We use var to define variable in javascripte also we use prompt to get answer form our users

Prompt(“What is your name ”)

Alert(“Do it again it is wrong”)

Slice funtion : it slice the number

Var name = “maaz”

Name.slice(0,1) // you output is is m

Alert([prompt(“Compose your tweet ”).slice(0,140)

Your will not have more than 140 character

toUpperCase() it will change the string to uppercase letter it is really usefull

Arrays.pop it will take the last element of the array and it will delete it

Math.floor(Math.random() \* max);

Ways selecting elements in the dom

The first way is to select by tag name document.getElementByTagName("Li") // This get the list name

IT will Fetch all the elements with that tag name

When we want to change the style of the tag

document.getElementByTagName("li")[2]

This index 2 or number 2  of the list .style.color = "purple"

Another one to select the element is

document.GetElementByClassName("Btn") The two select multiple items

But getElementById selects only one item

And querySelector() Also the same

querySelector() I can select id Class and also element like h1 , h2 , ol, li , like Everything things you like

querySelector("li.item") we can combine the element

querySelector("#item a ") we can combine the element

So if we want to select mo

re than one item we should say

querySelectorAll()

**ClassList**

document.querySelector("button").classList.add("invisible");

now we here add to the class another class

Classlist // Will show us our class in the HTML that we class

Classlist.remove("invisible") it is very usefull we can add style in css than

We can add it in javascritpe remmber the video

we can use toggle also it useful  // on and off  like true // off

document.querySelector("button").classList.add("invisible");

now we here add to the class another class

ClasslIst // Will show us our class in the HTML that we class

Classlist.remove("invisible")

we can use toggle also it useful  // on and off  like true // off

**textContent :-**

***Innerhtml : gave you the real element inside the html but textcontent gave only the content***

***Also in innerHtml you can have extra html code like “<emp>Good bye </emp>”***

**TO play song or audio in javascripte you use**

**Var audio = new Audio(“”)**

**Audio.play();**

/This/ we can get button things triggers

Console.log(this.style.color = “white”)

this.style.color = “white”

Var x = this.innerHtml

**Object in javascripte**

Var x = {

Name:”maaz”

}

**Constructor function**

Function bellyboy (x){

This.x = x;

]

Var belly1 = new bellyboy(“maaz”);

Hey guys,

**TL;DR keypress is now deprecated, you should use keydown instead.**

Just a quick heads up, as technology moves incredibly fast, every week or so something else will change. This is just a quick reminder that in the next lesson, when we cover detecting key presses, you should be using the keydown event listener instead of keypress.

Use the event name in methods like [addEventListener()](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener" \o "addEventListener()), or set an event handler property.

addEventListener("keypress", (event) => {});

onkeypress = (event) => {};

**Jquery**

jQuery is a fast, lightweight, and feature-rich JavaScript library that simplifies tasks like HTML document manipulation, event handling, animation, and AJAX. It provides a simpler syntax to work with JavaScript and is widely used in web development.

### ****Why Use jQuery?****

* Simplifies DOM manipulation (addClass, removeClass, css, html)
* Cross-browser compatibility
* Easy event handling (click, hover, on)
* AJAX support ($.ajax, $.get, $.post)
* Built-in animations (fadeIn, slideUp, toggle)

### ****Common jQuery Features****

#### ****1. Selecting Elements****

$("#myID"); // Selects an element with ID "myID"

$(".myClass"); // Selects all elements with class "myClass"

$("p"); // Selects all <p> elements

1. Manipulating Content

$("#myID").text("Hello, World!"); // Changes text

$("#myID").html("<b>Bold Text</b>"); // Inserts HTML

$("#myID").val("New Value"); // Changes input value

3. CSS Manipulation

$("#myID").css("color", "red"); // Changes text color to red $("#myID").addClass("newClass"); // Adds a class $("#myID").removeClass("oldClass"); // Removes a clas

4. Event Handling

$("#btn").click(function(){

alert("Button Clicked!");

});

$("#myDiv").hover(function(){

$(this).css("background", "yellow");

}, function(){

$(this).css("background", "white");

});

5. Animations & Effects

$("#myDiv").hide(); // Hides an element

$("#myDiv").show(); // Shows an element

$("#myDiv").fadeOut(); // Fades out

$("#myDiv").slideUp(); // Slides up

$("#myDiv").toggle(); // Toggles visibility

6. AJAX Requests

$.get("data.txt", function(response){

$("#result").text(response);

});

$.post("submit.php", {name: "Maaz"}, function(data){

alert("Response: " + data);

});

The Math.round() static method returns the value of a number rounded to the nearest integer.

.push IT puss it in to the end of the array IT is really help full for you guys

If we want to have get the attributes inside the javascripte we have to add

Document.quertSelector(“x”).attributes

.getAttribute(); we get single attributes

.setAttribute(“href”,”https:”) /// change the attributes

Document.querySelector(“click”, fuction (){

})

 //2. Inside the handler, create a new variable called userChosenColour to store the id of the button that got clicked.

  var userChosenColour = $(this).attr("id");

  //4. Add the contents of the variable userChosenColour created in step 2 to the end of this new userClickedPattern

  playSound(userChosenColour)

  userClickedPattern.push(userChosenColour);

var buttonColours = ["red", "blue", "green", "yellow"];

var gamePattern = [];

var userClickedPattern = [];

$(".btn").click(function() {

  var userChosenColour = $(this).attr("id");

  userClickedPattern.push(userChosenColour);

  //1. In the same way we played sound in nextSequence() , when a user clicks on a button, the corresponding sound should be played.

  playSound(userChosenColour);

});

function nextSequence() {

  var randomNumber = Math.floor(Math.random() \* 4);

  var randomChosenColour = buttonColours[randomNumber];

  gamePattern.push(randomChosenColour);

  $("#" + randomChosenColour).fadeIn(100).fadeOut(100).fadeIn(100);

  //4. Refactor the code in playSound() so that it will work for both playing sound in nextSequence() and when the user clicks a button.

  playSound(randomChosenColour);

}

//2. Creae a new function called playSound() that takes a single input parameter called name.

function playSound(name) {

  //3. Take the code we used to play sound in the nextSequence() function and add it to playSound().

  var audio = new Audio("sounds/" + name + ".mp3");

  audio.play();

}

use Google/Stackoverflow to figure out how you can use Javascript to remove the pressed class after a 100 milliseconds.

Once complete, you will get this effect when you click on any button.

setTimeout(function(){

$(formMessages).removeClass('error');

//....and whatever else you need to do

}, 3000);

if (input\_user.value.trim() === "") { // Trim to avoid spaces being treated as input output.textContent = "Please enter a valid number"; output.style.color = "red"; return; // Stop execution here }

Sure! When you add an event listener to a button inside a <form>, clicking the button will trigger the form's default behavior, which is to submit the form and reload the page.

### Why does this happen?

By default, when you place a <button> inside a <form>, it acts as a **submit button**, meaning it will try to send the form data to a server. Even if the button isn’t explicitly marked as type="submit", some browsers assume it is.

### How to prevent this?

To stop this from happening when clicking the **sortButton**, you need to use event.preventDefault() inside your event listener. This method prevents the default action (which, in this case, is form submission).

document.getElementById("sortButton").addEventListener("click", function(event) {

event.preventDefault(); // Prevents form submission

console.log("Sort button clicked!");

// Your sorting logic here...

});

the sortButton. Because buttons associated with a form element submit by default, you need to prevent that behavior. Call event.preventDefault() in your function to do this.

Remember that .getElementsByClassName() method returns an HTMLCollection, which is an array-like object of all the elements that have a matching class name. You can use the spread operator to convert it into an array.

const sortButton = document.getElementById("sort");

const sortInputArray = (event) => {

  event.preventDefault();

  const inputValues = document.getElementsByClassName("values-dropdown");

}

sortButton.addEventListener("click", sortInputArray);

Convert the document.getElementsByClassName() call to an array with the spread operator and assign it to a variable called inputValues.

 const inputValues = document.getElementsByClassName("values-dropdown");

const inputValues = [...document.getElementsByClassName("values-dropdown")];

Update your .map() callback to call the Number() function. Pass dropdown.value to that function call.

const inputValues = [...document.getElementsByClassName("values-dropdown")].map((dropdown) => Number(dropdown.value));

  console.log(inputValues);

To perform an action on each element in the array, use the method that is meant for iterating over arrays.

Use the forEach() method, and pass it an empty callback which takes num and i as the parameters.

This is array.forEach((num, i) => { // Empty callback function for now });

const updateUI = (array = []) => {

  array.forEach((num,i) => {

  })

}

Time to implement another sorting algorithm. This time, you'll be implementing a selection sort. Selection sort works by finding the smallest value in the array, then swapping it with the first value in the array. Then, it finds the next smallest value in the array, and swaps it with the second value in the array. It continues iterating through the array until it is completely sorted.

const bubbleSort = (array) => {

  for (let i = 0; i < array.length; i++) {

    for (let j = 0; j < array.length - 1; j++) {

      if (array[j] > array[j + 1]) {

        const temp = array[j];

        array[j] = array[j + 1];

        array[j + 1] = temp;

      }

    }

  }

A selection sort relies on tracking the index of the smallest value in the array. Declare a variable minIndex and set it to i - this ensures that if your current value is the smallest, it will be swapped with itself and not be moved. You will need to be able to reassign the value of minIndex as you iterate through the array.

Then, write another for loop, using j as the iterator. This loop needs to start at the index after i and iterate through the rest of the array.

const selectionSort = (array) => {

  for (let i = 0; i < array.length; i++) {

    let minIndex = i;

    for (let j = i + 1; j < array.length; j++) {

      if (array[j] < array[minIndex]) {

        minIndex = j;

      }

    }

    const temp = array[i];

    array[i] = array[minIndex];

    array[minIndex] = temp;

  }

  return array;

}

The last sorting algorithm you will implement is the insertion sort. This algorithm works by building up a sorted array at the beginning of the list. It begins the sorted array with the first element. Then it inspects the next element and swaps it backward into the sorted array until it is in a sorted position, and so on.

Start by declaring an insertionSort variable and assigning it an arrow function which takes an array parameter.

Notice how the number 10 is placed at the beginning of the array. This is because the default behavior of .sort() is to convert the numbers values to strings, and sort them alphabetically. And "10" comes before "2" alphabetically

o fix this, you can pass a callback function to the .sort() method. The callback function has two parameters - for yours, use a and b. The parameters of a and b represent the number values in the array that will be sorted.

If you press the Sort button again, you should see that 10 is now in the correct position of the Output.

const sortedValues = inputValues.sort((a, b) => {

    return a - b;

  });

This is when we want to get values

 const value = document.querySelector("#numbers").value

The .split() method takes a string and splits it into an array of strings. You can pass it a string of characters or a RegEx to use as a separator. For example, string.split(",") would split the string at each comma and return an array of strings.

const array = value.split(/,\s\*/g);

console.log(array);

Create a numbers variable and assign it the value of array.map(). Remember that .map() creates a new array, instead of mutating the original array.

# **Step 5**

The .map() method takes a callback function as its first argument. This callback function takes a few arguments, but the first one is the current element being processed. Here is an example:

**array.map(el => {**

**})**

The callback function needs to return a value. In this case, you want to return the value of each element converted to a number. You can do this by using the Number() constructor, passing the element as an argument.

Notes : in html it gave us String so here we change the array to the Number by using map which gave us new array Also it gave us number

 const numbers = array.map(el =>{

    Number(el)

  });

const numbers = array.map(el => Number(el));

You need to filter these values out – thankfully, arrays have a method specifically for this. The .filter() method will allow you to filter elements out of an array, creating a new array in the process.

However, you cannot check for equality here, because NaN is not equal to itself. Instead, you can use the isNaN() method, which returns true if the argument is NaN.

The !isNaN(element) part is a way to filter out NaN (Not-a-Number) values from the array. Let’s break it down:

isNaN(element): This function checks whether the given element is NaN. It returns true if the element is NaN, and false otherwise.

!isNaN(element): The ! operator negates the result. So, if isNaN(element) returns true (meaning the element is NaN), !isNaN(element) will return false. If isNaN(element) returns false (meaning the element is a valid number), !sNaN(element) will return true.

const filtered = numbers.filter(el => !isNaN(el));

Array methods can often be chained together to perform multiple operations at once. As an example:

**array.map().filter();**

The .map() method is called on the array, and then the .filter() method is called on the result of the .map() method. This is called method chaining.

Following that example, remove your filtered variable, and chain your .filter() call to your .map() call above. Do not remove either of the callback functions.

. The .reduce() method takes an array and applies a callback function to condense the array into a single value

**array.reduce((acc, el) => {**

**});**

**array.reduce((acc, el) => acc + el.toLowerCase(), "");**

To be safe, it's best to set an initial value. Here is an example of setting the initial value to an empty string:

**array.reduce((acc, el) => acc + el.toLowerCase(), "");**

const getMean = (array) => {

  const sum = array.reduce((acc, el) => acc + el, 0);

  const mean = sum / array.length;

  return mean;

}

To use implicit return syntax

const getMean = (array) => array.reduce((acc, el) => acc + el, 0) / array.length;

If you test your form with a list of numbers, you should see the mean display on the page. However, this only works because freeCodeCamp's iframe has special settings. Normally, when a form is submitted, the event triggers a page refresh.

To resolve this, add return false; after your calculate(); call in the onsubmit attribute.

<form onsubmit="calculate() return false;">

By default, the .sort() method converts the elements of an array into strings, then sorts them alphabetically. The .sort() method mutates the original array. This works well for strings, but not so well for numbers. For example, 10 comes before 2 when sorted as strings, but 2 comes before 10 when sorted as numbers.

To fix this, you can pass in a callback function to the .sort() method. This function takes two arguments, which represent the two elements being compared. The function should return a value less than 0 if the first element should come before the second element, a value greater than 0 if the first element should come after the second element, and 0 if the two elements should remain in their current positions.

const getMedian = (array) => {

  const sorted = array.sort((a,b) =>{

    return a-b

  });

}

**// check if array length is even**

**arr.length % 2 === 0;**

**// check if array length is odd**

**arr.length % 2 === 1;**

Here is a longer example finding the middle number of an array with 5 elements:]

**const numbers = [1, 2, 3, 4, 5];const middleNumber = numbers[Math.floor(numbers.length / 2)];**

**console.log(middleNumber); // 3**

This ways you can find middle array of the element

const oddListMedian = testArr1[Math.floor(testArr1.length / 2)]

**const numbers = [1, 2, 3, 4];**

**const firstMiddleNumber = numbers[numbers.length / 2];const secondMiddleNumber = numbers[(numbers.length / 2) - 1];// result is 2.5getMean([firstMiddleNumber, secondMiddleNumber]);**

const midIndex = testArr2.length / 2;

const evenListMedian = (testArr2[midIndex - 1] + testArr2[midIndex]) / 2;

console.log(evenListMedian);

const evenListMedian = getMean([testArr2[testArr2.length / 2 - 1], testArr2[testArr2.length / 2]]);

const sorted = array.sort((a, b) => a - b);

**Notes The .sort() method mutates** the original array - in other words, it modifies the order of the elements directly. This is generally considered bad practice, as it can result in unexpected side effects.

Instead, you should use the .toSorted() method, which creates a new array. Change your .sort() call to .toSorted(). Do not modify the callback function.

 const sorted = array.toSorted((a, b) => a - b);

const getMedian = (array) => {

  const sorted = array.toSorted((a, b) => a - b);

   const midIndex = Math.floor(sorted.length / 2);

  if(sorted.length % 2 == 0){

    return (sorted[midIndex] + sorted[midIndex - 1])/2

  }else{

    return  sorted[midIndex]

  }

}