**USE console.log(event) FOR ASSOSIATED EVENTS TO THAT ELMENT**

**Variable declaration**

var a;

let a;

**Logging HTML**

console.log(document.body);

**Alerts**

Alert();

Prompt();

Confirm();

var deleteConfirm = window.confirm("Are you sure you want to delete this?");

**Typeof**

console.log(typeof(num1));

console.log(typeof(string));

**Types of Variables**

var myInteger = 12; // 32-bit number (from -2,147,483,648 to 2,147,483,647)

var myLong = 9310141419482; // 64-bit number (from -9,223,372,036,854,775,808 to

9,223,372,036,854,775,807)

var myFloat = 5.5; // 32-bit floating-point number (decimal)

var myDouble = 9310141419482.22; // 64-bit floating-point number

var myBoolean = true; // 1-bit true/false (0 or 1)

var myBoolean2 = false;

var myNotANumber = NaN;

var NaN\_Example = 0/0; // NaN: Division by Zero is not possible

var notDefined; // undefined: we didn't define it to anything yet

window.alert(aRandomVariable); // undefined

var myNull = null; // null

**Arrays and Objects**

var myArray = []; // empty array

var favoriteFruits = ["apple", "orange", "strawberry"];

var primeNumbers = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31];

var randomVariables = [2, "any type works", undefined, null, true, 2.51];

myArray = ["zero", "one", "two"];

window.alert(myArray[0]); // 0 is the first element of an array

// in this case, the value would be "zero"

**.toString();**

**.join();** num.join(“-“); [1,3,5] -> 1-3-5

**.push();**

**.pop();**

**a.concat(b);**

**.sort**

**.slice**

**.reverse**

**.map**

// let arr = [45, 23, 21]

// let a = arr.map((value, index, array) => {

// console.log(value, index, array)

// return value + index

// })

// console.log(arr)

**.filter**

// let arr2 = [45, 23, 21, 0, 3, 5]

// let a2 = arr2.filter((a) => {

// return a < 10

// })

// console.log(a2, arr2)

**.reduce**

let arr3 = [1, 2, 3, 5, 2, 1]

const reduce\_func = (h1, h2) => {

return h1 + h2

}

let newarr3 = arr3.reduce(reduce\_func)

console.log(newarr3)

**An object** is a group of values; unlike arrays, we can do something better than them:

myObject = {};

john = {firstname: "John", lastname: "Doe", fullname: "John Doe"};

billy = {

firstname: "Billy",

lastname: undefined,

fullname: "Billy"

};

window.alert(john.fullname); // John Doe

window.alert(billy.firstname); // Billy

Rather than making an array ["John Doe", "Billy"] and calling myArray[0], we can just call john.fullname

billy.fullname.

**Math library**

Math.floor("a")

Math.sqrt(-1)

Math.random()\*num //num = number to which you want to find random

Number i.e. 0 to num

**Formatting console output**

console.log('%s has %d points', 'Sam', 100);

Displays = Sam has 100 points.

The full list of format specifiers in JavaScript is:

%s Formats the value as a string

%i or %d Formats the value as an integer

%f Formats the value as a floating point value

%o Formats the value as an expandable DOM element

%O Formats the value as an expandable JavaScript object

%c Applies CSS style rules to the output string as specified by the second parameter

**console.table()**

var personArr = [

{

"personId": 123,

"name": "Jhon",

"city": "Melbourne",

"phoneNo": "1234567890"

},

{

"personId": 124,

"name": "Amelia",

"city": "Sydney",

"phoneNo": "1234567890"

},

{

"personId": 125,

"name": "Emily",

"city": "Perth",

"phoneNo": "1234567890"

},

{

"personId": 126,

"name": "Abraham",

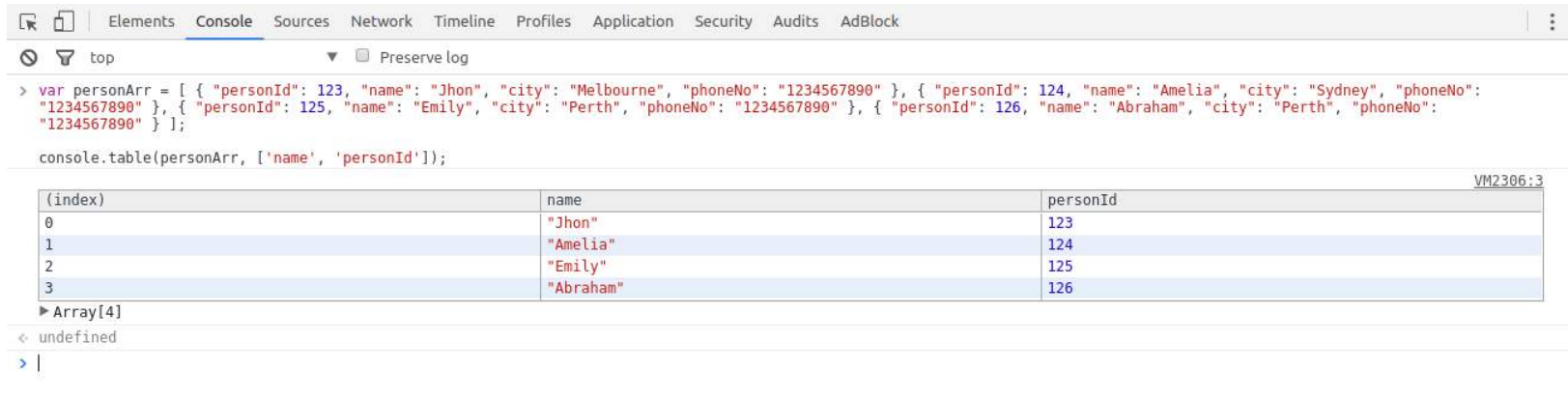
"city": "Perth",

"phoneNo": "1234567890"

}

];

console.table(personArr, ['name', 'personId']);



**Displaying objects**

var myObject = {

"foo":{

"bar":"data"

}

};

console.dir(myObject);

console.dirxml(myObject);

console.log(myObject);

**Strings**

**Basic**

Template Literals (backticks) `hello`.

var hello = "Hello";

var helloW = `Hello World`;

**Strings can be created from other types using the String() function.**

var intString = String(32); // "32"

var booleanString = String(true); // "true"

var nullString = String(null); // "null"

var intString = (5232).toString(); // "5232"

var booleanString = (false).toString(); // "false"

var objString = ({}).toString(); // "[object Object]"

**Strings also can be created by using String.fromCharCode method.**

String.fromCharCode(104,101,108,108,111) //"hello"

**Creating a String object using new keyword is allowed, but is not recommended as it behaves like Objects unlike primitive strings.**

var objectString = new String("Yes, I am a String object");

typeof objectString; //"object"

typeof objectString.valueOf(); //"string"

**Concatenating Strings**

**String concatenation can be done with the + concatenation operator, or with the built-in concat() method on the String object prototype.**

var foo = "Foo";

var bar = "Bar";

console.log(foo + bar); // => "FooBar"

console.log(foo + " " + bar); // => "Foo Bar"

foo.concat(bar) // => "FooBar"

"a".concat("b", " ", "d") // => "ab d"

**Strings can be concatenated with non-string variables but will type-convert the non-string variables into strings.**

var string = "string";

var number = 32;

var boolean = true;

console.log(string + number + boolean); // "string32true"

**String Templates**

**Strings can be created using template literals (backticks)**

`hello`.

var greeting = `Hello`;

**With template literals, you can do string interpolation using ${variable} inside template literals:**

var place = `World`;

var greet = `Hello ${place}!`

console.log(greet); // "Hello World!"

You can use String.raw to get backslashes to be in the string without modification.

`a\\b` // = a\b

String.raw`a\\b` // = a\\b

**Includers, startWith, endWith**

The **includes** method returns true if the string includes the specified substring, otherwise it returns false.

The **startsWith** method returns true if the string starts with the specified substring, otherwise it returns false.

The **endsWith** method returns true if the string ends with the specified substring, otherwise it returns false.

let str = "Hello World";

console.log(str.includes("Hello")); // true

console.log(str.startsWith("Hello")); // true

console.log(str.endsWith("World")); // true

console.log(str.includes("harry")); // false

console.log(str.startsWith("new")); // false

console.log(str.endsWith("new")); // false

**Splitting a string into an array**

**Use .split to go from strings to an array of the split substrings:**

var s = "one, two, three, four, five"

s.split(", "); // ["one", "two", "three", "four", "five"]

**Use the array method .join to go back to a string:**

s.split(", ").join("--"); // "one--two--three--four--five"

**Slice**

.slice()

.indexOf("o")

.lastIndexOf("o")

.includes("Hello")

.replace( "Hello", "Bye" );

.toUpperCase());

.toLowerCase());

**String to other datatype**

parseFloat()

parseInt()

**DOM API**

**document.getElement**

**ById("para");**

**document.getElementByClassName("para");**

**document.getElementByTagName("para");**

**document.getElementByName("para");**

**document.querySelectorAll(‘’)[]**

**document.querySelector(‘’)**

**element.matches(‘’)**

**element.closest(‘’)**

**element.contains(‘’)**

**console.log(document.getElementsByTagName('span')[0]);**

**console.dir(document.getElementsByTagName('span')[0]);**

**Changing text**

console.log( document.getElementById("para").innerHTML ) **or**

console.log( para.innerHTML )

para.innerHTML = "<h1 id=\"para\">Hey I am changed</h1>";

para.innerHTML = "Hey I am changed";

console.log( document.getElementById("para").outerHTML ) **or**

console.log( para.outerHTML )

para.outerHTML = "<h1 id=\"para\">Hey I am changed</h1>";

console.log( document.getElementById("para").textContent ) **or**

console.log( para.textContent )

para.textContent = "Hii";

**Hiding element**

document.getElementsByTagName('div')[0].hidden = true;

document.getElementsByTagName('div')[0].hidden = false;

**Attribute methods**

//Code

<div id="first" class="hey" data-game="mario" data-player="harry">

        Hey I am first container

    </div>

//js

let first = document.getElementById("first");

let a = first.getAttribute("class")

console.log(a) //hey

console.log(first.hasAttribute("class")) //true

console.log(first.hasAttribute("style")) //false

// first.setAttribute("hidden", "true") //hidden property is set to true and

Added into that element

first.setAttribute("class", "true sachin") //class=”true sachin”

first.removeAttribute("class") //while class attribute is removed

console.log(first.attributes) //all attributes are displayed with all

details in it

console.log(first.attributes[0].nodeValue) //first.attributes[0] = id

//nodevalue of id is first

**data-\* (dataset property) reserved for programmers use**

console.log(first.dataset) // *DOMStringMap {game:*

*'mario', player: 'harry'}*

// console.log(first.dataset.game) //mario

// console.log(first.dataset.player) //harry

**Insert element inside another element (body/div/etc.)**

<body>

    <h1>Hello world</h1>

    <div id="div1"></div>

</body>

var bodymain = document.body;

var element = document.createElement("p");

element.textContent = "hello world 2";

var div = document.getElementById("div1");

var element1 = document.createElement("p");

element1.textContent = "hello world 3";

bodymain.appendChild(element);

div.appendChild(element1);

<body>

<p> hello world 2 </p>

    <h1>Hello world</h1>

    <div id="div1">

<p> hello world 3 </p>

</div>

</body>

node.append(element) //add at end of node

node.prepend(element) //add at beginning of node

node.before(element) //Insert before node

node.after(element) //insert after node

node.repalceWith(element) //replace with given node

**Insert element adjacent to another element (using adjacent )**

//main code

<div id="first" style=" border: 1px solid;">

Hey I am first

</div>

//js

first.insertAdjacentHTML('beforebegin', '<div class="test">beforebegin</div>');

first.insertAdjacentHTML('beforeend', '<div class="test">beforeend</div>');

first.insertAdjacentHTML('afterbegin', '<div class="test">afterbegin</div>');

first.insertAdjacentHTML('afterend', '<div class="test">afterend</div>');

// first.remove()

//console code

<div class="test">beforebegin</div>

**<div id="first" style=" border: 1px solid;">**

<div class="test">afterbegin</div>

**Hey I am first**

<div class="test">beforeend</div>

**</div>**

**// first.remove() //**removes whole div

**className and classList**

**before**

<div id="first" class="yellow text-dark">

<span>Hey I am good</span>

</div>

first.className = "text-dark blue" //replaces all existing classes with given

**After**

<div id="first" class=" text-dark blue ">

<span>Hey I am good</span>

</div>

**classList**

// first.classList.add("red")

// first.classList.remove("red")

// first.classList.contains("red") //true-false

// first.classList.toggle("red") //add if not there and remove if exist there

**Image file**

If you already have an image file containing the desired text and have it placed on a server, you can add the URL of the image and then add the image to the document as follows: var

img = new Image();

img.src = 'https://i.ytimg.com/vi/zecueq-mo4M/maxresdefault.jpg';

document.body.appendChild(img);

**SetTimout and SetInterval**

**SetTimeout**

#1

const sum = (a, b, c) => {

  console.log("Yes I am running " + (a + b + c))

  a + b

}

//call

setTimeout(sum, 5000, 1, 2, 7)

#2

// setInterval(function() {

//   alert("setinterval")

// }, 5000)

**//To remove set timeout**

// let a = setTimeout(function() {

//   alert("I am inside of settimeout")

// }, 5000)

 clearTimeout(a)

**SetInterval**

// setInterval(function() {

//   alert("setinterval")

// }, 5000)

**Mouse Event**

For best notes and examples check this <https://www.w3schools.com/jsref/obj_mouseevent.asp>

* onclick
* oncontextmenu //right click
* ondblclick
* onmousedown
* onmouseenter
* onmouseleave
* onmousemove
* onmouseout
* onmouseover
* onmouseup

**Keybord Event**

<https://www.w3schools.com/jsref/obj_keyboardevent.asp>

* onkeydown
* onkey-press
* onkeyup

**Event listener**

**addEventListner**

*Var element = document.getElementById(“button”);*

*element*.addEventListener("click", function(){ alert("Hello World!"); });

**removeEventListner**

*element*.addEventListener("click", function(){ alert("Good bye!"); });

**Callback Function**

Function loadScript (src, callvack) {

Let script = document.createElement(‘script’);

Script.src = src;

Script.onload = ()=> callback (script);

Document.head.append (script);

}

loadScript ( ‘https://cdn.hy.com’ ,(script) => {

alert(‘script loaded’);

alert(‘script.src’)

});

**Another example**

function loadScript(src, callback) { **//( link , goodmorning )**  **( Step 2 )**

var script = document.createElement("script");

script.src = src;

script.onload = function() {

console.log("Loaded script with SRC: " + src)

callback(null, src); **// Calls step 3**

}

script.onerror = function() {

console.log("Error loading script with SRC: " + src);

callback(new Error("Src got some error")) **//Calls step 3**

}

document.body.appendChild(script);

}

function goodmorning(error, src) { **( Step 3 )**

if (error) {

console.log(error)

sendEmergencyMessageToCeo();

return

}

alert('Good morning' + src);

}

loadScript("https://cdn.jsdelivr.net/npm/bootstrap@5.2.1/dist/js/bootstrap.bundle.min.js", goodmorning) **( Step 1 )**

**Promise**

**const** promise = **new** Promise((resolve, reject) => {

*// Perform some work (possibly asynchronous)*

*// ...*

**if** (*/\* Work has successfully finished and produced "value" \*/*) {

resolve(value);

} **else** {

*// Something went wrong because of "reason"*

*// The reason is traditionally an Error object, although*

*// this is not required or enforced.*

**let** reason = **new** Error(message);

reject(reason);

*// Throwing an error also rejects the promise.*

**throw** reason;

}

});

The then and **catch** methods can be used to attach fulfillment and rejection callbacks:

promise.then(value => {

*// Work has completed successfully,*

*// promise has been fulfilled with "value"*

}).**catch**(reason => {

*// Something went wrong,*

*// promise has been rejected with "reason"*

});

**Eg.**

let promise = new Promise(function(resolve, reject) {

alert("Hello")

resolve(56)

})

console.log("Hello One")

setTimeout(function() {

console.log("Hello Two in 2 seconds")

}, 2000)

console.log("My name is " + "Hello Three")

console.log(promise)

**.then .catch**

let p1 = new Promise((resolve, reject) => {

console.log("Promise is pending")

setTimeout(() => {

// console.log("I am a promise and I am resolved")

resolve(true)

}, 5000)

})

let p2 = new Promise((resolve, reject) => {

console.log("Promise is pending")

setTimeout(() => {

// console.log("I am a promise and I am rejected")

reject(new Error("I am an error"))

}, 5000)

})

// To get the value

p1.then((value) => {

console.log(value)

})

// To catch the errors

// p2.then((value)=>{

console.log(value)

}).catch((error) => {

// console.log("some error occurred in p2")

// console.log(error);

// })

p2.then((value)=>{

console.log(value)

},(error)=>{

console.log(error)

})

**Loadscript**

const loadScript = (src) => {

return new Promise((resolve, reject) => {

let script = document.createElement("script")

script.type = "text/javascript"

script.src = src

document.body.appendChild(script)

script.onload = (script) => {

resolve("Script has been loaded sir")

// resolve(5);

}

script.onerror = () => { reject(0) }

})

}

let p1 = loadScript("https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/js/bootstrap.bundle.min.js")

p1.then((value) => {

console.log(value)

return loadScript("https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js")

}).then((value) => {

console.log("Second script ready")

console.log(value)

reject(5);

}).catch((error) => {

console.log("We are sorry but we are having problems loading this script")

console.log(error)

})

**Promise chaining**

let p1 = new Promise((resolve, reject) => {

setTimeout(() => {

console.log("Resolved after 2 seconds")

resolve(56)

}, 2000)

})

p1.then((value) => {

console.log(value)

return new Promise((resolve, reject) => {

setTimeout(() => { resolve("Promise 2") }, 2000)

})

}).then((value) => {

console.log("We are done")

return 2

}).then((value)=>{

console.log("Now we are pakka done")

})

**Pomise API**

Promise. all ( )

Promise. a11Sett1ed( )

Promise. any ( )

Promise. prototype. catch ( )

Promise. prototype. finally ( )

Promise. race( )

Promise. reject()

Promise. resolve()

let p1 = new Promise((resolve, reject) => {

setTimeout(() => {

resolve("Value 1");

}, 5000);

});

let p2 = new Promise((resolve, reject) => {

setTimeout(() => {

// resolve("Value 2");

reject(new Error("Error"));

}, 2000);

});

let p3 = new Promise((resolve, reject) => {

setTimeout(() => {

// resolve("Value 3");

// reject(new Error("Error"));

}, 3000);

});

// let promise\_all = Promise.all([p1, p2, p3])

// let promise\_all = Promise.allSettled([p1, p2, p3])

// let promise\_all = Promise.race([p1, p2, p3])

// let promise\_all = Promise.resolve(6)

// let promise\_all = Promise.reject(new Error("Hey"))

// promise\_all.then((value) => {

// console.log(value)

// })

**Async/ Await**

The keyword async before a function makes the function return a promise:

**Example**

async function myFunction() {  
  return "Hello";  
}

Is the same as:

function myFunction() {  
  return Promise.resolve("Hello");  
}

**Example**

async function myFunction() {  
  return "Hello";  
}  
myFunction().then(  
  function(value) {myDisplayer(value);},  
  function(error) {myDisplayer(error);}  
);

**Await**

async function myDisplay() {  
  let myPromise = new Promise(function(resolve) {  
    setTimeout(function() {resolve("I love You !!");}, 3000);  
  });  
  document.getElementById("demo").innerHTML = await myPromise;  
}  
  
myDisplay();