const timeZone = Intl.DateTimeFormat().resolvedOptions().timeZone;

1. Java script

Js is a dynamic programming language. Run in web browsers only. It is loosely coupled.

Top-down execution.

<script type=”text/javascript” src=” main.js”></script>

<noscript>java script does not enable in your system<noscript> // it renders when in browser Js is not enabled.

If “src” is there, in b/w script tag code will not execute.

“defer” -> will stop execution of script file and execute all script files which has not defer then defer file will execute.

1. Java script variables.

var abc = 5. // to store the data.

typeof abc. // number.

Null: if we assigned value as ‘null’. By adding users. Typeof will come as “object”.

Undefined: if value did not assign it is called as undefined. Typeof will come as “undefined”.

Null == undefined // true. Value wise both are same.

Null === undefined // false. Data types are different.

Nan: Typeof will come as “number”. It is not a number. Ex: 2\*d // is nan

Arrays:

var abc = [1,2,3]; typeof abc // “object” .

Objects: store data in ‘key’ ‘value’ pair.  
 var abc = {name:”test1”, age:22}; typeof abc // “object”.

typeof undefined // "undefined"

typeof 0 // "number"

typeof 10n // "bigint"

typeof true // "boolean"

typeof "foo" // "string"

typeof Symbol("id") // "symbol"

typeof undefined // undefined

typeof Math // "object" (1)

typeof null // "object" (2)

typeof alert // "function" (3)

1. “use strict”

Using “use strict” if we remove “;” it throws error.

1. Different types of create object.
2. A= {};
3. A = new object ();
4. Using function constructor
5. Object. Create ()
6. Dynamic typing  
   var var1 = 5;

var Var1 = 10; // java script is case sensitive.

Dynamically change the variable of the name also.

Variable hosting It takes all variable in program. And hosted at top. And then it initializes value and print the value.

Declaration of variable is not important initializes the value is important.

Var1 = 5; // initializes the value.

Var var1; // declaration as ‘var’.

1. Function in javascript.

Set of code which can execute at any time we want.

function name(){ // Function Declaration

Console.log(‘test1’)

} // no semi-colon needed

name() // calling function.

Var name = function (){ // Function expression (anonymous function)

Console.log(‘test1’)

}; semi colon need.

name() // calling function.

typeof name // function.

Typeof name() // based on return type. If no return type it will come “undefined”.

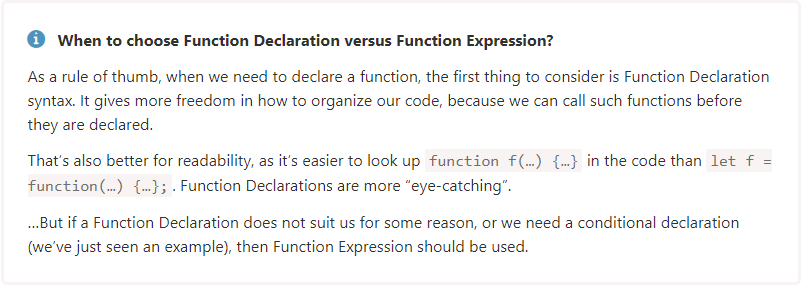
Functions arguments and return

Function add(a, b){ return a + b}

add(3,4); // 7.

In function declaration variable can access inside the function only.

In function declaration we can call before creation function. Whereas function expression not allowed.



1. Javascript conditional statements

True -> 1, “ ”,-1,”0” (-1 == true // false).

False-> 0,’’, null, undefined, NaN

As it treats null and undefined similarly, we’ll use a special term here, in this article. For brevity, we’ll say that a value is “defined” when it’s neither null nor undefined.

The result of a ?? b is:

* if a is defined, then a,
* if a isn’t defined, then b.

The important difference between them is that:

* || returns the first *truthy* value.
* ?? returns the first *defined* value.
* let height = 0;
* height || 100; // 100
* height ?? 100; // 0

If(true){

Console.log (‘true block’);

}else{

Console.log (‘false block’);

}

1. Switch statement

switch(2){

case 1:

console.log (‘1’);

break;

case 2:

console.log(‘2’);

break;

case 3:

console.log (‘3’);

break;

default:

console.log(‘default’);

}

1. For loop

for(var i=0; I <5;i++){console.log(i)}; // 0, 1, 2. 3, 4.

Break – it does not continue loop if conduction satisfies. Come out of loop.

Continue – it skip the value based on condition. And continues the loop.

While loop

While(i < 5){console.log(i); i++} // here we can write our own conduction. It checks the condition first. And execute.

Do {console.log(‘first time’)} while(conduction) // it execute logic and checks the conduction.

for (var i = 0; i < 3; i++) {

   console.log('inner',i); // 0, 1, 2

}

console.log('final',i) // 3

if we use let it get error no such variable.

* while – The condition is checked before each iteration.
* do..while – The condition is checked after each iteration.
* for (;;) – The condition is checked before each iteration, additional settings available.

1. Operators’ java script

“hello” + 3 –> “hello3”;

[1,2,3] + “hello” -> “1,2,3hello”;

True + “hello” -> “truehello”;

True+false (1+0)- > 1;

True+4 (1+4)- > 5 (false+4 -> 4);

Null + 4 (0+4) -> 4;

Undefined+4 -> NaN; (NaN + 4 -> NaN).

“hello”/1 -> NaN.

Number(‘abcd efg’) // NaN.

Number(true); // 1

Number(false); // 0

Boolean(1); // true

Boolean(0); // false

Boolean("hello"); // true

Boolean(""); // false

Boolean("0"); // true

Boolean(" "); // true

let counter = 1;

let a = counter++; // (\*) changed ++counter to counter++

alert(a); // 1

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

let counter = 1;

let a = ++counter; // (\*)

alert(a); // 2

Number(undefined) // NaN

Number(null) // 0

null + 1 = 1

undefined + 1 = NaN

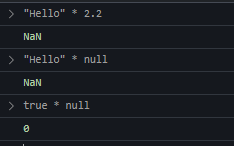
let a = 0;

alert( Boolean(a) ); // false

let b = "0";

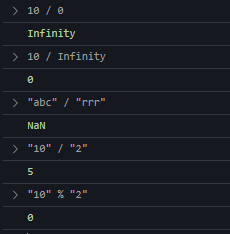
alert( Boolean(b) ); // true

alert(a == b); // true!

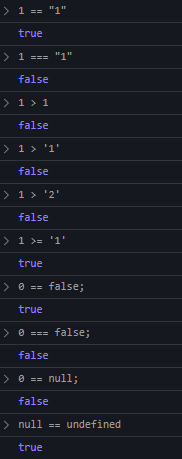
--------------------------------------------------------------------------------  
 

A screenshot of a computer program

Description automatically generated with medium confidence



1. Javascript comparing operators.  
   null/undefined are converted to numbers: null becomes 0, while undefined becomes NaN.



0 == undefined -> false.

NaN == NaN -> false.

0\*NaN -> NaN.

1. Primitive types vs reference type

Var a= 5; var b = a; var str = “hello” -> primitive types. In memory location for a with value. For b it will create new memory. If we change b value a will not change value.

Var a = {a=1}; var b =a; -> reference type. A variable holds pointer location. When we change b it reflect on a also.

Typeof array // “object”

Var a = [1,2,3];

Var b = a;

A= [‘a’,’b’,’c’];

Console.log(a) // [‘a’,’b’,’c’];

Console.log(b) // [1,2,3];

1. Array methods in javascript

Var a = [1,2,3];

a. length // 3.

a[0] // 1; a[3] // undefined; a[3] = 10 // [1,2,3,10];

loop -> forEach(ele,I,arr), for(var I =0; i<a.length;i++){}

push -> to add element at the end of the array.

pop -> to remove element at end of array.

shift -> it removes element at starting of the array.

unshift -> it add element at starting of the array.

indexOf -> to check element is present it the array are not. And return index. Not exist it return ‘-1’.

slice -> to slice array at part it will return new array. It will not be affected on original array.

splice -> it remove elements from original array. A.splice(index, total elements, new elements).

Map -> it performs operation on array and return new array. Not effected on old array.

Filter-> to filter array based on condition it will return elements new array. Not effected on old array.

Reverse-> it reverse new array. It effects old array it self.

Concat-> it will concat two arrays a.concate(b); and form new array.

Join-> it will take array and give string with separator. [1,2,3].join(“: ”) // 1: 2: 3.

Reduce-> arguments (total, value). It return value.

1. Javascript objects

let {name:nm ,age:{userage}} = {name:'test1',age:{userage:20}};

Objects are assigned and copied by reference. In other words, a variable stores not the “object value”, but a “reference” (address in memory) for the value. So copying such a variable or passing it as a function argument copies that reference, not the object itself.

All operations via copied references (like adding/removing properties) are performed on the same single object.

To make a “real copy” (a clone) we can use Object.assign for the so-called “shallow copy” (nested objects are copied by reference) or a “deep cloning” function structuredClone or use a custom cloning implementation, such as [\_.cloneDeep(obj)](https://lodash.com/docs#cloneDeep).

var person = {name: ’abc’, age:28};

person.name // ‘abc’ to get only name. (or) person[‘name’] (to get dynamic value).

typeof person // “[object]”.

var person = {name: ’abc’, age:28, welcome: function(){ console.log(‘welcome’, this.name)}};

person. welcome // ‘welcome abc.

var newsperson = new Object();

newperson.name = ‘def’.

var anotherPerson = Object.create(null); // protype is inheritance properties of the object.

var anotherPerson = Object.create(person);

object prototypes:-

console.log(person.\_\_proto\_\_); // undefined (inheritance which will be inherited).

Preson.toString() // it contains prototype.

Object.prototype.hello = function(){

Console.log(‘hello world’,this.name);

}

person.hello(); // present in global ‘object’.

var person1 = Object.create(person);

console.log(preson.name); // abc.

console.log(preson1.hello()); // hello world. It checks this method in person1 are method is not then it checks in “object prototype”.

var person2 = Object.create(person);

Comparing prototypes:

Person2.\_\_proto\_\_ == person //true.

preson2.\_\_proto\_\_.\_\_proto == person.\_\_proto\_\_ //true.

preson2.\_\_proto\_\_.\_\_proto == Object. prototype //true.

Object.getPrototypeOf(person2) == person //true.

Create object using constructor functions.

function person(name){

this.name = name

}

Var newperson = new person(‘name’); // same as new object.

person.name = “abc”; //console.log(newperson); -> ‘abc’. newperson.\_\_proto == object.prototype //false. newperson.\_\_proto == person.prototype //true.

In function we do not assign value it shows default value.

* Person instance Object //true. It is a normal object.
* Person instance Object // true. Using new Object ().
* Person instance Object // false. Using Object.create(null). Not defining prototype. Create objects with no prototypes. Object default methods also not access.
* Using the function to create object it will work it inheritance form object. Instance also true. Blue print.

Lets learn call(), apply(), bind()

in function we log ‘this’ it prints window object. In object in methods ‘this’ print object details. We can send our own ‘this’. we need to use below.

Call -> call method instantly. (this, ’argument1’, ’argument2’).

Apply -> call method instantly. (this,[ ’argument1’, ’argument2’])

Bind-> return method. (this, ‘argument1’, ’argument2’).

Object.define

Don’t change the object values.

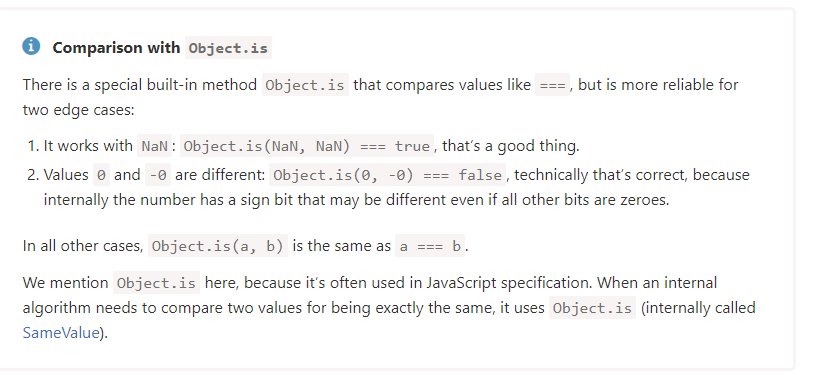
Object.defineProperty(person,’id’,{value:1}); writable: true.

person.id = 4 //it will not change the value. We can use ‘get’ and ‘set’.  
 ----------------------------------------------

delete person.name; //it will delete name from object.

name in person; // true, check property present in object is not.

for (var field in person) {console.log(field)}.// name, age.



1. Try, catch, finally

Var b =10;

try{

var c = a \* b;

}catch (error){

Console.log(error);

}finally{

Console.log(‘finally’);

}

Don’t block with error occurs it will run.

1. Immediately invoke functions and functions.

Function with has its own local scope. It will not affect global scope.

(function hello(number){

Console.log(“hello world”, number);

})(10);

function hello(msg){

console.log(“hello world”);

console.log(arguments); // we will get array of arguments.

}

hello.name // it will give name of the function.

hello.length // no of arguments

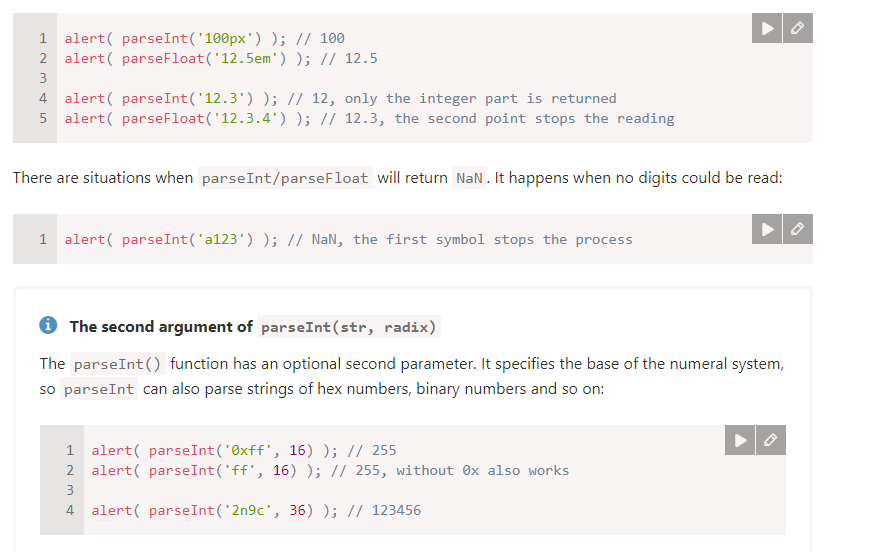
Let hello = function(msg){

console.log(“hello world”);

console.log(arguments); // we will get array of arguments.

}

1. Format values in java script



var a = ‘10’;

Number- convert string to number;

parseInt- convert string to number; ‘string’ and return NaN. parseInt(‘FBB12’,16). It skips decimal value. parseInt(str, base)

toString:- number to string convert. num.toString(base)

toFixed:- it return string. Ex:- (10.3333).toFixed(2) // 10.33.

parseFloat:- it returns floating number.

Var a = “hello world”. (string)

a[1] // ‘e’.

a.charAt(2) // ‘l’.

a.length // 11. Include white space.

Concat -> it concat two string.

Split-> convert into array.

Trim-> it trim white space start and end. Not in b/w.

Math methods

Pi -> Math.PI.

Math.E -> 2.718281828459045.

Math.abs -> it returns absolute value form ‘0’.

round -> (1.27) // 1. Decimal number less than ‘5’ it take original number. It greater than or equal 5 it take next number.

Ceil-> upper value it take. Next coming number. (round up)

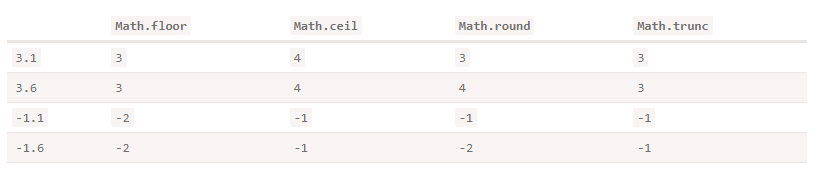
Floor-> least number.(round down)

Max(1,2,3) -> 3.

Min(1,2,3) -> 1.

Random -> random number b/w 0 and 1. Math.floor((Math.random() \* 100) + 1).

Trunc()-> which remove decimal point.



1. Regular Expression

Var str = “abc”;

Var pattern = /ab/; // pattern.exec(str); // [“ab”];

Pattern.test(str) // true.

Str.match(pattern) // [“ab”].

1. Dom in JavaScript (document object model)

innerWidth-> only the width of web page. With dev tools

outerWidth-> width of the browser.

textContent -> to change text.

nextsibling -> to get next element sibling.

getElementsByTagName(‘ul’) // in array

getElementsByClassName(‘sample’);// in array

getElementById(‘id’);// it return one element.

querySelector(‘#sample’)// css selector, class name, tag name.

querySelectorAll() // return array.

appendChild() // append child particular element.

insertBefore() // before which element.

removeChild() // it remove’s the child.

Event handling

document.querySelector(‘button’).onclick = function(){console.log(‘button clicked.’)};

document.querySelector(‘button’).addEventListener(‘click’. function(){

console.log(‘clicked…’);

});

removeEventListener(‘type of event’, “method name”)

event bubbling and stop propagation.

the event movement beings from target to outermost element in the file.

• stopPropagation:- prevents further propagation of the current event in the capturing and bubbling phases.

• preventDefault:- prevents the default action the browser makes on that event.

stopImmediatePropagation:- method to add two click events to the same button. However,

the stopImmediatePropagation() method stops the second event handler from being

1. Hoisting in js.

Compiling and execution phase. Var, functions all this thing allocated some memory before it execution.

Var -> functional scope(global)

Let, const -> are block level scope. (script)(temporal dead zone)

1. Clouser (lexical scope)

Accessing variables from the outside.

That functions which are declare in side a function (map, callbacks…).

function generateInput(input){

var number = input;

return function(){

return number \* 2;

}

}

Console.log(generateInput(100)()); // 200.

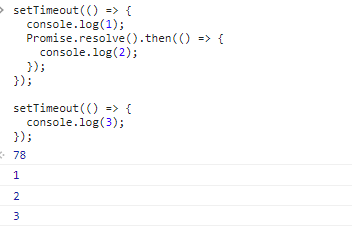
1. States in java script (or) scope in js.  
     
   var -> global scope

Let, const -> block level scope.  
functional scope. Local scope

Global scope: if variable declare at top of javascript it is called as global scope.

Local scope: with in function or conditional block.

1. The event loop

Yes, that's because Promise. Resolve() has a different queue called the JOB QUEUE or MICROTASK QUEUE, and this Job Queue has the higher Priority than the Callbacks Queue. Note that we are dealing with Promises now and not the callbacks when we do Promise. Resolve ! So JavaScript ES6 came up with this Job Queue to handle Promises differently and call backs differently  
  


1. Event flow in java script
2. JavaScript how to navigate to another page  
     
   location. Replace("http://example.com");

location.href = "http://example.com"

window.location = "http://example.com"

window.location.href = <http://example.com>

ES6 study

Javascript -> Ecmascript (es5).

Es6 not supported by browser. Latest version of javascript.

Compiler “babel compiler”. convert javascript.

Just don’t forget to use a transpiler (if using modern syntax or operators) and polyfills (to add functions that may be missing). They’ll ensure that the code works.

For example, later when you’re familiar with JavaScript, you can setup a code build system based on [webpack](https://webpack.js.org/) with the [babel-loader](https://github.com/babel/babel-loader) plugin.

1. Let keyword

Let is a Block Scope.

If(true){

Let age = 30;

}

Console.log(age); //reference error age not defined.

1. Const keyword

Const age = 30;

We can’t modify the age primitive types. For non-primitive types it works differently. (reference type)

1. Variable hosting in es6

age = 30;

console.log(age); //can’t access ‘age’ before initialization.

let age;

function fn (){

age = 30;

}

let age;

fn(); // it will run now.

function scope:- accessing variable inside function (var).

global scope:- entire application js (not in function and block).

block scope:- inside a block we can use (let, const). (if we use var can it can accessing outside also)

1. Arrow functions

Regular function created using function declaration or expressions are constructible and callable.

since regular functions are constructible, they can call by using new key word. however, the arrow

function is only callable and not constructible; arrow function can never be as constructor function.

Handling ‘this’ keyword.

In normal function we get ‘this’ as window object. In arrow function we get ‘this’ as window object.

In arrow function we no need to use ‘call, bind, apply’.

Default parameters

1. Object declaration in es6

Let name = ‘abc’;

Let age = 20;

Let obj = {name, age}; if key and value looking same we can write this way.

1. Rest and spread operator.

Rest operator will use in ‘function’.

Sum(10,20,30);

Function sum (…values){ //parameters convert to array

Console.log(values);

}

Spread operator use to contact object and array. “Shallow copy”

var ggg = [1,2,3];

console.log(typeof ggg.join(',')); // “1,2,3”

1. For-Of and For-in in es6.

For-of -> for array we need to use.

For-in -> use for objects.

1. Destructing arrays and object .

Let number = [1,2,3];

Let [one, two, three] = number;

Let [one, , three] = number; // 1,3. Skip ‘two’ variable.

Let [one, two, three=”default”] = number.

Let [one, …three] = number; // 1, [2,3].

Let [three, two, one] = number; // 1,2,3 // swiping of variables.

Let obj = {name: ‘abc’, age:30};

Let {name, age} = obj;

If we are skipping value does not need to use empty ‘’.

Let {name, age: welcome} = obj; // changing variable “age to welcome”.

1. Babel compiler

It converts code into es5(java script).

Webpack -> it bundles all files into single file.

* Create package.json. npm install –save-dev @babel/core @babel/cli
* In package.json add below script cmd.

script:{

“build”: “babel src -d dist” // npm run build.

}

* We need to create “.babelrc” configuration file.
* Install @babel/preset-env –save-dev.
* And past bellow code in babelrc file.
* {

“presets”: [“@babel/preset-env”]

}

* Now run npm run build. It creates files in dist. folder.

Es6 modules.

<script src=” js/main.js” type=”module”></script> -> in index.html

Export, import

Import {keyvalue} from ‘. /variables.js’;

Webpack

Open source javascript bundler. It converted into a single file. It will not convert es6 to es5.

npm install –save-dev webpack.

npm install –save-dev webpack-cli.

In package.json add below script cmd.

script:{

“build”: “webpack –config webpack.config.js” // npm run build.

}

Create webpack.config.js, file.

Here we need to add configurations. entry path and mode.

In webpack.config.js file

module:{

rules:[

{test:/\.js$, exclude: /node\_modules/, loader:’babel-loader’}

]

}

We need to create babelrc file.

Import and export syntax es6

1. one.js

export const keyvalue = 1000;

export const value4 = 2000;

(or)

export {

keyvalue,

value4

}

two.js

import {keyvalue} from ‘./one.js’

console.log (‘key value’, keyvalue)

1. export default ab; // without this brackets ‘{}’.

Import ab from ‘one.js’;

or

Import a from ‘one.js’;

Or

import ab, {keyvalue} from ‘./one.js’

1. import {keyvalue as key} from ‘./one.js’; //alias
2. import \* as variableData from ‘./one.js’ // export all at a time.

Class and constructor

Class person {

Constructor(name){

this.name = name.

}

hello(){

console.log(‘hello world’, this.name);

}

}

Let p1 = new person(‘abc’);

person.hello();

p1.\_\_proto\_\_ == person.prototype; // true

class inheritance and super in javascript.

Class person {

Constructor(name){

this.name = name.

}

hello(){

console.log(‘hello world’, this.name, this.age);

}

}

class abc extends person{

constructor(age){

super(‘abc’); // when arguments coming we need to call parent using ‘super’.

this.age = age;

}

}

Let childabc = new person(‘abc’,28);

childabc.hello(); //’hello world abc’.

if same method exist in both parent and child. It calls in child method only.

If we want call parent method from child class. if same method exist in both parent and child need to use (super.hello()).

Static methods in javascript classes

If we want call method in class without creating “new object”. We call with “class name” using “static”.

Class person {

Constructor(name){

this.name = name.

}

hello(){

console.log(‘hello world’, this.name, this.age);

}

}

class abc extends person{

static getmessage(msg){

console.log(‘test’, msg);

}

}

person. getmessage(); //test abc’.

getters and setters

Class person {

Constructor(name){

this.\_name = name.

}

get name(){

return this.\_name;

}

set name(name){

this.\_name = name;

}

}

Let childabc = new person(‘abc’);

Console.log(childabc.name);

Extend built-in object like arrays, object to custom classes in es6

class newArraymethod extends Array{

multiplyby2(){

let returnArray = [];

this.forEach(value => returnArray.push(value \* 2));

return returnArray;

}

}

Let childabc = new newArraymethod ();

childabc.push(2);

childabc.push(2);

childabc.multiplyby2() // [4,4]

symbols basics in es6 and how to use symbols  
  
 symbols are also primitive data type. It is a unique identifier.

In object we can’t delete that property, key. When looping the object we will not get symbol.

Let symb = Symbol();

Console.log(symb.toString()); // “symbol()”.

Let symb = Symbol(‘debug’);

Let anotherSymb = Symbol(‘debug’);

Console.log(symb.toString()); // “symbol(debug)”. Typeof symb //”symbol”.

Symb == anotherSymb; // false.

Let obj = {

name:’abc’,

[symbol]: 22

};

console.log(obj); // without symbol we get output.

obj[symbol] // to get symbol.

When looping also we will not get symbol will not come.

Sharing symbols in javascript

Let sym1 = Symbol.for(‘age’);

Let sym2 = Symbol.for(‘age’);

sym1 == sym2 // true.

Let person = {

name:’abc’,

age: 27;

}

function makeAge(){

let symbol = Symbol.for(‘age’);

let person[symbol] = 30;

}

makeAge(); console.log(person[sym1]);

if we have same ‘key’ in person object and symbol it can access both (person[‘age’])

1. iterators in javascript

let number= [1,2,3];

let itfn = number[Symbol.iterator]();

console.log(itfn.next()); // {

done:false, value:1

};

Done became ‘true’. All loop completed. And value as ‘undefined’.

let person = {

name:’leela’, hobbies:[‘sports’, ‘dancing’]

[Symbol.iterator]: function(){

Let hobbies = this.hobbies; I = 0

return{

next:function(){ i++

return {

done: I > hobbies.length ? true : false,

value: hobbies[i]

}

}

}

}

}

for(let hobby of person){

console.log(hobby); // ‘sports’, ‘dancing’

}

1. Promises in javascript. Resolve and reject.

Promise is nothing but it return a promise whenever it return data in asynchronous call.

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

setTimeout(()=>{resolve(‘done’)},2000)

});

promise.then((res)=> {

console.log(res)

}).catch((err)=>{ console.log(err)}};

Promises changing call multiple asynchronous call

function waitAsecond(seconds){

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

setTimeout(()=>{

seconds++;

resolve(seconds)

},1000)

})

};

waitAsecond(0)

.then(waitAsecond)

.then(waitAsecond)

.then((data)=>{ console.log(data)}).catch((err)=>{ console.log(err)}};

## any -> it check which one is success .

## race -> which promise execute first if is success or fail. all -> it will resolve all promises, if any promise fail it throws error. It return array of promises it success.

## allSettled -> I will wait to all promises if it is rejected.

12) object methods

Object.assign(source ,dist). // combine two object.

Object.setPrototypeOf(person, boss); // to set prototype.

“name”.startwith(‘name’)// it check case sensitive. It return true or false.

“name”.endswith(‘name’)// it check case sensitive. It return true or false.

“name”.includes(‘name’)// it check case sensitive. It return true or false.

isNaN(10) // false.

isNaN(“10”) //false in es6 Number.isNaN() we can use.

isFinate() // to check weather it is finite are not.

isInteger() // to check an integer like (1,2,3…)

let arr = Array(5) // [undefined, undefined, undefined, undefined, undefined].

let arr = Array.of(5) // [5].

let arr = Array.from(array ,(val)=> val \*2); // [1,4,6]

arr.fill(100,indexStart,indexEnd) // it replace all values in array with given values.

map:- will create new array of values, and also perform actions

forEach:- it will operate function it not retun any array.

Filter:- based on conduction check and result as new array.

Find:- it checks array if conduction is true are not and return value. if it has duplicate values it ignore remaining it return first match only.

Findindex:- it retuns value index, not find it return -1. it ignore remaining it return first match only.