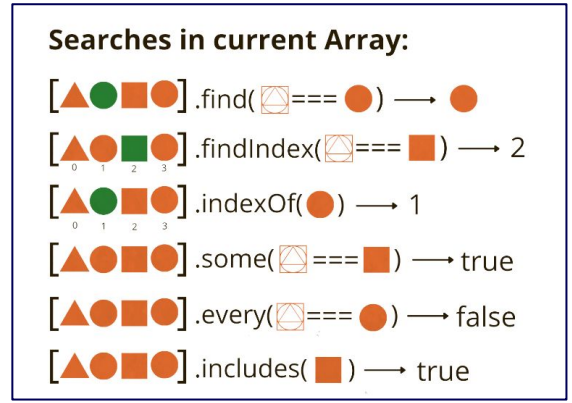
Creates new Array:

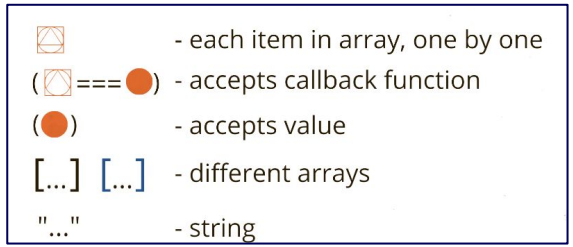


Edits current Array:



Searches in current Array:

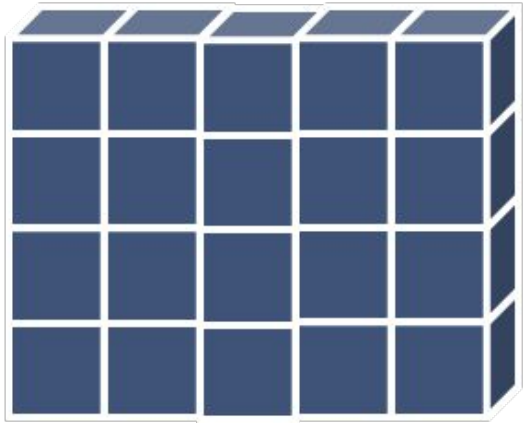




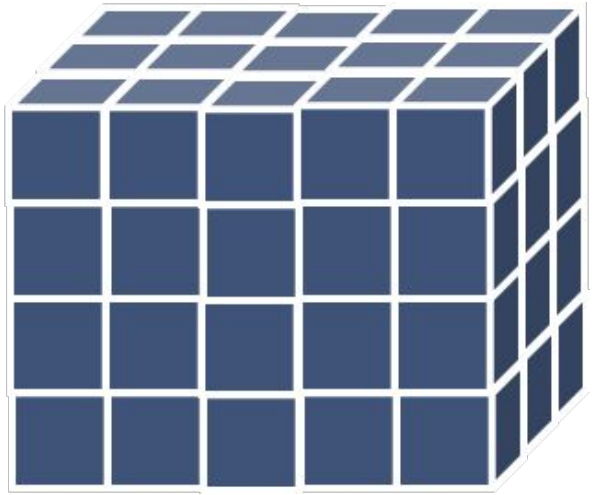
Multi-Dimensional Arrays

- Two dimensional (2D) array is one type of multidimensional array.

1-Dimensional Array



2-Dimensional Array

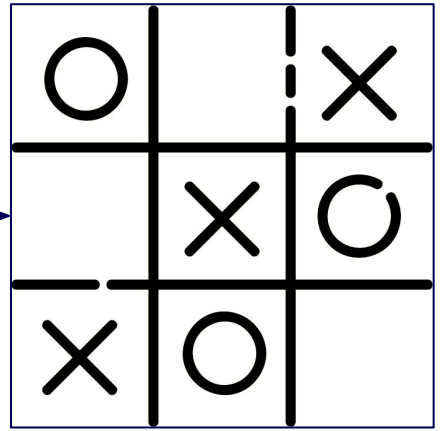


3-Dimensional Array

Nested Arrays

```
let colors = [
    ['red', 'yellow'],
    [1,2,3],
    [true, false, 'cat']
];
```

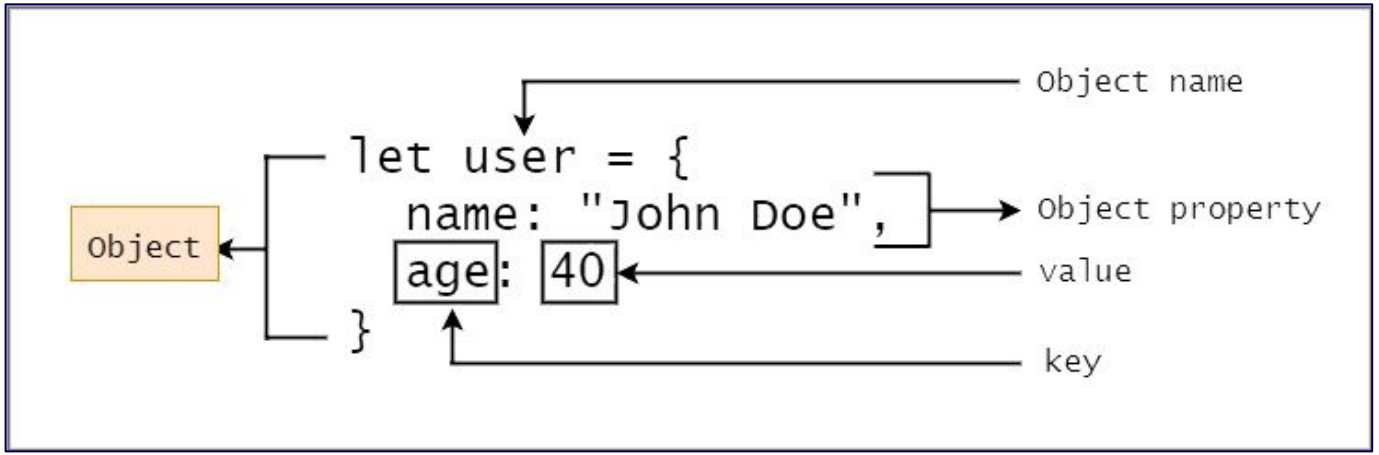
```
let checkerBoard = [
    ['O', null, 'X'],
    ['null', 'X', 'O'],
    ['X', 'O', null]
];
```



Javascript Objects

JS Objects

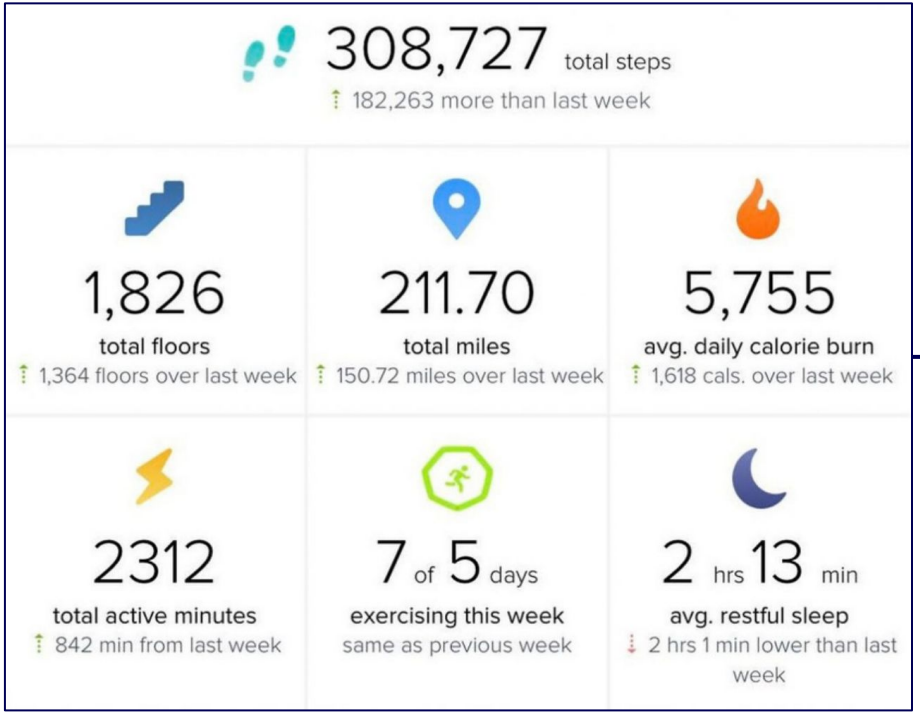
- **Objects** are collections of properties, which are **key-value** pairs.
- Instead of accessing data using an index, we use custom keys.
- The syntax and usage is **very similar to dictionary of Python**.



https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object

Create JS Object Example

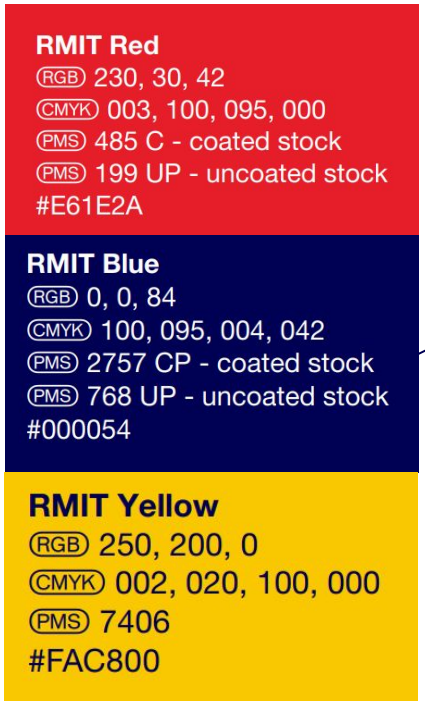
- An example to store actual data from fitBit to a js object:




```
let fitBitData = {
    totalSteps: 308727,
    totalMiles: 211.7,
    avgCalorieBurn: 5755,
    workoutsThisWeek: '5 of 7',
    avgGoodSleep: '2:13'
};
```

Access Object

- Access a value in a js object via its key.




```
let rmitColors = {
    red: '#E61E2A',
    yellow: '#FAC800',
    blue: '#000054'
}
```

```
// Two ways to access a value via a key
rmitColors['red']; // returns '#E61E2A'
rmitColors.red; // returns '#E61E2A'

// Not possible to use syntax . for a variable
let color = 'blue'
rmitColors[color]; // returns '#000054'
rmitColors.color // Not possible, returns undefined
```

Modify Object

- Modify a value in a js object via its key.

```
let studentGPAs = {
    tom: 3.8,
    jerry: 2.5,
    charles: 4.0
}

// Modify a gpa of tom
studentGPAs['tom']= 4.0;

// Can use either [] or . to modify the values
// like how we access the values
// Modify all gpa to US gpa system
studentGPAs.tom= 'A+';
studentGPAs['jerry']= 'B-';
studentGPAs.charles= 'C+';
```

Nesting Objects & Arrays

- You can have array and objects nested as needed.

```
let shoppingCart = [
    {
        product: 'chocolate',
        price: 6.5,
        quantity: 1
    },
    {
        product: 'chicken soup',
        price: 12.45,
        quantity: 2
    },
    {
        product: 'toilet paper',
        price: 2,
        quantity: 20
    }
];
```

```
let student = {
    firstName: 'Tom',
    lastName: 'Huynh',
    hobbies: ['Music', 'Video Games'],
    exams: {
        midterm: 93,
        final: 85
    }
};
```

Browser Object Model (BOM)

Browser Object Model (BOM)

- Many properties of the user's browser is available to a developer. They are grouped under the Window object:
  - **Document**: contains references to all elements in the page. We will look at this object in great detail shortly.
  - **History**: contains a list and count of the users visited urls.
  - **Navigator**: contains information on the browser, such as vendor, version, platform, geolocation etc.
  - **Location**: contains information about the server that is serving the current page such as name, protocol etc.
  - **Screen**: contains information about the user's screen such as size, orientation, color depth etc.

Accessing BOM

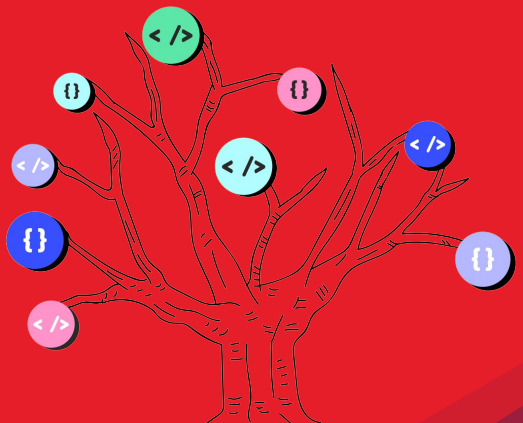
- BOM objects properties and methods are structured in a branching tree-like structure. That is, some methods and properties are located inside other properties. All can be accessed using standard object referencing notation or by treating the object as an associative array:

```
alert(screen.width);
alert(screen['width']);
alert(screen.orientation.angle);
alert(screen['orientation']['angle']);
```

- Many of the properties can be modified, as this fun website demonstrates: http://stewd.io/pong/

```
window.open();
...
window.moveTo( xPos, yPos );
...
window.close();
```

DOM (Document Object Model)



Document Object Model (DOM)

- The **Document Object Model (DOM)** is a programming interface for HTML documents.
  - It represents the web page as a hierarchical tree structure, where each node in the tree corresponds to an element, attribute, or piece of text in the document.
  - In other words, it is just a bunch of objects that you can interact via JS.
- The **Document object** is the root node of the DOM tree.
  - It is the starting point for accessing and manipulating the content of the web page through JavaScript.
- Every website has the document object, and to see it as a string, type in the console:

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

- To display document object in tree-like format format of Javascript objects then type in the console:

```
console.dir(document);
```

https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

Document Object Model (DOM)

- Some of the properties of the document object might be useful like document.documentElement, document.head, document.body, document.title, and so on.
- For example, the following returns a tree like structure of all the elements in the webpage **<body>** element:

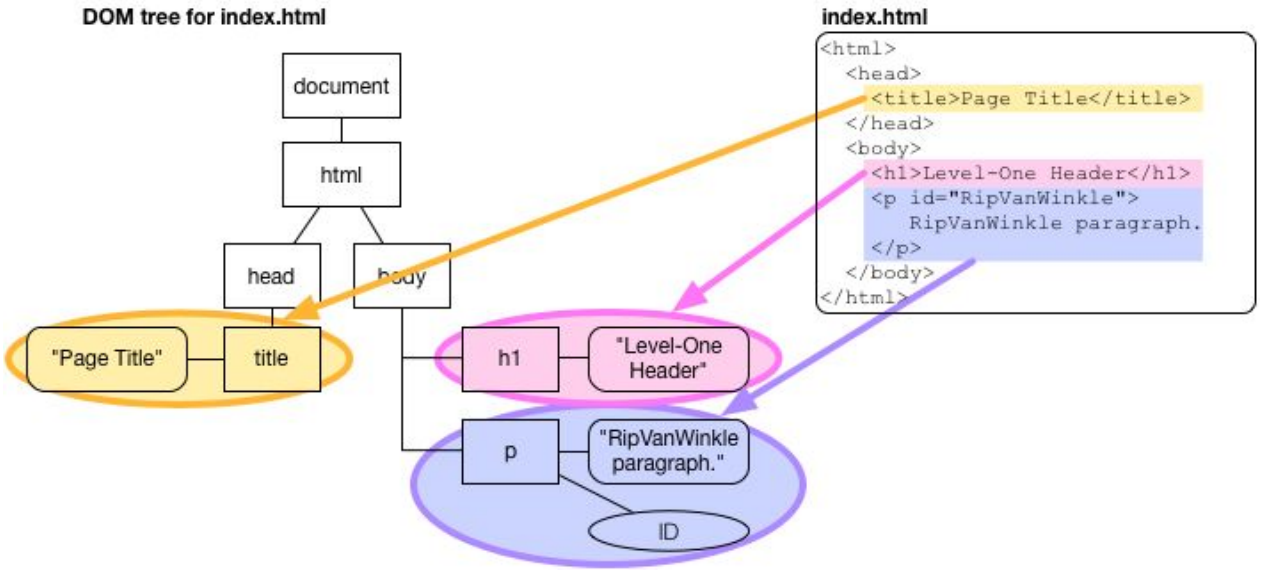
```
console.log(document.body);
```

- To access the 4th element in the <body> element, you need to look at the children of the body element:

```
console.log(document.body.children[3]);
```

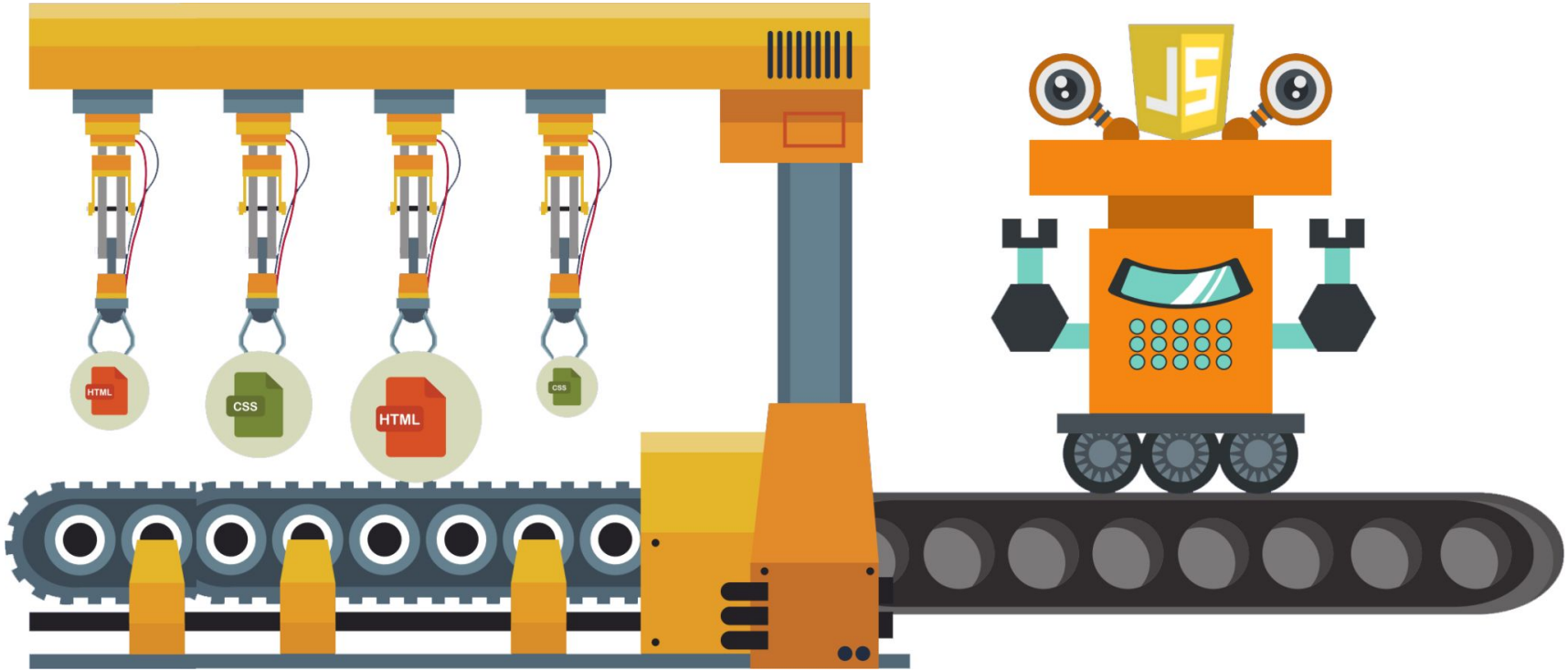
Chrome Extension

- HTML Tree Generator: This extension displays any web page as a DOM tree.



DOM Object Creation

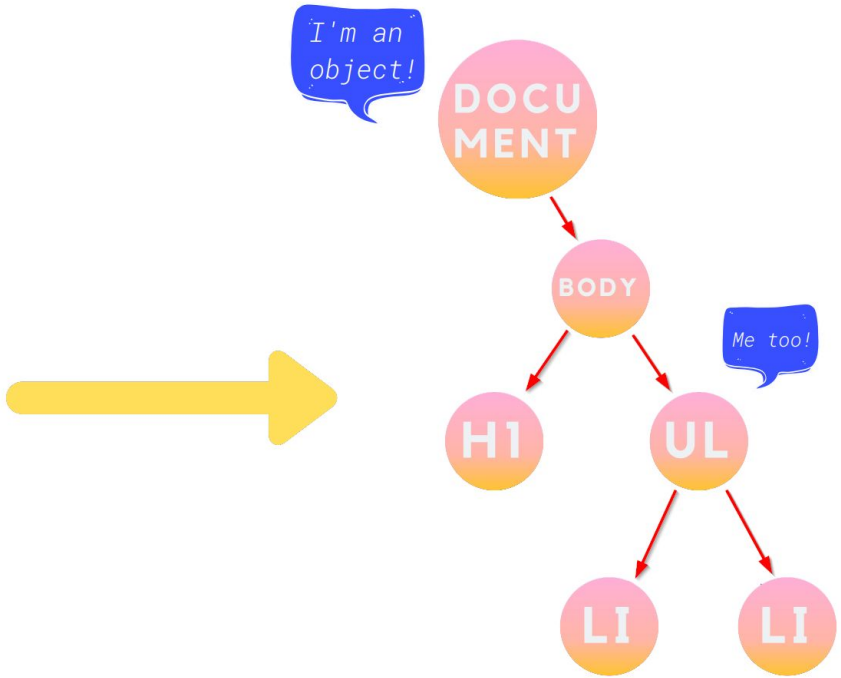
HTML + CSS go in → JS Object come out!



DOM Object Creation

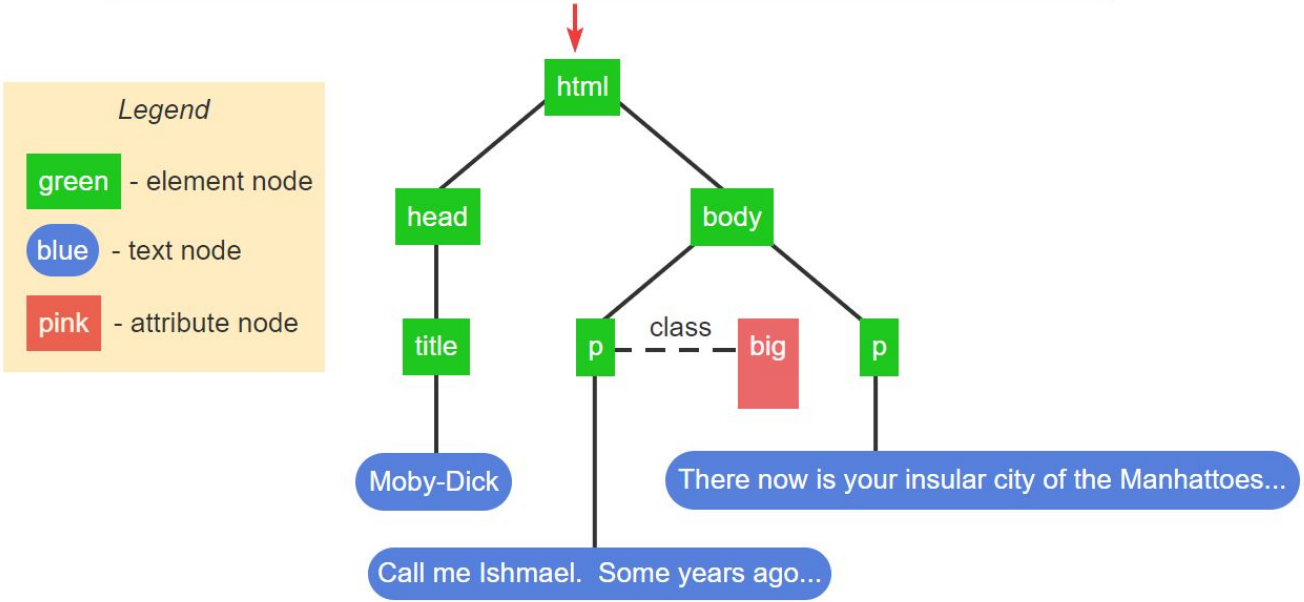
HTML + CSS go in → JS Object come out!

```
<body>
  <h1>Hello!</h1>
  <ul>
    <li>Water Plants</li>
    <li>Get Some Sleep</li>
  </ul>
</body>
```



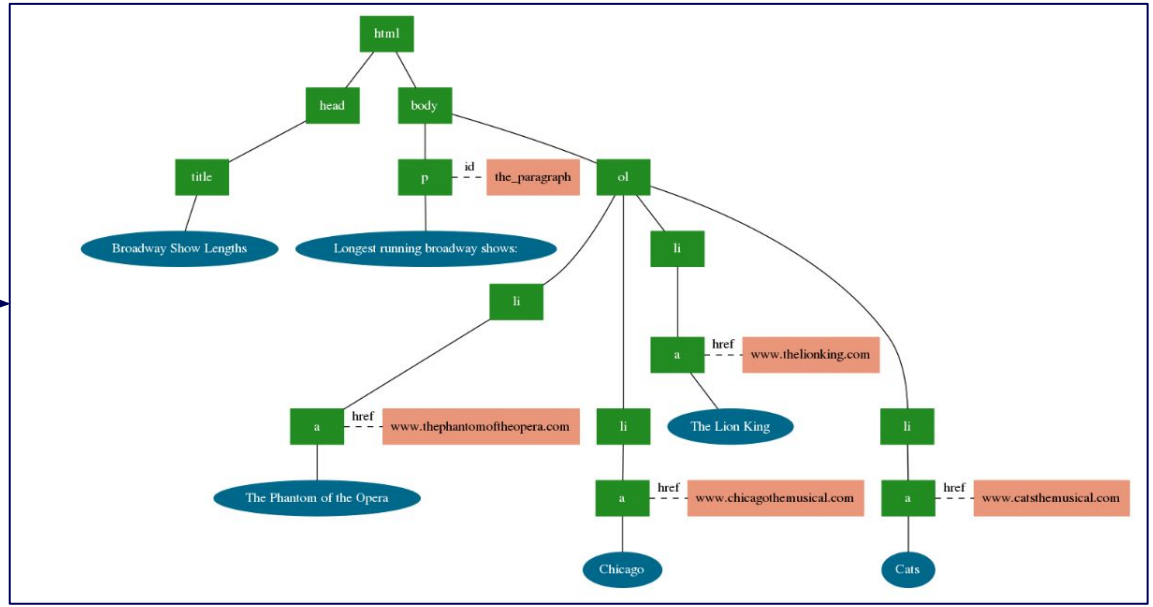
DOM Object Creation Example

```
<html>
  <title>Moby-Dick</title>
  <body>
    <p class="big">Call me Ishmael.  Some years ago...</p>
    <p>There now is your insular city of the Manhattoes...</p>
  </body>
</html>
```



DOM Object Creation Example

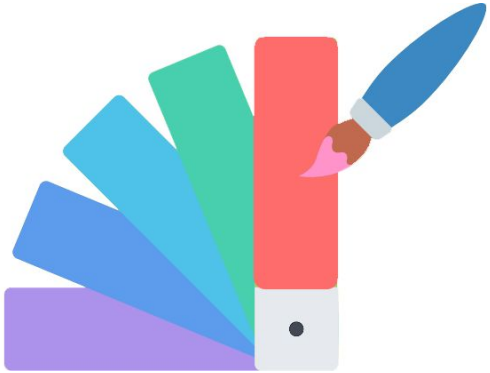
```
<!DOCTYPE html>
<html>
   <head>
      <title>Broadway Show Lengths</title>
   </head>
   <body>
      <p id="the_paragraph">Longest running broadway shows:</p>
      <ol>
         <li><a href="http://www.thephantomoftheopera.com/">The Phantom of the Opera</
         <li><a href="http://www.chicagothemusical.com/">Chicago</a></li>
         <li><a href="http://www.thelionking.com/">The Lion King</a></li>
         <li><a href="http://www.catsthemusical.com/">Cats</a></li>
      </ol>
   </body>
</html>
```



Two important steps to work with DOM

1. Selecting
   - Select the element(s) you want to manipulate.

2. Manipulating
   - Manipulate to change the element based on your need(s).



Selecting Elements

- There are two main ways to select elements in the DOM:
  - **Group 1:** selecting elements based on their specific attributes, such as ID, tag name or class name **(Not recommended)**
    - getElementById
    - getElementsByTagName
    - getElementsByClassName
  - **Group 2:** selecting elements using CSS selectors **(Recommended)**
    - querySelector
    - querySelectorAll
- **Group 2** is better as developers to use the same familiar CSS selector syntax to select elements from the DOM, easier to use and reduce code complexity by selecting multiple elements with a single selector.

querySelector & querySelectorAll

- querySelector: to select a single element:

```
// Finds first h1 element
document.querySelector('h1');

// Finds first element with the ID "red"
document.querySelector('#red');

// Finds first element with the class "big"
document.querySelector('.big');
```

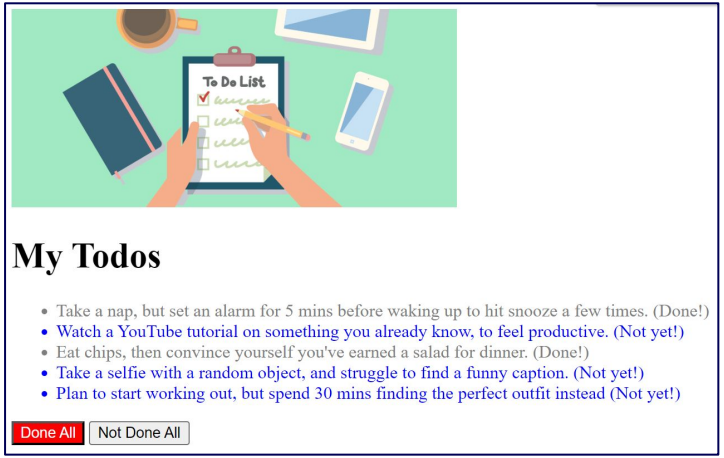
- querySelectorAll: to select multiple elements:

```
// Finds all h1 element(s)
document.querySelectorAll('h1');

// Finds all element(s) with the class "big"
document.querySelectorAll('.big');
```

Query Selector Challenges

- To-do list website link (Starting here).

```
<img class="banner" src="https://www.workflowmax.com/hubfs/6-things-to-do-list.png" alt="" srcset="">
<h1>My Todos</h1>
<ul>
<li class="done">Take a nap, but set an alarm for 5 mins before waking up to hit snooze a few times.</li>
<li class="not-done">Watch a YouTube tutorial on something you already know, to feel productive.</li>
<li class="done">Eat chips, then convince yourself you've earned a salad for dinner.</li>
<li class="not-done">Take a selfie with a random object, and struggle to find a funny caption.</li>
<li class="not-done">Plan to start working out, but spend 30 mins finding the perfect outfit instead</li>
</ul>
<button id="done-all-btn">Done All</button>
<button id="not-done-all-btn">Not Done All</button>
```



Use querySelector or querySelectorAll to select:

- Banner image.
- H1 text.
- All list items.
- Done items.
- Not-done items
- All buttons
- Done button.
- Not-done button.

Query Selector Challenges Solution

```
<img class="banner" src="https://www.workflowmax.com/hubfs/6-things-to-do-list.png" alt="" srcset="">
<h1>My Todos</h1>
<ul>
<li class="done">Take a nap, but set an alarm for 5 mins before waking up to hit snooze a few times.</li>
<li class="not-done">Watch a YouTube tutorial on something you already know, to feel productive.</li>
<li class="done">Eat chips, then convince yourself you've earned a salad for dinner.</li>
<li class="not-done">Take a selfie with a random object, and struggle to find a funny caption.</li>
<li class="not-done">Plan to start working out, but spend 30 mins finding the perfect outfit instead</li>
</ul>
<button id="done-all-btn">Done All</button>
<button id="not-done-all-btn">Not Done All</button>
```

My Todos

- Take a nap, but set an alarm for 5 mins before waking up to hit snooze a few times. (Done!)
- Watch a YouTube tutorial on something you already know, to feel productive. (Not yet!)
- Eat chips, then convince yourself you've earned a salad for dinner. (Done!)
- Take a selfie with a random object, and struggle to find a funny caption. (Not yet!)
- Plan to start working out, but spend 30 mins finding the perfect outfit instead (Not yet!)

Done All | Not Done All

```
// Selecting banner image
bannerImg = document.querySelector(".banner");

// Selecting h1 text
titleText = document.querySelector("h1").innerText;

// Selecting all list items
myListItems = document.querySelectorAll("ul li");

// Selecting all done list items
myDoneListItems = document.querySelectorAll("ul li.done");

// Selecting all not done list items
myNotDoneListItems = document.querySelectorAll("ul li.not-done");

// Selecting all buttons
buttons = document.querySelectorAll("button");

// Selecting the done button
doneButton = document.querySelector("#done-all-btn");

// Selecting the not done button
notDonebutton = document.querySelector("#not-done-all-btn");
```

Manipulating elements

- **Important** (**bold**) properties and methods to manipulate elements:
  - **classList**
  - **getAttribute**()
  - **setAttribute**()
  - **innerText**
  - **textContent**
  - **innerHTML**
  - **style**
  - value
  - **appendChild()**
  - **append()**
  - **prepend()**
  - **removeChild()**
  - **remove()**
  - **createElement**
  - **parentElement**
  - children
  - nextSibling
  - previousSibling

innerHTML

- **Description**: The innerHTML property gets or sets the HTML markup contained within the element.
- **Usage**: It's used when you want to get or change the inner HTML content of an element, which includes all the child elements, text, and HTML tags.
- **Example**: If you set the innerHTML of a element, you can include tags like <p> or , and they will be rendered as part of the document.

outerHTML

- **Description:** The outerHTML property gets or sets the HTML markup of the element, including the element itself.
- **Usage:** It is useful when you need to replace the entire element, including its tags, with new content.
- **Example:** Setting the outerHTML of a will replace the and its contents entirely with the provided markup.

textContent

- **Description:** The textContent property sets or returns the text content of the specified node and all its descendants.
- **Usage:** This property is used when you're only interested in the text within an element, ignoring any HTML tags or structure. It's also a safer alternative to innerHTML for avoiding cross-site scripting (XSS) attacks since it automatically escapes HTML tags.
- **Example:** Getting the textContent of an element will return all the text without any HTML tags.

innerText

- **Description:** The innerText property gets or sets the text content of the specified element, similar to textContent, but with differences in processing.
- **Usage:** innerText is aware of the rendered appearance of text and will respect CSS styling, such as visibility and display properties, meaning it won't return the text of elements that are styled with display: none.
- **Example:** Unlike textContent, innerText will trigger a reflow (the process of laying out the page again), which can affect performance. It also returns the text as it appears to the user, considering CSS styles.

nodeValue

- **Description:** This property is used to get or set the value of a text node or a comment node. It does not work on HTML elements themselves but rather on the text or comment nodes within those elements.
- **Usage:** It's useful when you're dealing specifically with text or comment nodes and want to manipulate their content.
- **Example**:
  - For example, if you have a reference to a text node, you can retrieve its content like so:
    - var textContent = textNode.nodeValue;
  - This replaces the current content of the node with the new string:
    - textNode.nodeValue = "New text content";

The difference between innerHTML, textContent, innerText, and nodeValue

index.html

```html
<div id="example">
    Visible text,
    <span style="display: none;">hidden text,</span>
    <span>more visible text.</span>
</div>
```

script.js

```js
const element = document.getElementById('example');

console.log('outerHTML:', element.outerHTML);
console.log('innerHTML:', element.innerHTML);
console.log('textContent:', element.textContent);
console.log('innerText:', element.innerText);

const firstTextNode = element.childNodes[0];
console.log('nodeValue:', firstTextNode.nodeValue);
```

Output:

```
outerHTML: <div id="example">
        Visible text,
        <span style="display: none;">hidden text,</span>
        <span>more visible text.</span>
    </div>

innerHTML:
        Visible text,
        <span style="display: none;">hidden text,</span>
        <span>more visible text.</span>

textContent:
        Visible text,
        hidden text,
        more visible text.

innerText: Visible text, more visible text.

nodeValue:
        Visible text,
```

[Question] Can you explain for each output?

[The demo link]

getAttribute() and setAttribute()

- **Description**: These methods get and set attributes of elements.
- **Usage**: They are used to manipulate the attributes of an element, such as id, class, src, etc. getAttribute() retrieves the value of an attribute, and setAttribute() changes or sets the value.
- **Example**:
  - element.getAttribute('href') would return the href value of an anchor element.
  - element.setAttribute('id', 'newId') sets the id of an element to "newId".

classList

- **Description**: This property provides methods to add, remove, and toggle CSS classes on an element.
- **Methods**:
  - **add**(className): Adds a class to the element's list of classes.
  - **remove**(className): Removes a class from the list.
  - **contains**(className): Checks if a class is in the element's list of classes.
  - **toggle**(className): Toggles a class in the list. If the class exists, it's removed; if it doesn't, it's added.
  - **replace**(oldClassName, newClassName): Replaces an existing class with a new class.
- **Usage**: classList is extremely useful for manipulating an element's classes without worrying about accidentally overwriting existing classes. It's especially handy for adding, removing, or toggling stateful classes, like .active or .hidden, in response to user interactions.

style

- **Description**: The style property is an object that represents the inline style attributes of an element. Through the style property, you can get or set the inline styles of an element.
- **Usage**: While classList is used for adding or removing classes, the style property is used for directly manipulating the styles of an element. It's useful for dynamically changing the appearance of elements, such as colors, sizes, and positions, directly with JavaScript.
- **Example**: element.style.backgroundColor = 'blue' changes the element's background color to blue.

createElement()

- **Description**: This method creates an Element Node with the specified tag name.
- **Usage**: It's used when you want to create a new element in the document that can then be inserted into the document tree using other methods.
- **Example**: let newElement = document.createElement('div');

appendChild()

- **Description:** This method adds a node as the last child of a node.
- **Usage:** After creating a new element (or selecting an existing one), you can use appendChild() to insert it into the document. It's commonly used in conjunction with createElement().
- **Example**:
  - **Create a new element:**
    - var newElement = document.createElement('div');
  - **Set some properties (optional):**
    - newElement.textContent = 'Hello, World!';
  - **Select a parent element:**
    - var parentElement = document.getElementById('parentDiv');
  - **Append the new element to the parent:**
    - parentElement.appendChild(newElement);

insertBefore()

- **Description**: This method inserts a node before a reference node as a child of a specified parent node.
- **Usage**: If you need to insert an element at a specific position among a parent's children, this method is very helpful.
- **Example**:
  - **Step 1**: Identify or create the new node you want to insert.
    - let newNode = document.createElement('div');
  - **Step 2**: Identify the reference node before which the new node will be inserted.
    - let referenceNode = document.querySelector('#some-element');
  - **Step 3**: Identify the parent node of the reference node.
    - It could be the same as referenceNode.parentNode or any node that you've identified separately.
  - **Step 4**: Use the insertBefore() method to insert the new node before the reference node within the parent node.
    - referenceNode.parentNode.insertBefore(newNode, referenceNode);

removeChild()

- **Description**: This method removes a child node from the DOM.
- **Usage**: Used to remove an existing element from the document.
- **Example**:
  - Assume there is a div element with id 'parent' and it has a child element <p> with id 'child'
    - var parentElement = document.getElementById('parent');
    - var childElement = document.getElementById('child');
  - **Remove** the child element from the parent
    - parentElement.removeChild(childElement);

Manipulating Element Challenges

```
<img class="banner" src="https://www.workflowmax.com/hubfs/6-things-to-do-list.png" alt="" srcset="">
<h1>My Todos</h1>
<ul>
<li class="done">Take a nap, but set an alarm for 5 mins before waking up to hit snooze a few times.</li>
<li class="not-done">Watch a YouTube tutorial on something you already know, to feel productive.</li>
<li class="done">Eat chips, then convince yourself you've earned a salad for dinner.</li>
<li class="not-done">Take a selfie with a random object, and struggle to find a funny caption.</li>
<li class="not-done">Plan to start working out, but spend 30 mins finding the perfect outfit instead</li>
</ul>
<button id="done-all-btn">Done All</button>
<button id="not-done-all-btn">Not Done All</button>
```

My Todos

- Take a nap, but set an alarm for 5 mins before waking up to hit snooze a few times. (Done!)
- Watch a YouTube tutorial on something you already know, to feel productive. (Not yet!)
- Eat chips, then convince yourself you've earned a salad for dinner. (Done!)
- Take a selfie with a random object, and struggle to find a funny caption. (Not yet!)
- Plan to start working out, but spend 30 mins finding the perfect outfit instead (Not yet!)

Done All | Not Done All

```
.banner {
    width:400px;
}

.not-done {
    color: blue;
}

.not-done::after {
    content: " (Not yet!)";
}

.done {
    color: grey;
}

.done::after {
    content: " (Done!)";
}

#done-all-btn {
    background-color: red;
    color: white;
}
```

Using JS to make these changes:

- Get h1 text and change its text value and color.
- Get second list item text and change its value.
- Get the image link url and change it to another link.
- Change all list items to be done.
- Change all list items from done to be not done and vice versa.

Manipulating Element Challenges Solution

```
<img class="banner" src="https://www.workflowmax.com/hubfs/6-things-to-do-list.png" alt="" srcset="">
<h1>My Todos</h1>
<ul>
<li class="done">Take a nap, but set an alarm for 5 mins before waking up to hit snooze a few times.</li>
<li class="not-done">Watch a YouTube tutorial on something you already know, to feel productive.</li>
<li class="done">Eat chips, then convince yourself you've earned a salad for dinner.</li>
<li class="not-done">Take a selfie with a random object, and struggle to find a funny caption.</li>
<li class="not-done">Plan to start working out, but spend 30 mins finding the perfect outfit instead</li>
</ul>
<button id="done-all-btn">Done All</button>
<button id="not-done-all-btn">Not Done All</button>
```

MY NEW TITLE FOR TO DO LIST

- Take a nap, but set an alarm for 5 mins before waking up to hit snooze a few times. (Not yet!)
- I want to sleep till the end of the day! (Not yet!)
- Eat chips, then convince yourself you've earned a salad for dinner. (Not yet!)
- Take a selfie with a random object, and struggle to find a funny caption. (Not yet!)
- Plan to start working out, but spend 30 mins finding the perfect outfit instead (Not yet!)

Done All | Not Done All

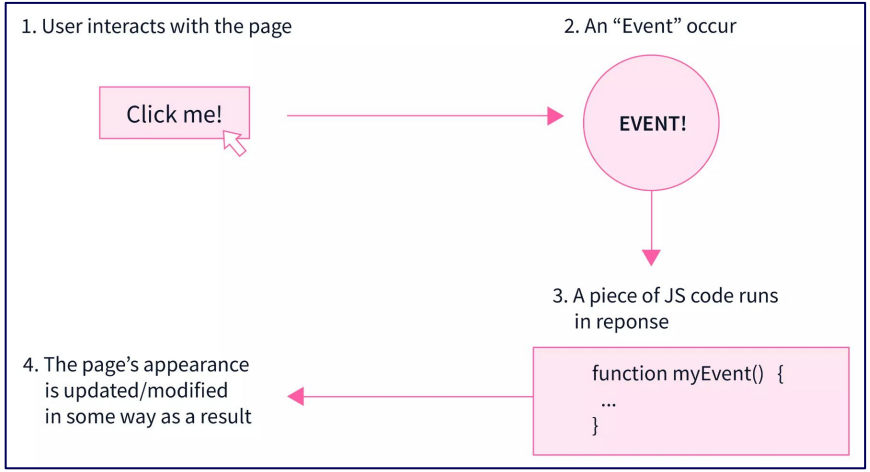
```
// Selecting h1 text and change its text value and color
myHeader = document.querySelector("h1");
myHeader.innerText = "MY NEW TITLE FOR TO DO LIST";
myHeader.style.color = "red";

// Get second list item text and change its value
mySecondListItem = document.querySelector("ul li:nth-child(2)");
mySecondListItem.innerText = "I want to sleep till the end of the day!";

// Get the imagine link url and change it to another link
bannerImg = document.querySelector(".banner");
bannerImg.setAttribute('src','https://www.sheknows.com/wp-content/
uploads/2018/08/dgldbejjyofmcsfvkfd9.jpeg?w=1920');

// Change all list items to be done
myListItems = document.querySelectorAll("ul li");
myListItems.forEach(item => {
    item.classList.remove("not-done");
    item.classList.add("done");
});

// Change all list items from done to be not done and vice versa.
myListItems = document.querySelectorAll("ul li");
myListItems.forEach(item => {
    item.classList.toggle("not-done");
    item.classList.toggle("done");
});
```

DOM Events

Events in Javascript

- **Events** are **actions** or **occurrences** that **happen in the browser**, such as a button being clicked, a page finishing loading, or a user typing in a text field.
- Events can be **triggered by the user** or **by the browser itself.**
- To **handle events**, you can use **event listeners**.
  - An **event listener** is a function that waits for a specific event to occur and then executes some code in response.

- You can **attach event listeners** to elements on a web page, such as buttons or input fields, using the **addEventListener()** method.



https://developer.mozilla.org/en-US/docs/Web/Events

Some Event Examples

- clicks
- drags
- drops
- hovers
- scrolls
- form
- submission
- key presses
- focus/blur
- mouse wheel
- double click
- copying
- pasting
- audio start
- screen resize
- printing

https://developer.mozilla.org/en-US/docs/Web/Events

Add a Event Listener to a HTML element

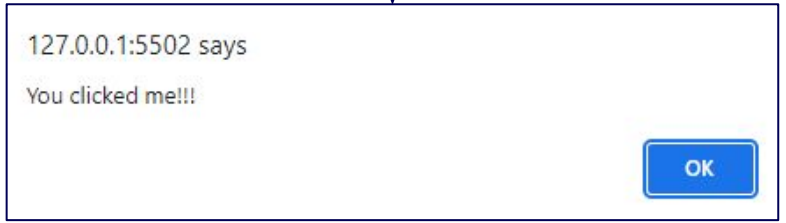
- To trigger a function when an event happens, we need to specify the event type and a callback function to run.
- The demo link is here.

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Event Listener Example</title>
</head>
<body>
    <h1>Click me!</h1>

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

script.js

```
let button = document.querySelector('h1');

button.addEventListener('click', () => {
    alert('You clicked me!!!');
})
```



https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener

Question ❓

- What happens if we put the js script in the head of the HTML instead of at the end of the body?

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Event Listener Example</title>
    <script src="script.js"></script>
</head>
<body>
    <h1>Click me!</h1>
</body>
</html>
```

script.js

```
let button = document.querySelector('h1');

button.addEventListener('click', () => {
    alert('You clicked me!!!');
})
```

❓

Solution with window.onload

- To ensure that the JS code runs after all the HTML elements have loaded, you can wrap your code inside the **window.onload** event handler.
- This event fires when all the HTML content, including images and other resources, has been loaded.
- The demo link is here.

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Event Listener Example</title>
    <script src="script.js"></script>
</head>
<body>
    <h1>Click me!</h1>
</body>
</html>
```

script.js

```
window.onload = () => {
    let button = document.querySelector('h1');

    button.addEventListener('click', () => {
        alert('You clicked me!!!');
    })
};
```

EventListener Challenges

```
<img class="banner" src="https://www.workflowmax.com/hubfs/6-things-to-do-list.png" alt="" srcset="">
<h1>My Todos</h1>
<ul>
<li class="done">Take a nap, but set an alarm for 5 mins before waking up to hit snooze a few times.</li>
<li class="not-done">Watch a YouTube tutorial on something you already know, to feel productive.</li>
<li class="done">Eat chips, then convince yourself you've earned a salad for dinner.</li>
<li class="not-done">Take a selfie with a random object, and struggle to find a funny caption.</li>
<li class="not-done">Plan to start working out, but spend 30 mins finding the perfect outfit instead</li>
</ul>
<button id="done-all-btn">Done All</button>
<button id="not-done-all-btn">Not Done All</button>
```

My Todos

- Take a nap, but set an alarm for 5 mins before waking up to hit snooze a few times. (Done!)
- Watch a YouTube tutorial on something you already know, to feel productive. (Not yet!)
- Eat chips, then convince yourself you've earned a salad for dinner. (Done!)
- Take a selfie with a random object, and struggle to find a funny caption. (Not yet!)
- Plan to start working out, but spend 30 mins finding the perfect outfit instead (Not yet!)

Done All | Not Done All

```
.banner {
    width:400px;
}

.not-done {
    color: blue;
}

.not-done::after {
    content: " (Not yet!)";
}

.done {
    color: grey;
}

.done::after {
    content: " (Done!)";
}

#done-all-btn {
    background-color: red;
    color: white;
}
```

Using addEventListener() to handle these events:

- Clicking on the button “Done All” makes all list items to be done.
- Clicking on the button “Not Done All” makes all list items to be done.
- Clicking on any item list will make the item toggle from done to not done or vice versa.

EventListener Challenges Solution

```html
<img class="banner" src="https://www.workflowmax.com/hubfs/6-things-to-do-list.png" alt="" srcset="">
<h1>My Todos</h1>
<ul>
<li class="done">Take a nap, but set an alarm for 5 mins before waking up to hit snooze a few times.</li>
<li class="not-done">Watch a YouTube tutorial on something you already know, to feel productive.</li>
<li class="done">Eat chips, then convince yourself you've earned a salad for dinner.</li>
<li class="not-done">Take a selfie with a random object, and struggle to find a funny caption.</li>
<li class="not-done">Plan to start working out, but spend 30 mins finding the perfect outfit instead</li>
</ul>
<button id="done-all-btn">Done All</button>
<button id="not-done-all-btn">Not Done All</button>
```

My Todos

- Take a nap, but set an alarm for 5 mins before waking up to hit snooze a few times. (Done!)
- Watch a YouTube tutorial on something you already know, to feel productive. (Not yet!)
- Eat chips, then convince yourself you've earned a salad for dinner. (Done!)
- Take a selfie with a random object, and struggle to find a funny caption. (Not yet!)
- Plan to start working out, but spend 30 mins finding the perfect outfit instead (Not yet!)

Done All | Not Done All

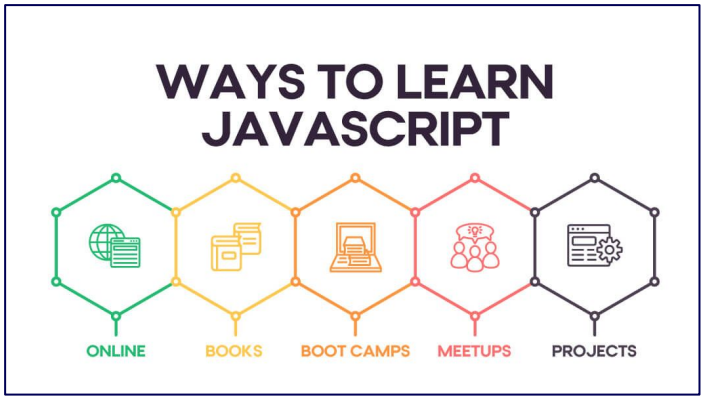
```javascript
// Clicking on the button "Done All" makes all list items to be done.
doneButton = document.querySelector("#done-all-btn");
doneButton.addEventListener('click', () => {
    myListItems = document.querySelectorAll("ul li");
    myListItems.forEach(item => {
        item.classList.remove("not-done");
        item.classList.add("done");
    })
})

// Clicking on the button "Not Done All" makes all list items to be done.
notDonebutton = document.querySelector("#not-done-all-btn");
notDonebutton.addEventListener('click', () => {
    myListItems = document.querySelectorAll("ul li");
    myListItems.forEach(item => {
        item.classList.remove("done");
        item.classList.add("not-done");
    })
})

// Clicking on any item list will make the item toggle from done to not done or vice versa.
myListItems = document.querySelectorAll("ul li");
myListItems.forEach((item) => {
    item.addEventListener('click', () => {
      item.classList.toggle('done');
      item.classList.toggle('not-done');
    });
});
```

To Infinity and Beyond with Javascript!

- If you want practise and learn more Javascript, there are 3 free resources that I recommend:

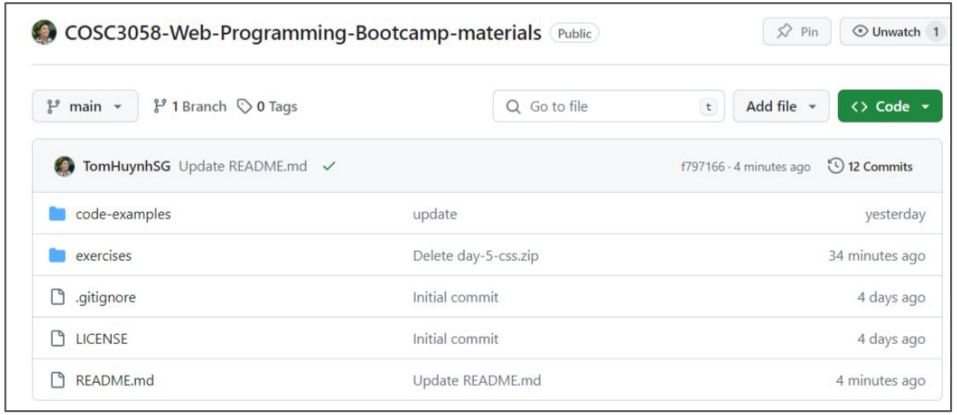
1. https://github.com/getify/You-Dont-Know-JS
2. http://javascript.info/
3. http://dmitrysoshnikov.com/ecmascript/javascript-the-core-2nd-edition/

The more you invest in your Javascript skill, the more it will benefit you in the near future!



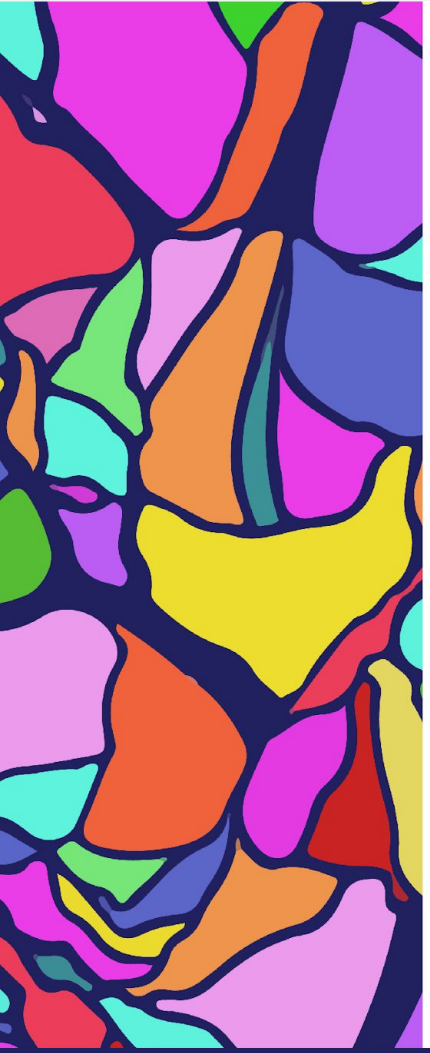
COSC3058 Materials Github Repo

- All of the file examples of the lectures and tutorials will be uploaded weekly and available for everyone to check them out.
- [Github repository link](#).



JS Reference

- MDN (Mozilla Developer Network) is always the best reference source.
  - Link: https://developer.mozilla.org/en-US/docs/Web/JavaScript
- Comprehensive Reference Devdoc.io:
  - Link: https://devdocs.io/javascript/
- W3School for Javascript:
  - Link: https://www.w3schools.com/js/default.asp
- JS Cheat Sheet:
  - Link: https://ilovecoding.org/blog/js-cheatsheet
- DOM Documents:
  - Link: https://www.w3schools.com/jsref/dom_obj_document.asp
- DOM Style objects:
  - Link: https://www.w3schools.com/jsref/dom_obj_style.asp



“Motivation is what gets you started.
Habit is what keeps you going.”

Jim Ryun

RMIT UNIVERSITY