**Why We use Javascript**

**Javascript is open source, light weight object oriented programming language. Used for creation of dynamic, responsive web centric application**

**Knows as synchronous single thread programming language**

**Primative Data Type / Non-Primative Data Type**

Primative Data Type : store single value : number, string, Boolean, undefined, null

**Feature of Javascript**

* Lightweight, interpreted programming language (don’t need to transfer code into another from before running into brower).
* Open source Cross platform
* Object Oriented
* Integrated with front end and back end technologies

**Hoisting**

Hoisting is JavaScript default behavior in which functions and variables are used before declaration.

We can access variable and functions in because js allocated memory to each variable and function before executing code

JS Engine move function and variable declaration to top of the scope

In Hoisting only declaration is hosted. Not initialization

Eg.

console.log(a);

Add();

var a;

function Add(){

console.log(2)

}

**Higher Order Function**

Function that operates on other functions, either takes them as arguments or returning them are known as higher order function

Eg.

// function with another function as argument

let a = (fn) => {

fn();

}

let b = () =>{

console.log('B Here');

}

a(b);

// function with another function as return

let c = () => {

let d = () => {

console.log('D here')

}

return d();

}

c();

**Synchronous vs Asynchronous**

By default javascript is synchronous single threaded programming language. This means instructions can run one after another and not in parallel

let a = 1;

let b =

2;

let sum = a + b;

console.log(sum);

But suppose we want to fetch large data from database and display on user interface. When interpreter reach to this line and fetch data till rest of code is blocked from executing until data fetch on UI

So delay compounded does not user would want to come across.

This problem is solve with asynchronous javascript

In order to properly implement asynchronous behavior there are few diff sol developer built. Each solution improves upon previous ones

Asynchronous functions are executing parallel with synchronous function

CallBack : - A callback is a function that is passed inside another function as arguments , and then called in that function to perform a task.

**Closures**

Closures are ability of function to remember the variables and functions of outer scope from inner scope.

function a(){

var x = 10;

function b(){

console.log(y);

}

b();

}

a();

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

function a(){

var x = 10;

function b(){

console.log(x);

}

return b();

}

let x = a();

so when function a() runs it seems that it returning function b() and it using x variable inside it

Therefore a() function instead of destroying value of x variable after execution..save value for futher reference. This is the reason why the returning function is able to use the variable declared in the outer scope even after the function is already executed

**This ability of a function to store a variable for further reference even after it is executed is called Closure**

**Var Vs Let**

* Var variable we can access before initialization let variable can’t access
* Var variable can be re-decalare but let can’t
* Var variable global scope and let has functional

if(true){

var x = 10;

let y = 20;

}

console.log(x);

console.log(y);

**This Keyword**

This keyword always depend on object that is calling the function

* Alone and in functions, this refers to the global object.
* In an object method, this refers to the object.
* In an event, this refers to the element that received the event.
* In a function, in strict mode, this is undefined.

**Bind(), Call(), Apply() in JavaScript**

In js every object as properties and methods but this methods use only by respective objects to overcome this restriction this methods are use.

Call() and Apply() methods allow you to use methods of another object.

The call() and Apply() method can take two parameters:

thisArg - The thisArg is the object that the this object references inside the function func.

arg1, ... argN (optional) - Arguments for the function func.

.bind(someobj) -> does not invoke the function, it just allows you to bind whatever object you want, you have to call the function yourself.

.call(someobj) and .apply(someobj)-> both invoke the function immediately, and modify the context. The only difference is how you pass your own arguments.

let a = {

fn: 's',

ln: 'p',

fn: function(x,y){

console.log(x + ' ' + y + ' ' + this.fn + ' ' + this.ln);

}

}

let b = {

'fn': 'k',

'ln': 'c'

}

a.fn.call(b,'Hello', 'Call');

a.fn.apply(b, ['Hii','Apply']);

let x = a.fn.bind(b,'Hello', 'Bind');

x();

**rest parameter and spread operator**

* rest parameter gather remaining elements passed into the function as an array or object
* spread operator to spread array values or iterables into maybe an array or object.
* The rest operator (…) allows us to call a function with any number of arguments and then access those excess arguments as an array
* spread operator expands elements of an iterable
* The rest operator also allows us in destructuring array or objects.

Spread Operator

Eg.

const arr = ['a','b','c'];

function func(x,...y){ // used rest operator here

console.log(x);

console.log(y);

}

func(...arr); // used spread operator here

**Anonymous Function**

- Anonymous functions is a function definition that is not bound to any identifier.

- Used for passed argument to higher order functions or used for constructing result of higher order function that needs to return a function.

**Lexical Scope**

A lexical scope in JavaScript means that a variable defined outside a function can be accessible inside another function defined after the variable declaration. But the opposite is not true; the variables defined inside a function will not be accessible outside that function.

**Arrow Functions**

* Arrow Functions are shorter way to write function, arrow functions are functions without name
* Using Arrow Functions curly braces, parenthesis, return are optional
* Arrow functions are also known as anonymous functions
* To call arrow function and reuse it. We need to store in variable
* Arrow Functions mostly use as callback functions

**Diff Between Regular Function and Arrow Function**

1.Behevour of this keyword is different in regular and Arrow Function

var variable = 'Global Level Variable';

let myObject = {

variable: 'Object Level Variable',

arrowFunction: () => {

console.log(this.variable);

},

regularFunction() {

console.log(this.variable);

}

};

myObject.arrowFunction();

myObject.regularFunction();

2. Argument Object does not present in arrow function.

function Add(x,y){

console.log(arguments);

}

Add(5,5);

let Add1 = (x,y) => {

console.log(arguments);

}

Add1(10,4)

3. Duplicate Arguments not allowed in arrow function

let a = (a,a) => { console.log(a) } a(5); // return error

function a(x,x) { console.log(x)} a(5); // return undefine

4. Implicit return : If arrow function contain one expression we can omit curly braces, then the expression will be implicitly return

5. Arrow functions can not use as Contructor

**Currying**

It is technique in functional programming. Transformation of function having multiple arguments into several functions of a single arguments in sequence. Currying helps you avoid passing the same variable multiple times, and it helps you create a higher order function.

function add(x,y,z){

return x+y+z

}

console.log(add(1,2,3));

function addition(a){

return function(b){

return function(c){

return a+b+c;

}

}

}

// alternate way

function addition(a){

return (b) => (c) => a + b + c;

}

let res = addition(2)(3)(4);

console.log(res);

**Scope And Scope Chain**

Scope in JS determines the accessibility of variables and functions at various parts of one’s code.

Three Types of Scope :

1. Global
2. Local/Functional
3. Block

Global Scope : Variables and Functions are declared outside the functions are knows as Global Space

Local Scope : Variables and Functions are declared inside the functions are knows as Local Space

Block Scope : block scope is relate to let and const. block scope tells that variable declare inside block {} scope can only access in it.

Scope Chain:

It is process in which javascript engine search value in scope of functions however search in lexical manner. If value not found in scope of fuction. Find in parent scope. If not there, global space is last place to check

var a = "Hello world";

function first() {

var b = "I am Rahul.";

second();

function second() {

var c = " Subscribe to RAHULISM";

console.log(a + b + c);

}

}

first();

**Object Constructor/Function Constructor**

Object Constructor/Function Constructor use when we want create multiple objects having same properties and function It is use as classes before ES6

Obj1 = {

Name: ‘shubham’,

Age: 23

}

Obj2 = {

Name: ‘kunal’,

Age: 25

}

*function Details(name, age){*

*this.fname = name;*

*this.age = age;*

*}*

*Student1 = new Details('shubham' ,24);*

*console.log(Student1);*

**Prototype and Prototype Object**

The prototype is an object that is associated with every functions and objects by default in JavaScript.

In js we have object likes Dates, Math and Array this all inherit properties from its prototype. On top of the chain every prototype inherit properties and methods from Object.prototype.

Prototype is blueprint of the object. That allows us use of properties and methods on an object even if properties and methods not exist on the current object

The JavaScript prototype property allows you to add new properties to object constructors:

var arr = [];

arr.push(2);

console.log(arr);

**DOM**

DOM stands for document object model. It is also known as document tree having branches and each branch has node represent element. When brower tries to render HTML to brower its create DOM object based on HTML Document

**BOM**

Brower Object Model in javascript includes properties and methods for javascript to interact with brower. BOM provide window Object. Allows us to talk with brower

**Classes**

Classes are nothing but new way of writing function constructor in ES6. Class is not js object it is template for js objects

Construcrtor method called automatically when new object is created

The constructor method is a special method:

It has to have the exact name "constructor"

It is executed automatically when a new object is created

It is used to initialize object properties

If you do not define a constructor method, JavaScript will add an empty constructor method.

Eg.

class Student{

constructor(name,rollNumber,grade,section){

this.name = name;

this.rollNumber = rollNumber;

this.grade = grade;

this.section = section;

}

// Methods can be directly added inside the class

getDetails(){

return 'Name: ${this.name}, Roll no: ${this.rollNumber}, Grade:${this.grade}, Section:${this.section}';

}

}

let student2 = new Student("Garry", 673, "7th", "C");

student2.getDetails();

**Object Destructuring**

Object Destructuring new way to exact or assign element from array or object.

let obj = {

name: 'Shubham',

age1: 23,

city: 'Pune'

};

//Old way to extract element

let name1 = obj.name;

//New way to extract element

const { name , age } = obj;

console.log(name);

console.log(name1);

const arr = [1,2,3];

//old way

let x = arr[0];

const [a,b,c] = arr;

// new way

console.log(a);

**Garbarge Collector**

When you declare variable, function, object or array it store some where in memory

Suppose I have one function where I declare one variable and printing that variable after execution of program data inside location is garbage data which needs to cleared. So the automatic clearing the memory spaces which don’t contain useful data and reallocating those memory to some other data

**Pure Functions**

Pure Functions is the functions that returns same result if the same argument is passed

**function** calculateGST( productPrice ) {

**return** productPrice \* 0.05;

    }

    console.log(calculateGST(15))

**Local Storage and Session Storage**

Locat Stoage : website store data in local storage to reduce loading time. This data does not deleted when the brower is closed, are available for future session

Session Stoage : data stored in session storage will expire when page session end

**Immediately/Self Invoked Function**

Immediately Invoked Functions runs as soon as it defined. To run this function, we need to call immediately unless it return body of function

(function()

{

// Do something;

})

();

**Javascript Strict Mode**

Strict mode is ES6 features . this makes it easier to write good and secure js code

Enable the strict mode by declaring this in the top of your script/function

Some Feature of Strict Mode:

* Can’t use variable before declaration
* Can’t delete variable, functions after creation
* Duplicate parameters not allow in function

**Temporal Dead Zone**

Temporal Dead Zone behavior occur when variable declare using let and const are access before initialization

**Lexical Scoping**

Inner functions contain scope of outer function even after outer function is excuted

Lexical scope is the ability for a function scope to access variables from the parent scope.

**Types of Error**

Syntax error, reference error and type error

**isNaN() function**

isNan() function returns true if the variable value is not a number

**exec () and test () methods**

exec() is regular expression method it is used to search string with specific pattern if found return string else return null

test() is regular expression method it is used to search string with specific pattern if found return true else return false

**Set and Weak Set**

Set object is used to store the unique values. The values can be any type i.e primitive or non-primative

Weakset object used to store weakly held object

var set = new Set();

set.add("jQuery");

set.add("AngularJS");

set.add("Bootstrap");

**Map and Map Object**

Map javascript is used to map keys to values. It store each element as key-value pair

The key-value pairs can be of both primitive and non-primitive types.

Diff between map and map object is keys and values in weakmap should always be an object.

var map=new Map();

map.set(1,"jQuery");

map.set(2,"AngularJS");

map.set(3,"Bootstrap");

Different Between Callback Promises and Async Await

Callback : - A callback function is passed as arguments to another function

Promises :- Promise is object that is something achieved and completed in future

Callback promises async await all used to handle asynchronous operations callback handle asynchronous operation in traditional way, promises handle asynchronous operation in elegant way and async await handle asynchronous operation in more convenient and elegant way

ES6 and ES5

ES5 is fifth version of EcmaScript and ES6 is sixth version fo EcmaScript

ES6 we have one addition primitive data type known as symbol

In ES5 we can define variable only with var in ES6 there is two new new way to define variable let and const through which we define variable

ES5 is lower in terms of performance than ES6

ES6 also have some additional features such arrow functions, classes, object, promises, map fuction and spread operator

Main different between the spread and rest operator is spread operator is used to spread out the elements of an iterable while rest operator is used put multiple function arguments in array

Second diff is spread operator is used in function invocations whereas the rest operator is used as a function parameter

Apart from the function invocation spread operator also use in merging arrays, copying arrays