**JavaScript youtube links:**

<https://www.youtube.com/watch?v=lGmRnu--iU8&t=13347s>

**content:**

**Block scope vs function scope**

**Default parameters**

**Rest parameters**

**Param destructuring**

**callback functions**

**Function returning function**

**Array Methods (forEach method**

**map method**

**filter method**

**reduce method**

**sort method**

**find method**

**every method**

**some method**

**fill method**

**splice method)**

**Iterables, array like objects**

**sets**

**map data structure**

clone using Object.assign

optional chaining

Create your own methods

This keyword

call, apply, bind methods

small warning

arrow functions and this

short syntax

create functions to create multiple objects

store methods in different object

solution using object.create

What is prototype

Use prototype

New keyword

hasOwnProperty

more about prototype

class keyword

class practice and extends keyword

super keyword

same method in base class getters and setters

static methods and properties

**JAVASCRIPT**

Java Script is a dynamic interpreted programing language

It allows client-side scripting to create completely dynamic web applications and websites

 Java Script is a scripting language that enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else.

What is JavaScript?

* It defines all the actions to be done on a web page.
* It is object-oriented programming language which uses JIT complier
* It is everywhere all the web browsers will be installed with it
* JavaScript ranges from web development and mobile application etc.
* JavaScript is very easy and compatible with html and CSS

Features of JavaScript?

* JavaScript is not a proper programming language it’s a scripted language
* In c and c++ we need a compiler but In JS we don’t need a compiler to run
* It is lightly weighted scripted language. Coz, we need not to tell the datatype
* It’s a client and server-side technology (client side means it will run on their Pc whereas server side means it will run on web
* It’s a user validation
* It’s a platform independent language
* It’s an object-oriented scripting language,
* It is case sensitive

**Applications in JS:**

* Web development
* Mobile application
* Web application
* Games
* Smart Watches
* Client-side Validation

What is JIT (just in time) compiler

Simple words it’s the combination of complier and interpreter

Identifies the hot or warm components of the code Ex: Repetitive code

Transforms the components into the machine code

Optimizes the machine code

Hot swap the previous implementation of the code

**History of the JavaScript:**

JavaScript founder was Brendan Eich in just 10 days working for netscape

He developed JS in 1995

He was developed first Js engine spider monkey and it is still used by the Mozilla Firefox

Js Named was changed to mocha and live script but due to some trademarks it’s still continuing as JavaScript

**Variables JS:**

Two types of languages

1. Statically typed language
2. Dynamically typed language

1.Statically typed Languages: Where each variable and expression type are already known at compile time and when it declared it cannot be change to other data type

Ex: C, C++, JAVA.

2.Dynamically typed language: These languages can receive any type of data type. We don’t need to specify the data type. In JS it can understands which datatype need to be specified. We can re-assign the variable to any other value.

Ex: python, Ruby, JS.

Variable: Variables in JS are containers which holds the reusable data. It is a simple name of the storage location

scope types of variables:

Local scope variables: if it declared inside a block or function. It is accessible with the function or the block

Global scope variables: it is declared outside the function and block. It can be accessible from any function

We can declare the variable by using three types of keywords

1.var: no block {} scope is created, can be redeclared

2.let : Let is not accessible outside the block in which its declared and it will not able to redeclare the variable

3.const: It is fixed we can’t change the value of the const variable

Naming convention:

1.Snake: Here, we use underscore (\_) like

Ex: first\_name

2.kebab: Here, we use Hyphen (-)

Ex: first-name

3.Camel: Here, we use Second word capital letter

Ex: firstName

**Datatypes:**

Every JS variable holds the datatype

JS has two types of datatypes:

Primitive datatypes:

1. String

In string we have a special character like

* \” content\” we can use this as double quotes
* \n the content after this will be in next line
* \t the content after this will have a tab space
* \b the content after this will be deleted

**Methods of the string:**

If we want to add two strings, we will use + operator to concatenate the strings

Method 1 : using + operator

let firstName="sharath";

let lastName="Chintakindi";

console.log(firstName+" " +lastName);

o/p: sharath chintakindi

Method 2 : by using template literal(its new version ):

let fisrtName="sharath";

let lastName="chintakindi";

console.log(`My name is : ${fisrtName} ${lastName}`);

o/p: sharath chintakindi

when we want to get a letter or character from the string (here index start from “0”)

let fisrtName="sharath";

console.log(fisrtName[2]);

o/p: a

To upper case and lower case we use these methods

toLocaleUpperCase: it will convert into uppercase

toLocaleLowerCase: it will convert into lowercase

let fisrtName="sharath";

let lastName="CHINTAKINDI";

console.log(fisrtName.toLocaleUpperCase());

console.log(lastName.toLocaleLowerCase ());

Indexof(): it will in which index the letter or character is there.

let fisrtName="sharath";

let lastName="CHINTAKINDI";

console.log(firstName.indexof(“a”);

o/p: 3

lastindexof(): it will show the last index of the character

let fisrtName="sharath";

let lastName="CHINTAKINDI";

console.log(lastName.lastIndexof(“I”);

o/p:10

includes: it tells us that it is existed in the string or not

let fisrtName="sharath";

let lastName="CHINTAKINDI";

console.log(lastName.includes(“CHINTAKINDI”);

o/p:true

trim: trim will trim the trailing and leading spaces of the string

let fisrtName=" sharath ";

console.log(firstName.trim());

o/p: sharath // without spaces

slice: it will extract the particular part of the string

let fisrtName="sharath";

console.log(“firstName.slice(0,4); // it will excludes the last index ie.. 3

o/p: sha

1. Boolean
2. Number
3. Undefined
4. Null

Composite types or non-primitive types:

1. Object
2. Array
3. RegExp

JS is loosely Typed programming language

JS Is dynamically typed scripting language

**Loose equality (==) v/s strict equality operator(===)**

**Loosely equality(==):**  In this case its only checks for the value and returns true

let firstName="sharath";

console.log(firstName==sharath);

o/p: true

let score=22;

console.log(score=="22");

o/p:true

In this case it will check only for values, it will not check for datatypes

**strict equality operator(===):**  In this case it will check for both the values and datatypes as well if its true it will returns true

let score=22;

console.log(score==="22");

o/p: false

In this case it will check for values and datatypes

Arrays:

let cars =["bmw","audi","benz"];

console.log(cars);

let cars1=["lambo","byw"];

console.log(cars.concat(cars1));

o/p: {bmw,audi,benz,lambo,byw}

**Type conversions and type coercion**: Converting one datatype to another datatype

* Numbers to string
* String to number
* Dates to numbers
* Numbers to date
* Boolean to number
* Number to Boolean

**For Numbers:** to convert the string to numbers

Syntax: let x=number (“123”)

**Methods:**

* **Number():** Returns the number
* **parseFloat():**  Parse a string and returns a floating point number
* **parseInt():**  Parse a string and returns an integer

**For Strings:**  to converts the numbers to string

Syntax: let string=string (1)

We can use tostring() also

**Methods:**

* **toExponentional ():** Returns a string, with a number rounded and written by using exponential notation
* **toFixed ():** Returns string, with a number rounded and written with specified number of decimal’s
* **toPrecision ():** Returns a string with a number written with a specified length

**Operators**

Operators are the symbols which are used to assign, Compare, Perform the arithmetic operations

The variables are called as operands

1. Arithmetic operators: Used to do Arithmetic operations on numbers

Ex: +, -, \*, /, \*\*, %, ++, --

++ (variable name) is called as pre increment

(variable name) ++ is called as post increment

Difference between the post and pre increment

In post increment the value will be assigned and next it will be incremented

In pre increment it will be incremented first and then next assigned the value

1. Assignment: assign values to JS variables

|  |  |  |
| --- | --- | --- |
| Operator | Example | Same as |
| = | X=Y | X=Y |
| += | X+=Y | X=X+Y |
| -= | X-=Y | X=X-Y |
| \*= | X\*=Y | X=X\*Y |
| /= | X/=Y | X=X/Y |
| %= | X%=Y | X=X%Y |
| \*\*= | X\*\*=Y | X=X\*\*Y |

1. Comparison: Used to compare the operators used to tests for true or false

|  |  |
| --- | --- |
| Operator | Description |
| == | Equals to |
| === | Equals value and equals type |
| != | Not equal |
| !== | Not equal value or not equal type |
| > | Greater than |
| < | Less than |
| >= | Greater than or equals to |
| <= | Less than or equals to |
| ? | Ternary operator |

1. String: All comparison operators used as string operators

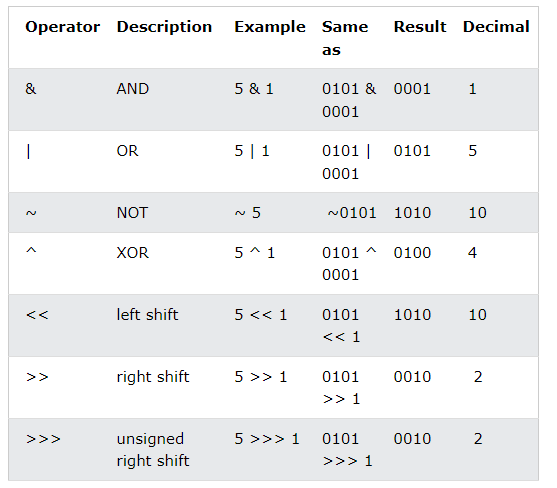
“+” can also to be used to add (concatenate) strings

1. Logical operators:

|  |  |
| --- | --- |
| Operators | Description |
| && | Logical and |
| || | Logical or |
| ! | Logical not |

Bitwise operators: Bit operators work on 32 bits numbers.

Any numeric operand in the operation is converted into a 32-bit number. The result is converted back to a JavaScript number



1. Ternary or conditional operators: Js also contains the conditional operators that assigns the value based on some condition

Syntax: variable name=(condition)? Value1:value2

Var x=(age<18)? “too young”: “too old”;

1. Type operators: used to determine the type of data being stored in the variable

|  |  |
| --- | --- |
| Operators | Description |
| Typeof | Returns the type of the variable |
| instanceof | Returns true if an object is an instance of an object type |

**Control statements:**

Js control statements are also known as the control structures and flow control statements are the statements which decides the execution of the program.

Typically, the JS code execution begins from the first line to the last line of the code. We can reiterate based on some criteria. This functionality is provided by the control statements in JS.

start

<script>

Code statement1;

Code statement2;

Code statement3;

Code statement4;

Code statement5;

Code statement6;

</script>

End

JS control statements

|

------------------------------------------------------------------------------------------------------------

Conditional control Break Continue lopping/iterative

Statements Statements control statements

| |

1.If-else Statements 1.For Loop

2.switch Statements 2. While Loop

3.Dowhile Loop

4.For-in Loop

**If else statements:**  it is used to execute the code whether condition is true or false

Three statements:

1.If statement

2.If else statement

3. if else if statement

**1. if statement:** It runs only if the statement is true.

Syntax:

If(expression)

{

Content to be executed;

}

**2.If-else statement:** It will be executed if the condition is true or false

Syntax:

If (expression)

{

Content to be executed if it is true;

}

Else {

content to be executed if it’s false;

}

3.**if- else if statement:** it executes the content only if expression is False.

Syntax:

If (expression)

{

Content to be executed if it is true;

}

Else if{

content to be executed if it’s true;

}

Else if

{

Content to be executed if the condition is true.

}

**Switch statements**: It can be used for decision making purposes in some cases, using switch cases is more efficient than if else statements.

switch(expression) {

case value1:

 code to be executed;

 break;

case value2:

 code to be executed;

 break;

......

default:

 code to be executed if above values are not matched;

}

Here, Break statement can be used to break the sequence

In JS switch case we can give numbers, strings, Boolean values unlike C, C++

We can give any logical expression in the switch case in JS

Default statement is an optional statement

**Iterative or looping statements:**  When we want to do the task for repetitive or more times. We can use the iterative or looping statements.

**For loop:** when we know how many times you need to repeat the block of the code.

Three steps:

1. Initialization Statement
2. Condition statement
3. Increment statement

Syntax: for (initialization; condition; increment)

{

    code to be executed;

}

**While loop:** A while is used when you don’t know how many times you need to repeat a block of code, but you know the condition that will end the loop

In while we will check for the conditions and execute the block

Syntax:

While(condition)

{

Code to be executed;

}

**Do-while loop:** A Do while loop is similar to a while loop, but the block of code is executed at least once, even if the condition is false.

In do while we will execute the block first and check for the condition next.

do {

Code to be executed;

}

While(condition);

**While loop and for loop” which can we use”**

For loop: If we know the starting and ending point, we will use for loop

While loop: In while loop we don’t know the starting and ending point, but we know the condition

**For-in loop**: A for in loop is used to loop through the properties of an object

Ex: let user\_name{firstname:”sharath”,lastname:”chintakindi”,

Education:{btech:2018,};

For(let(key) in user\_name);

{

Console.log(key,user\_name,username[key]);

**Functions In JS:**  A function is a set of statement takes the input. It is used for the repetitive code. Instead of writing the code same again and again we can call that function

It has three things

1. Function declaration
2. Function definition
3. Function calling

Advantages of the function:

* Code reusability
* Less coding

Here we can create a function by two methods

1. By creating the name of the function

Function functionName(){

Console.log(“sharath”);

}

functionName();

1. By creating the variable name of the function

Let variableName = function(parameters){

Console.log(`hellow world ${parameters});

}

variableName(values);

**Syntax:**

function greet () // declaring the function

{

Console.log(“Hello World”)

}

greet(); //calling a function

Here, we have to call the function

Ex2: using return statement

Function f()

{

Return “sharath”;}

Let name=f();

Console.log(f);

When return statement is used in a function body, the execution of the function is stopped

In function we can pass the value to the function

Passing a value:

Function f(user)

{

Return `sharath ${user}`;} // Here, we have to use backticks(`)

Let user=”sai”;

Let name=f(user);

Console.log(f);

**Function expression:**

Here, num1 and num2 are called as parameters and 5,6 are called as aruguments

Let add(num1,num2);

{

Return num1+num2;

}

Let result=add(5,6);

Console.log(result);

let inviation = function(name,place) // function declaration

{

console.log(`Hey! ${name} welcome, to the college at the ${place}`);

}

inviation(“sharath”,”Hyderabad”); // function calling

o/p: Hey! Sharath welcome, to the college at the Hyderabad

**Arrow Function:**  Arrow functions were introduced in ES6

It allows us to write function faster because no need to write the function keyword

No need of using the parenthesis () in case of single parameter

No need use the curly {} in case we are using single statement

No need to write the return statement

Ex: let greet= () => // Here, we won’t write the function name

{ return “sharath”; }

Console.log(greet());

If the function has only one statement, and the statement returns a value. You can remove the brackets and the return keyword.

Ex: let greet=() => “sharath”;

**Higher order function- callbacks & returning functions:**  see afterwards

**IIFE(immediately invoked function expression):**

(Function(name){ // declaring the function

Console.log(“name”)})

(“sharath”); //calling the function

**setTimeOut**: run the function “once” after the set out “interval” time and returns the result of line (we can pass the arugments)

function greet(){

console.log(“Welcome Home”);

}

setTimeOut(greet,5000); // here, 5000ms=5secs (only in ms)

function greet(name){

console.log(`Welcome Home ${name}`);

}

setTimeOut(greet,5000,”sharath”); // here, 5000ms=5secs (only in ms)

**setInterval**: runs function repeatedly, starting after the interval of time, then repeating

function greet(name){

console.log(`welcome to my family ${name});

}

setInterval(greet,1000,”sharath”);

o/p: welcome to my family sharath // it will be executed infinitely

**Template literal:** Template literal in ES6 provides new features to create a string that gives more control over dynamic strings

String is created by using single quotes (‘) and double quotes (“). Template literals is created by using backtick (`)

**Hositing in JavaScript:**  imp topic for interview questions

**Object**: Object is a collection of real entities or collection of named values

In JS Everything is an object

* Boolean is object (If we use new keyword)
* Numbers is object (If we use new keyword)
* String is object (If we use new keyword)
* Dates are always Objects
* Math’s are always Objects
* Arrays are always Objects
* Functions are always Objects
* Objects are always Objects

All JS values, except primitive are objects

A primitive value is a value that has no properties or methods

* 1. is a primitive value

primitive values are immutable they can be changed

Objects values are written as “name:value pairs”

Objects can be created by using object literal

Syntax:

Let x={first\_name:’sharath’,last\_name:’chintakindi’};

The name:values pairs in JavaScript objects are called as properties

Accessing the objects properties:

Two ways

* By using . : objectname.propertyname
* By using [] : objectname[“propertyname”

Complex objects /delete object/ ”?”

“?” Is used if we don’t know if the property is available in objects

Ex: let x={name:’sharath” age:26,

Education{ btech:”EEE”};

Console.log(x.education?.length);

Here, Length is used to know the length of the string

Delete is a keyword which is used to delete the property of an object

Ex: let x={name:’sharath” age:26,

Education{ btech:”EEE”};

Delete(x.age);

Console.log(x.education?.length);

**Methods in Functions**: It is nothing but the object property (key) holding the function is “value”

**This keyword**: This is a keyword refers to an object of the same block

**Arrays:**

JS array is a single variable that is used to store elements of different data types. Js array are zero indexed

**Declaration of array:**

1. Creating an array using array literal:

syntax: let arrayName=[ “value1”,” value2”,” value3”,” value4”---];

let arrayName=[ “html”,”CSS”,”JS”,”React”];

1. Creating an array by using new keyword:

Syntax: let arrayName=new array();

let arrayName=new array (“html”,”CSS”,”JS”,”React”);

Converting array into the string: by using arrayname.tostring();

Let color=[“Blue”,”red”,”Green”];

Console.log(colors.tostring());

o/p: blue,red,green

**Methods to add and delete the values in Arrays**

1. Push: used to add the value into the array.

Let color=[“Blue”,”red”,”Green”];

Color.push=(“Yellow”);

Console.log(colors);

1. Pop : used to delete a value from an array and gives the result as undefined. The last value will be deleted by default

Let color=[“Blue”,”red”,”Green”];

Color.pop=();

Console.log(colors);

3.shift and unshift Is just like push and pop in arrays

**3.for Each method of array:**  Used for the array values and it will be displayed each value in next line

Let car=[“bmw”,”audi”,”benz”]

Car.forEach(function(element){

Console.log(car(element)})

o/p: bmw

audi

benz

let array=[“1”,”2”];

array.forEach(element){

console.log( (element);})

**objects inside the array**

letblocklist=[{username:”sharath”, password:”123456”},{newUsername:”chintakindi” password:”09876”}

console.log(blocklist);

for (var i=0;i<blocklist.length;i++){

console.log(blocklist[i]);

}

**Math Object**: Inbuilt in object in Javascript

Floor,ceil,random,trunc,round

Function: call and apply method

We can manually set the value of “this” keyword using call and apply

// objects /arrays how reference is passed to variable

Pass by value and pass by reference

let array=[1,2,3,4,5];

let copy=array;

//copy[7]=10;

console.log("original one" ,array);

console.log("copied one",copy);

o/p: original one is : [1,2,3,4,5<2 empyty>,10]

copied one is : [1,2,3,4,5<2 empyty>,10]

// here both are changed when we are changing the copy copied array to overcome this we can use spread operator

console.log("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

let array1=[...array]; //using the spread operator by square brackets

array1[8]=10;

array1.shift();

console.log("orginal one ",array);

console.log("copy " ,array1);

o/p: original one is : [1,2,3,4,5,10]

copied one is : [1,2,3,4,5<2 empyty>,10]

**Inputs from the user:**

**HTML Events:**  Html Events are the Things that happens to the Html Elements

When JS is used in the html pages then these events will react

An Html event can do something the browser does and something the user does

Common Html events:

Onchange : An HTML element has been changed

Onclick: When the user clicks an html element it will change according to the code written

Onmouseover: the user moves over the cursor to an html element

Onmouseout : : the user moves out the cursor to an html element

Onkeydown: The user pushes a keyword key

Onload: The browser has finished loading the page

**Dom Manipulations**

What is DOM in Js?

With the html DOM JS can access to change all the html elements and html tags

What is DOM (document object model)

The DOM is w3c (world wide web consortium)

The DOM defines the standard access to documents

When a browser opens a web page. It will create the DOM (document object model) of the page

The w3c DOM is a platform and language - neutral interface that allows the program and to script dynamically access and update the content, structure and style of the document



kIt has three parts

1.Core DOM: standard for all the documents

2.xml DOM: standard for all the xml documents

3.HTML DOM: Standard for all the html documents

Html DOM: it’s a standard object model and programming interface for html

The HTML elements as objects.

In simple words the html DOM is nothing but standard how to get, update, delete or add the html elements

In the DOM, All the html elements are defined as “Objects”

In DOM we will deal with three nodes

Hierarchy of the nodes are:

Element node

Attribute node

Text node

Two ways to add the JS

1.Embed JS in html (we can write the JS with the help of <script> tags)

2. External JS File

**Dom Methods**

getElementById method:

selectedIndex property:

select option: open the javascript folder 🡪 program name is (Quiz.html) and (Quiz.js)

The selectedIndex property in HTML DOM is used to set or return the index of the selected option in a drop-down list. The index of the drop-down list generally starts with 0 and returns -1 if no option is selected

getElementByTagName:

It returns all the elements in a specified tag name. It returns in HTML collection

A html Collection is an array-like collection (list) of html elements

It starts from the index “0”

**QuerySelector():**

The querySelector() gives the value of first html element that matches the css selector

**QuerySelectorAll():**

The querySelectorAll() gives all the values of html elements that matches the css selector’s

Ex: <!DOCTYPE html> //html file

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="Dom manipulations.js" type="text/javascript"></script>

</head>

<body>

<h1>Let's code sharath</h1>

<p>Lorem ipsum dolor sit amet </p>

<p class="first">Lorem ipsum dolor sit.</p>

<h1 id="second">Lorem, ipsum.</h1>

</body>

</html>

**Js file**

Using querySelector();

Let result=document.querySelector(‘p’); //by calling with the html element

Console.log(result);

o/p: <p>Lorem ipsum dolor sit amet </p>

Using querySelectorAll():

Let result=document.querySelectorAll(‘p’);

Console.log(result);

o/p: <p>Lorem ipsum dolor sit amet </p>

<p class="first">Lorem ipsum dolor sit.</p>

Access class elements by using the queryselector():

Let result=document.querySelector(‘.first);

Console.log(result);

o/p: <p class="first">Lorem ipsum dolor sit.</p>

Access Id elements in html by using queryselector():

Let result=document.querySelectorAll(‘#second’);

Console.log(result);

o/p: <h1 id="second">Lorem, ipsum.</h1>

**Updating and changing the content:**

**InnerText :**

* when we use this innerText it will ignore the spaces display the output
* Retrieve and displays the plain text

**innerHtml:**

* when we use this innerhtml it will not ignore the spaces display the output
* retrieve and set content in the html format with tags

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<!-- <script src="Dom manipulations.js" type="text/javascript"></script> -->

</head>

<body>

<h1>Let's code sharath</h1>

<p> Lorem ipsum dolor sit amet </p>

<p class="first">Lorem ipsum dolor sit.</p>

<h1 id="second"> Lorem, ipsum.</h1>

<script>

let x= document.querySelector("p");

console.log(x.innerText);

let y=document.querySelector("#second");

console.log(y.innerHTML);

</script>

</body>

</html>

Example 2:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<h1>Let's code sharath</h1>

<p> Lorem ipsum dolor sit amet </p>

<p class="first">Lorem ipsum dolor sit.</p>

<h1 id="second"> Lorem, ipsum.</h1>

<script>

let x = document.querySelector(".first");

x.innerText +="<p> Hello world</p>"; // here + used to concatenate with the first paragraph of the code( <p class="first">Lorem ipsum dolor sit.</p>

)

let y=document.querySelector("#second");

y.innerHTML+="<p> hello sharath</p>";

</script>

</body>

</html>

**Getting and setting the attribute of the element:**

**Two methods:**

1. getAttribute
2. setAttribute
3. getAtrribute: If we want to get the elements into the browser, we can use this
4. setAttribute: We can set the attribute values by using this

ex: Getting and setting the attribute in javascript

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

<script src="Dom manipulations.js" type="tet/javascript"></script> -->

</head>

<body>

<h1>Welcome To the coding Family</h1>

<a href="https://www.youtube.com/"> hello</a>

writing the script for js

<script>

getting the attribute

let setting = document.querySelector("a");

console.log(setting.getAttribute("href"));

setting the attribute

/ setting.setAttribute("href","https://www.youtube.com/watch?v=lI1ae4REbFM&t=4084s");

/ console.log(setting.getAttribute('href'));

setting.innerText = " JS CLass";

</script>

</body>

// </html>

**Add, Remove, replace of the HTML attributes**

**classListAdd():**

**classListRemove():**

**classListReplace();**

We can add, remove, replace the html attributes in Html elements

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Add Remove and replace</title>

</head>

<body>

<div class=”content”>

<h1>Add Remove and replace</h1>

<h2 class="first">Lorem, ipsum dolor.</h2>

<h3 class="second"> Lorem ipsum dolor sit amet.</h3>

<h4 class="third"> Lorem ipsum dolor sit amet.</h4>

</div>

<script>

// Adding the attribute

Let x=document.querySelector(“.first);

x.classList.add(“first1”); // the first attribute in h1 will change into “first1”

//remove the attribute

let y= document.querySelector(“.second”);

y.classList.remove(“second”); // The second attribute in h2 will be removed and will be empty

//Replace the attribute

Let z= document.querySelector(“.third);

z.classList.replace(“third”,”fourth”); // the third attribute value will be change into the fourth in h4 element

let p=document.querySelector(“.content);

console.log(p.children);

</script>

We can access the parent html elements by using (.children) statement

**Dom Events**

**Click event:** when we click on something with the mouse

Example:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Add Remove and replace</title>

</head>

<body>

<h1> Fruits name </h1>

<ol>

<li> apple</li>

<li> green</li>

<li> banana</li>

</ol>

<button class=”click”> submit </button>

<script>

// For just one element

Let element= document.querySelector(“click);

Element.addEventListener(“click,function()

{

Console.log(“clicked “);

}

//For More than one element we will use forEach

Let elements =document.querySelectorAll(“li”);

Elements.forEach(“click”,function(element){

Element.addEventListener(“click”,function(e){

Console.log(“clicked”); )}

Console.log(e.target); // we can print the elements //target is used for single element

e.target.style.color =”blue”;

e.target.remove(); // when we click any value its will be deleted

})

</script>

</head>

<body>

</html>

Creating and removing the element

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Add Remove and replace</title>

</head>

<body>

<h1> Fruits name </h1>

<ol>

<li> apple</li>

<li> green</li>

<li> banana</li>

</ol>

<button class=”click”> submit </button>

<script>

let allElements = document.querySelector("ol");

let button=document.querySelector(".click");

button.addEventListener("click",function(){

let li = document.createElement("li");

li.textContent=("one");

allElements.prepend(li);

})

</script>

</head>

<body>

</html>

Bubbling And Delegation: Have to learn

**Scopes in Javascript:**

**Local scope:**

* The variables which declared inside the block is known as local scope
* It can be accessed inside the block only

**Global scope:**

* The variables which declared outside the block is known as global scope
* It can be accessed anywhere in the program

**Function scope:**

* Var is function scope and it can be accessed anywhere in the program

**Block scope:**

* Let and const is block scope
* Before ES6, JavaScript had only two ie… global scope and function scope
* Es6 introduced two extra keywords: let and const
* Variables declared inside a { } block cannot be accessed from outside the block

**Functions Default and rest parameters**

**Default parameter’s:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>JavaScript Classes</title>

<script src="index.js"></script>

</head>

<body>

<h1 class="Hello">JavaScript Classes</h1>

<h3 class="Hello2">JavaScript Classes</h3>

<h4 class="Hello1">JavaScript Classes nahdsa</h4>

</script>

</body>

</html>

Js file

{

var x =100;

}

console.log(x);

normal Parameters

function addTwo(a,b){

return a\*b;

}

console.log(addTwo(4,5));

// if we miss one parameter then we can set the default parameter by two ways

function minusTwo(a,b){

return a / b;

}

console.log(minusTwo(4)); // Here we get Nan if we don't pass the second arugment

// to over come this we use two methods

// First method "Old Method"

function minusTwo(a,b){

if(typeof b==="undefined")

{

b=2; // Here we can set any value

}

return a / b;

}

console.log(minusTwo(4));

// second Method

function minusTwo(a,b=2){ // Here, by default we gave the value

return a-b;

}

console.log(minusTwo(10));

o/p:

100

20

2

2

8

**rest parameters (…):**

The rest parameter (…) will allows a function to take all indefinite arguments as an array

Ex:

**HTML file**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>JavaScript Classes</title>

<script src="index.js"></script>

</head>

<body>

<h1 class="Hello">JavaScript Classes</h1>

<h3 class="Hello2">JavaScript Classes</h3>

<h4 class="Hello1">JavaScript Classes nahdsa</h4>

</script>

</body>

</html>

**Js file**

// Creating a function with less parameter

function num(a,b,c){

console.log(`a is : ${a}`);

console.log(`b is : ${b}`);

console.log(`c is : ${c}`);

}

// calling a function

num(1,2,3);

// Just for seperation

console.log("---------------------------");

// if we give many arugments

num(1,2,3,4,5,6); // we will get only 3 values that fit within the arugments

// Just for seperation

console.log("---------------------------");

// Instead we use Rest function parameter

function num(a,b,...c){

console.log(`a is : ${a}`);

console.log(`b is : ${b}`);

console.log(`c is : ${c}`);

}

num(1,2,3,4,5,6,8,9,10);

**Param Destructing:**

**Js file**

let person={

name:"sharath",

age:27,

gender:"male",

}

// function personDetails(obj){

// console.log(person.name);

// console.log(person.age);

// console.log(person.name);

// }

// personDetails();

function personDetails({name,age,gender})

{

console.log(name);

console.log(age);

console.log(gender);

}

personDetails(person);

o/p: Sharath

27

male

**Callback Function:** A callback function passed as an argument to another function

// creating a function

function firstFunc(name){

console.log("this is the first function");

console.log(`My name is ${name}`);

}

// second Function

function secondFunc(callback){

console.log("this is second Function");

callback("sharath");

}

secondFunc(firstFunc); //Calling function into a function

function myFirst(){

console.log("hello");

}

function mySecond(first){

console.log("sharath");

first();

}

mySecond(myFirst);

o/p: this is second function

this is the first function

my name is sharath

sharath

hello

**Function returning function:**

// creating a function

function first(){

return [1,2,3,4,5];

}

let ans = first();

console.log(ans);

// we can call function return into a function

function myFunc(){

function hello(){

return "hello Sharath";

}

return hello;

}

let ans1 = myFunc();

console.log(ans1());

**Array methods in JS:**

* **forEach()**
* **map()**
* **filter()**
* **reduce()**
* **sort()**
* **find()**
* **every()**
* **some()**
* **fill()**
* **splice**

**ForEach() method:**

* forEach() method calls a function for each element in an array
* It will not be executed for empty elements
* We can user for of loop instead of the forEach() both will get same result

Ex: let bike = ["bike1","bike2","bike3"];

bike.forEach(

function (bikesN)

{

console.log(bikesN);

}

)

let numbers = [65,44,12,4];

numbers.forEach(function(item,ind,a)

{

console.log(a[ind]= item-2,ind);

})

// creating a objects

let userN = [

{Name:"sharath",age:10,},

{Name:"sharath1",age:104,},

{Name:"sharath2",age:103,},

{Name:"sharath3",age:102,}];

userN.forEach(function(user)

{

console.log(user.Name);

})

Using For of loop:

for (let user of userN)

{

console.log(user.Name);

}

**Map() method:**

* Map() creates a new array from calling a function for every array element
* Map() will not be executed the empty elements
* Map() does not change the original array

Ex:

let array3 = [2,21,24,55];

let array4 = function(num)

{

return num\*num;

}

let square = array3.map(array4);

console.log(square);

// Using names

let userName = [

{name:"sharath",age:20},

{name:"Chintakindi",age:20},

{name:"Sai",age:20},

{name:"Nunna",age:20},

]

let userName1 = userName.map((user)=>

{ return user.name;}

)

console.log(userName1);

**Filter() method:**

* Filter() method creates a new array filled with elements that pass a test provided by a function
* Filter() method doesn’t executed for empty elements
* Filter() doesn’t change the original array

Ex:

let numbers1 = [1,2,3,4,5,6,7,8,9,10];

let isEven = function(num)

{

return num % 2===0;

}

let ans = numbers1.filter(isEven);

console.log(ans);

let numbers3 = [1,2,3,4,5,6,7,8,9,10];

let isOdd =numbers3.filter((num) =>

{

return num % 2 !==0;

}

)

console.log(isOdd);

**Reduce():**

* Reduce() method executes a reducer function for array element
* Reduce() method returns a single value; the function accumulate result
* Reduce() method does not execute the function for empty array elements
* Reduce() method doesn’t change the original array

const numbers4 = [1,2,3,4,5,6,7];

const sum = numbers4.reduce((accu,cV)=>

{

return accu+cV;

}

)

console.log(sum);

// passing intial value

let flipkart = [

{prodcut:"mobile",price:20000},

{prodcut:"mobile1",price:30000},

{prodcut:"mobile2",price:40000},

]

const total = flipkart.reduce((prodcut,value)=>

{

return prodcut + value.price;

}

,10000)

console.log(total);

**sort() method:**

* Sort() methods sort the elements
* Sort() method overwrites the original array
* Sort() sorts the elements as strings in alphabetical and ascending order

Ex:

// sort() method

let array = [20,111172,915,5,5,342627,1578];

let array2 = array.sort((a,b)=>a-b);

console.log(array2);

console.log(array2[array2.length-1]); // to get highest value

let names = ["sharath","Chintakindi","hello"];

names.sort();

console.log(names);

// price lowtToHigh and HighToLow

let products=[

{product:"mobile",price:"10000"},

{product:"bike",price:"2098561"},

{product:"laptop",price:"201"},

{product:"adapter",price:"2578199"},

{product:"headphone",price:"9000"},

{product:"mouse",price:"0961"},

];

products.sort((a,b)=>

a.price-b.price

)

products.reverse(); // it will be in ascending to descending or descending to ascending

console.log(products[products.length-1]);

**find() method:**

* Find() method returns the value of the first element that passes a test
* Find() method executes a function for each array element
* Find() method returns undefined if no elements are find
* Find() method does not execute the function for empty elements
* Find() doesn’t change the original array
* Find() it will show the first occurrence in the array elements

Ex:

// find() method

//Outside the function

let ages = [18,30,40,8];

// function ages1(age){

// return age>20;

// }

// let age2 = ages.find(ages1);

// console.log(age2);

// Inside witharraow function

let age2 = ages.find((age)=>

{

return age>20;

})

console.log(age2);

const array = [

{userId:1, userName:"sharath"},

{userId:2, userName:"shth"},

{userId:3, userName:"shah"},

{userId:5, userName:"shath"},

{userId:4, userName:"shath"},

]

const array1 = array.find((user)=>user.userId===3)

console.log(array1);

**every() method:**

* Every() method executes a function for each array element
* Every () method returns true if the function returns true for all the elements
* Every() method returns false if the function returns false for all the elements
* Every() method doesn’t execute the function for empty elements
* Every() method doesn’t change the original array

**Ex:**

let array = [2,4,6,11];

let array2 = array.every((number)=>

number %2 ===0

)

console.log(array2);

**some() method:**

The some() method executes the callback function once for each array element.

The some() method returns true (and stops) if the function returns true for one of the array elements.

The some() method returns false if the function returns false for all of the array elements.

The some() method does not execute the function for empty array elements.

The some() method does not change the original array.

Ex:

let array = [2,3,4,6];

let array2 = array.some((number)=>number%2===0)

console.log(array2);

**fill() method:**

The fill() method fills specified elements in an array with a value.

The fill() method overwrites the original array.

Start and end position can be specified. If not, all elements will be filled.

Order: filled,starting index,ending index

let array =[2,3,4,5,6,7,8,9,10];

const array2 = [...array];

array2.fill(10,2,8);

console.log(array2);

console.log(array);

**splice() method:**

The splice() method adds and/or removes array elements.

The splice() method overwrites the original array.

Order: Start,delete,insert

let array = [1,2,3,4,5,6,7,8];

let array2 =array.slice(0);

array2.splice(2,0,22);

console.log(array2);

fill,sort and splice will overwrites the original array elements

**Iterables, array like objects:**

Iterables are the objects that can be iterated over with for..of

Iterables are only for the string and array but it will throw an error when we use an object

Two types:

1.iterable over a string

2.Itearble over an array

**1.iterable over a string:**

Ex: let userName= ”sharath”;

For (let user of username)

{

Conole.log(user);

}

**Iterable over an array**

Const array = [“name”,”age”];

For (let array1 of array){

Console.log(array1);}

**Sets:** A JavaScript set is a collection of unique values

* Each value can only occur once in a set
* A set can hold any value of any data type
* No index based unlike array
* No duplicates are allowed in the set
* Order is not guaranteed

Set():

let array4 =["array","array2","array"];

let array5 =[...array4];

let items = new Set(array4);

let items2=new Set(array5);

console.log(items);

console.log(items2);

let dup = new Set(array4);

dup.add(2);

dup.add(3);

dup.add(4);

dup.add(5);

dup.add(6);

dup.add([2]);

// console.log(dup);

for (let dup1 of dup){

console.log(dup1);

}

**map data structure:**

**js map():**

* A Map holds key-value pairs where the keys can be any datatype.
* A Map remembers the original insertion order of the keys.
* A Map has a property that represents the size of the map.

let array = {

name:"sharath",

age:26,

place:"jangaon"

}

// console.log(array.name);

for (let key in array){

console.log(key);

}

let array2 = new Map();

array2.set("name","sharath");

array2.set("age","26");

array2.set(2,"Jangaon");

console.log(array2.get(2));

// for(let keys of array2.values())

// {

// console.log(keys,typeof keys);

// }

const firstName = new Map();

firstName.set("name","sharath");

firstName.set(2,"sharath1");

// console.log(firstName,typeof firstName);

for(let [key,value] of firstName) // here we use array destuctor [key,value]

{

console.log(key,value,typeof key,typeof value);

// console.log(Array.isArray(first));

}

Differences between JavaScript Objects and Maps:

Object Map

Not directly iterable Directly iterable

Do not have a size property Have a size property

Keys must be Strings (or Symbols) Keys can be any datatype

Keys are not well-ordered Keys are ordered by insertion

Have default keys Do not have default keys

For array and map we use for of loop

For object we use for in loop

**Clone using object assign:**

**Object.assign({},cloning variable);**

let nameChin = {

name: 1,

age:2

}

// let nameChin1 = {...nameChin};

let nameChin1 = Object.assign({},nameChin); // This is cloning using object

nameChin1.name1 = "sharath";

console.log(nameChin1);

console.log(nameChin);

**Optional chaining:**

It’s a error proof way to access the nested object properties

If it doesn’t exist it will throw an error

When we use “?.” It will show us as undefined

Let array = {

Name:”sharath”,

Class:10,

}

Console.log(array.Name);

Console.log(array.green.hello);

**Uncaught TypeError: Cannot read properties of undefined (reading 'hello')**

Console.log(array?.green?.hello); // displays undefined

**Methods of an objects**

* This
* Call
* Bind
* apply

**This keyword**

function personalInfo()

{

console.log(`My name is ${this.firstName} ${this.lastName} and age is ${this.age}`);

}

let person1 = {

firstName:"sharath",

lastName:"CHINTAKINDI",

age:26,

Info: personalInfo

}

let person2 = {

firstName:"CHINTAKINDI",

lastName:"sharath",

age:27,

Info: personalInfo

}

let person3 = {

firstName:"CHINTAKINDI",

lastName:"CHINTAKINDI",

age:28,

Info: personalInfo

}

person3.Info();

**Call() method of an object**

* The call method is predefined JS method
* With call() an object can use a method belonging to another object

function personalInfo(hobby,place) // here we are using parameters

{

console.log(`My name is ${this.firstName} ${this.lastName} and age is ${this.age}` hobby, place);

}

let person1 = {

firstName:"sharath",

lastName:"CHINTAKINDI",

age:26,

Info: personalInfo

}

let person2 = {

firstName:"CHINTAKINDI",

lastName:"sharath",

age:27,

Info: personalInfo

}

let person3 = {

firstName:"sharath",

lastName:"sharath",

age:27,

}

// using call method

person1.Info.call(person3); // by using call we can use the above function in person3

using arguments

person1.info.call(person3,”guitar”,”Jangaon”);

**apply() method:**

* Its same as call() method
* We will pass the arguments in an array format

person1.info.call(person3,[“guitar”,”Jangaon”]);

**bind() method:**

* An object can borrow a method from another object

Let dis = person1.info.bind(person3,[“guitar”,”Jangaon”]);

Console.log(dis);

New keyword and \_\_proto\_\_ and [[prototype]]: has to learn

**Class keyword:**

* In 2015 ECMA script also known as ES6 Introduced JavaScript Classes
* JS classes are templates for JS objects
* Use Class keyword to create the class
* Use constructor to create an object

class userName{

constructor(firstName,age,place,education){

this.firstName=firstName;

this.age=age;

this.place=place;

this.education=education;

}

about(){

return `My name is ${this.firstName} and age is ${this.age}`;

}

btech1(){

return this.education==="b.tech";

}

}

let user1 = new userName("sharath",26,"jangaon","b.tech");

console.log(user1.btech1());

**class and extends**

* Extends Keyword extends the class(indicated that a class is inherited from another class
* Subclass(child): the class that inherits from another class
* Superclass(parent): the class being inherited from

Ex: Extends keyword

// First info (parent)

class info{

constructor(fName,lName,age,place){

this.fName=fName;

this.lName=lName;

this.age=age;

this.place=place;

}

about(){

return `Hello ${this.fName} \n ${this.lName} and age is ${this.age} `;

}

address(){

return this.place==="Jangaon";

}

}

// second info(child)

class userInfo extends info{

}

// calling the parent

let user1 = new info("sharath","chintakindi",26,"Jangaon");

// console.log();

console.log(user1.about());

console.log(user1.address());

// calling the child

let user11 = new userInfo("sharath","chintakindi",26,"Jangaon");

console.log(user11);

console.log(user11.address());

**class super keyword**

* The super keyword is used to call the constructor of its parent class to access the parent’s properties and methods

/ class super

class animal{

constructor(name,age,place){

this.name=name;

this.age=age;

this.place=place;

}

// methods

animalInfo(){

return `${this.name} is at the age ${this.age} and they born in ${this.place}`

}

}

// Creating a variable to store the data

// let info1 = new animal("tiger","10 years","india");

// console.log(info1.animalInfo());

// creating second one

class animalInfo1 extends animal{

constructor(name,zoo){

super(name);

this.zoo=zoo;

}

zooInfo(){

return ` ${this.name} is sitauted in the ${this.zoo} zoo`;

}

}

let animal1 = new animalInfo1("lion","haha");

console.log(animal1.zooInfo());

**Getters and setters:**

// getters and setters

// getter

// class animalInfo12 extends animal{

// constructor(name,zoo){

// super(name);

// this.zoo=zoo;

// }

// get zooInfo1(){

// return ` ${this.name} is sitauted in the ${this.zoo} zoo`;

// }

// }

// let animal12 = new animalInfo12("lion","haha");

// console.log(animal12.zooInfo1);

// setter

class animalInfo12 extends animal{

constructor(name,age,place,zoo){

super(name,age,place);

this.zoo=zoo;

}

set looseName(looseName){

let [firstName, LastName] = looseName.split(" ");

this.firstName=firstName;

this.lastName=lastName;

}

}

const loose = new animalInfo12 ("tiger",27,"ameruca","hapap");

console.log(loose);

loose.looseName="lion tiger";

console.log(looseName);

**static methods and properties :simple topic**