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 re-declared

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**: special kind of object

let arr = new Array();

let arr = [];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

arr.pop(); //remove last element

arr.shift(); //remove element at begining

arr.push(4); //insert element at end ,arr is now [1,2,3,4]

arr.unshift(0); //insert element at beginning ,arr is now [0,1,2,3,4]

arr.length;

**Symbol.iterator:** var arr=[1,2];   
 var iterator=arr[Symbol.iterator]();

console.log(iterator.next()); //{value:1,done:false}  
 console.log(iterator.next()); //{value:2,done:false}  
 console.log(iterator.next()); //{value:undefined,done:true}

**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

String.raw”it is not new line /n” //print as it is - it is not new line /n

**Numbers:**

let billion = 2000; //can written as let billion = 2e3;  
 let millisec = 0.002; //can written as let millisec = 2e-3

Hexad: alert( 0xff ); // 255 alert( 0xFF ); // 255 (the same, case doesn't matter)

num.toString(base); eg.a=3; a.toString(2); //0011 default base is 10

Two dots: directly called from number. 3.toString(2); //0011

Infinity (and -infinty) represents great (or less) than anything. isFinite(number); //checks whether its not infinity

NaN represents a non number type isNan(number); //checks whether is not number

**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]);}*

for...of *for(arrKey of arr){ console.log(arrVal);}*

**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

**Object wrapper:**

Primitive datatypes is no an object, but a object wrapper is temporarily created while using it functions eg.str.split() . But for null,undefined there is no functions and no object wrapper created.

eg. str = new String(“rajeh”); str.test=5; console.log(str.test);//undefined

|  |  |
| --- | --- |
| **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);

**Object. create:** create an empty object. Make the given arg object as prototype of the created empty object  
 oldObj = {this.name:”rajesh”}  
 Object.create(oldObj); //create obj {\_proto\_:this.name:rajesh………}

**eg:**

var Car = function(){ this.color='red'; }

Car.prototype.getColor=function(){ return this.color; }

var ToyCar = function(){ };

ToyCar.prototype=Object.create(Car.prototype);

ToyCar.prototype.color='orange';

var obj = new ToyCar();

console.log(obj.getColor());

It is real alternative for

let sayHiMixin = { \_\_proto\_\_: anotherObject} //but we should nor use \_\_proto\_\_ so we using Object.create

**Object.setPrototypeOf:** same like object create but it works for simple{} object literal not function constructor

var obj1 = {drive:function(){return ‘i can drive’;}, walk:function(){return ‘i can walk’;}};  
 var obj2 = { drive(){return **super.drive()**;}}

Object.setPrototypeOf(obj2.obj1); //obj1 will get obj1 as a prototype   
 obj2.walk(); //call walk function in obj1   
 obj1.drive(); //call drive function in obj, because of super object.

**Object .assign:** copy and append object to existing(given) object  
 var obj1 = {color:’red’};  
 var obj2={}; Object.assign(obj2,obj1); //1st way to assign

var obj3 = Object.assign({}.obj1); //another way to assign  
 var obj4 = function(){a,b}{ Object.assign(this,{a,b});} //also used in constructor

- you can merge more than one object eg: Object.assign(obj2, obj1.1,obj1.2)

**Sets:** collection of unique values

var mySet = new Set();

mySet.add(1).add(2).delete(1).clear();;

var mySet = new Set([1,2,3,5,4,4,4,4,4]); //mySet is 1,2,3,4

console.log(mySet.size);

for(val of mySet) { console.log(val);} //can be iterable

can convert Sets to array: console.log([..new Set([1,2,2,3])]);

Array.from(new Set([12,2,3]));

**WeakSets:** can have only as objects

var myWeakSet = new WeakSet([{a:1},{b:2}]);

myWeakSet.add(1); //throw error

myWeakSet.add({a:1});

**Maps:** can have more than one object key

var myMap = new Map();

myMap.set(a,’a’).set(b:’b’).set(a:’c’).delete(b);

for(let [key,value] of myMap.entries()){  
 console.log(key,value);  
 }

methods: new Map();

map.set(key,value)

map.get(key)

map.has(key)

map.delete(key)

map.clear()

map.size

**Class constructor:**

class Car{  
 construct(arg){}

func1(){}

static func2(){} //inside **static** we cannot use this object variables }

class Honda extends Car{  
 constructor(arg){ super(arg);….} //must call super constructer, otherwise error will occur

func1(); }

var obj = new Car(arg);

**Promise – resolve,reject:**

var promise = new Promise(function(resolve,reject){  
 setTimeout(function(){

success=true;

if(success){resolve(‘result’);} //wait until resolve function beeing called  
 else{reject(‘sorry’);} },1000);

});

promise.then( function(resolveResult){ ….handle resolveResult…}).catch(function(rejectResult){….handle reject…});

promise.then( function(resolveResult){ ….handle resolveResult…},function(rejectResult){….handle reject…});

promise.then( null,function(rejectResult){….handle reject…}); //to handle only error

promise.all([promise1, promise2….promisen]);

**Async Await::**

var promise1 = new Promise((resolve,reject)=>{setTimeout(resolve("rajesh"),3000);});

var promise2 = new Promise((resolve,reject)=>{setTimeout(resolve("23"),3000);});

var promise3 = new Promise((resolve,reject)=>{setTimeout(resolve("not a bad guy"),3000);});

var callSync=async ()=>{

var promresult1 = await promise1; console.log("his name is :"+promresult1); //his name is rajesh

var promresult2 = await promise2; console.log("his is :"+promresult2); //his is 23

var promresult3 = await promise3; console.log("sometimes he is :"+promresult3); //his not a bad guy

return 'success';

}

callSync().then((resultText)=>console.log('result is '+resultText));

Await should use only inside async: syntax: await promiseName; //the js will pause until result come from promise