Javascript developed by **Brendan Eich**, Netscape,1995. Mocha->Livescript->javascript

- Why name as script – it execute as the page loads

*- <script type=”text/javascript”>* type=”text/javascript” is not needed

- External js files can be stored as cache in browsers

-’use strict’; to use only latest functionality of javascript versions

-whitespace, case-sensitive

|  |  |  |  |
| --- | --- | --- | --- |
| Javascript reserved keywords | | | |
| break | as | any | switch |
| case | if | throw | else |
| var | number | string | get |
| module | type | instanceof | typeof |
| finally | for | enum | export |
| while | void | this | new |
| null | super | catch | let |
| static | return | true | false |

**Variables**: represents name for a memory block

var name=”rajesh”; //scope dependent for a function or window object

let name=”rajesh” //used as block scope, cannot be redeclared

const NAME=”rajesh” //cannot be reassigned, redeclared, requires declaration, immutable

Template literal: can use ${name} inside string. var name = ‘rajesh’; console.log(“my name is ${name}”);

- var {name,age,job}={name:"rajesh",age:"22",job:"it"}; console.log(name); //rajesh

**Datatypes:**

1.Number - integer & floating point

2. String - one or more characters

3. Boolean - true/false

4. Null - Unknown but (null==undefined) is true , (null==0) is false

5. Undefined – Known by non - defined

6. Object - collections of variables and properties

7. Symbol – unique identifiers

**Array**:

var arr[“name”,’age’,’job’]=[“rajesh”,”12”,’vetti’]; console.log(arr[“name”]); //rajesh

[1,2,3].includes(2); //return true false

[1,2,3].findIndex (2); //return true false

**String**:

“Hello world”.include(“world”); //return true false

“Hello world”.startswith(“H”, index); //return true false

“Hello world”.endswith(“d”, index); //return true false

“sorry ”.repeat(100); //print sorry 100 times

**Conditions**:

if

if..else

nested if else if

switch case

**Loops**:

for

while

do...while

Loop control:

break:

continue:

for...in *for(key in obj){ console.log(obj[key]);}*

**Function**:

*function functionName(){...body....}*

*functionName(); //function calling*

- function cannot be access outside scope

*if(true){function add(){…}} add();//cannot be called, undefined*

**Arrow function**:

*let functionName = (parameter)=>{...function body..…};*

*it does not have its own this object.*

*var obj = {name:"rajesh",*

*getName: function(){ (function(){console.log(this.name)})(); },*

*getNameArrow: function(){ (()=>{console.log(this.name)})(); }*

*}*

*obj.getNameArrow(); //print rajesh*

*obj.getName(); //undefined*

**Default parameter:**

-Primitive function get(i=1){ return i;} get(undefined);

-Array function get(a=[]){return ...a;} get([5]);

-Object function get({a=5}={}){return } x={a:5}; get();

|  |  |
| --- | --- |
| **Spread operator(rest parameters)** | Es6 |
| *function bigNum(){  var args=Array.prototype.slice.call(arguements,0);  var args=[].slice.call(arguements,0);  //for converting arguements to array }bigNum();* | *function bigNum(a,b, ...argArray){  //a=1,b=2,arrgArray is an array[3,4,5] } bigNum(1,2,3,4,5);* |
| We can combine two arrays. a=[1,2,3]; b=[4,5]; c= [...a,...b]; //c is [1,2,3,4,5]  console.log(...a); //1 2 3  We can combine two arrays. A=[3,4,5];b=[1,2]; a.push(...b); instead of // Array.prototype.push.apply(a,b); | |

**Object:**

let user = new Object(); // "object constructor" syntax

let user = {}; // "object literal" syntax

delect user ; //to delete the objec

{} means each time new reference allocated

var obj = {name : ”raj”,

“nick name”: “rajeshu”, //multiword key allowed  
 func(){…..}}

obj[‘nick name’]; //multiword key can accessed by square brackets

-Square brackets are much more powerful than the dot notation

-Shorthand syntax name=”raj”; var user = {name,age:23}; //user is {name:name,age:22};

-Objects are muttable   
 a={name:1}; b=a; b.a=2; console.log(a); //gives {a:2} both uses same reference

b={}; console.log(a); //a is {a:2} //while {} b points to new location reference

**Comparing object** : obj1={}; obj2={}; obj1==obj2; //false obj1===obj2; //false

obj1={age:22}; obj2=obj1; obj1==obj2;//true obj1===obj2;//true

**Const object:** it is changeable, but cannot be reassign

**Clone object:** newObj = Object.assign({},oldObj);

Garbage collections: The variable and objects which cannot be reached, get destroyed

|  |  |
| --- | --- |
| **Class in javascript** | Es6 |
| function Cricketer(name,age,position){  this.name=name; this.age=age; this.position=position;  }  Cricketer.prototype.changePosition=function(position){  this.position=position;  }  var cricketer = new Cricketer(“rajesh”,”22”,”batting”);  console.log(cricketer);  crickter.changePosition(“bowler”);  console.log(crickter); | class Cricketer {  constructor(name,age,position){  this.name=name;this.age=age;this.postion=position;  }  changePosition(position){  this.position=position;  }  }  let crickter = new Crickter(“rajesh”,”2”,”batting”);  console.log(crickter);  crickter.changePosition(“bowler”);  console.log(cricketer); |

**Call Apply Bind:**

varobj = {num:2};

var func=function(a,b){ console.log(this.num+a+b);}

func.call(obj,1,2);

func.apply(obj,[1,2]);

var bound = func.bind(obj); bound(1,2);