Contents

[ECMA 6 1](#_Toc506832853)

[Learn ES6 via java script 1](#_Toc506832854)

[Env 1](#_Toc506832855)

[Common rules 1](#_Toc506832856)

[Node.js and ES6 1](#_Toc506832857)

[Features 1](#_Toc506832858)

[Strict mode 1](#_Toc506832859)

[Variable declaration 1](#_Toc506832860)

[Hoisting 2](#_Toc506832861)

[operators: 2](#_Toc506832862)

[Functions: 3](#_Toc506832863)

[Events handling in ES6 4](#_Toc506832864)

[Onclick event 4](#_Toc506832865)

[Onsubmit 4](#_Toc506832866)

[Onmouseover 4](#_Toc506832867)

[Session handling 4](#_Toc506832868)

[DOM 5](#_Toc506832869)

[Objects 5](#_Toc506832870)

[Array 5](#_Toc506832871)

[Maps – intro in ES6 5](#_Toc506832872)

[Sets - in ES6 6](#_Toc506832873)

[classes 6](#_Toc506832874)

[Inheritence – ES6 6](#_Toc506832875)

[Promises and call backs – async programming 6](#_Toc506832876)

[Modules –ES6 powerful feature 8](#_Toc506832877)

[Export 8](#_Toc506832878)

[Import 8](#_Toc506832879)

[Navigator 8](#_Toc506832880)

# ECMA 6

ECMA 6 or ECMA 2015 – specification for scripting languages like java script, jscript, or vb script etc.

ES6 provides lot of improvement in scripting standards – hugely inspired from languages from java, ruby, python etc.

# Learn ES6 via java script

Java script is Object oriented language. All are objects internally. Objects contains functions and variables.

## Env

VS code IDE or Brackets

Node js is installed which has in-built js engine so we can run the js script without browser

## Common rules

* Case sensitive – js engine recognizes cases in variables
* Semicolons are optional
* ES6 ignores whitespaces, tabs, etc
* Comments are like \\ or \\*\*\

# Node.js and ES6

3 feature groups of ES6 for node.

1. Feature groups – node considers it as stable and enables by default
2. Staged group – almost completed but not yet stable by node v8 engine.
3. In progress – only for tested mode

# Features

## Strict mode

The Strict Mode imposes a layer of constraint on JavaScript. It makes several changes to normal JavaScript semantics. the entire code runs as a constrained variant of JavaScript.

Eg: v = 15

function f1() {

"use strict";

var v = "Hi! I'm a strict mode script!";

}

## Variable declaration

Traditional : var variable\_name

ES6 introduces -> let and const

"use strict"

function test() {

var num = 100

console.log("value of num in test() "+num) {

console.log("Inner Block begins")

let num = 200

console.log("value of num : "+num)

}

}

test()

const x = 10

x = 12 // will result in an error!! Cant reassign

## Hoisting

JavaScript engine, by default, moves declarations to the top. This feature is termed as hoisting. However, the concept of hoisting does **not apply to scripts** that are **run in the Strict Mode.**

Variable Hosting:

var main = function() {

//javascript engine internally makes this variable as

//var x; // x is hoisted to function scope. But it is not initialised

for(var x = 0;x<5;x++) {

console.log(x);

}

console.log("x can be accessed outside the block scope x value is :"+x);

console.log('x is hoisted to the function scope');

}

main();

## operators:

traditional :

var num = 12

console.log(typeof num); //output: number

## Functions:

Simple function

//define a function

function test() {

console.log("function called")

}

//call the function

test()

Ananymous function

var func = function(x,y){ return x\*y };

function product() {

var result;

result = func(10,20);

console.log("The product : "+result)

}

product()

Function constructor:

var func = new Function("x", "y", "return x\*y;"); // defined analymous function

function product() {

var result;

result = func(10,20);

console.log("The product : "+result)

}

product()

Lambda function:

Lambda refers to anonymous functions in programming.

var foo = (x)=>10+x

console.log(foo(10))

2. var msg = ()=> {

console.log("function invoked")

}

msg()

3. var msg = x=> { // optional paranthesis

console.log(x)

}

# Events handling in ES6

JavaScript is meant to add interactivity to your pages. JavaScript does this using a mechanism using events. Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events that can trigger JavaScript Code.

## Onclick event

<input type = "button" onclick = "sayHello()"

## Onsubmit

<form method = "POST" action = "t.cgi" onsubmit = "return validate()">

## Onmouseover

<div onmouseover = "over()" onmouseout = "out()">

There are n number of event handlers available in JS by ES6. Learn yourself.

# Session handling

Session handling done by cookie in general. Server remembers the cookie value for earlier page.

Cookie fields : expires, domain, path, secure, name=value

document.cookie = "name = " + cookievalue;

# DOM

HTML document that is displayed in that window. The document object has various properties that refer to other objects which allow access to and modification of the document content.

Types: legacy DOM, W3C Dom, IE4 DOM - > now W3C is the standard

# Objects

Variable and methods. All js are objects only.

Var a = {} // object declaration

new Object() // object declaration by new operator

var person = {

firstname:"Tom",

lastname:"Hanks",

func:function(){return "Hello!!"},

};

## Array

Special type of object to store list of values

var array\_name; //declaration

array\_name = [val1,val2,valn..] //initialization

OR

var array\_name = [val1,val2…valn]

or

var arr\_names = new Array(4)

console.log(alphas[0]);

various methods of array objects helps to do numerous things.

## Maps – intro in ES6

var map = new Map();

map.set('name','Tutorial Point');

map.get('name'); // Tutorial point

console.log(map.has("1")); //false

## Sets - in ES6

var set = new Set(['a','b','c','d','e']);

var iterator = set.entries();

console.log(iterator.next())

## classes

var Polygon = class Polygon { // declaring the classes

constructor(height, width) {

this.height = height;

this.width = width;

}

}

//initializing the object from class

Var obj = new Polygon();

## Inheritence – ES6

class Circle extends Shape {

disp() {

console.log("Area of the circle: "+this.Area)

}

}

## Promises and call backs – async programming

Promises are a clean way to implement async programming in JavaScript (ES6 new feature). Prior to promises, Callbacks were used to implement async programming.

Sync method – blocks the other method invocation.

**Callbacks – old method**

<script>

function notifyAll(fnSms, fnEmail) {

setTimeout(function() {

console.log('starting notification process');

fnSms();

fnEmail();

}, 2000);

}

notifyAll(function() {

console.log("Sms send ..");

},

function() {

console.log("email send ..");

});

console.log("End of script"); //executes first or not blocked by others

</script>

**Promises - clean way to asyn or non-blocking calls**

function getSum(n1, n2) { -- step 1 write a method uses promise for async

varisAnyNegative = function() {

return n1 < 0 || n2 < 0;

}

var promise = new Promise(function(resolve, reject) { -- step2 – attach promise with fn getsum and write positive scenario and negative scenario. Callback reject for –ve case and resolve for +ve case

if (isAnyNegative()) {

reject(Error("Negatives not supported")); -- handling –ve case

}

resolve(n1 + n2); -- handling +ve case

});

return promise;

};

getSum(5, 6) --step3: write a calling method which accepts promise object. Handle +ve and –ve case

.then(function(result) {

console.log(result);

},

function(error) {

console.log(error);

});

## Modules –ES6 powerful feature

 parts of JavaScript code need to be reused. ES6 comes to your rescue with the concept of Modules.

This feature requires transpilers to convert to ES 5 from ES6.

### Export

Consider a JavaScript file, Message.js, with a method printMsg() in it. To be able to reuse the functionality provided by this method, include the following in the Message.js file −

Export default printMsg or export {printMsg,usrMsg….}

### Import

The script file that intends to consume the function, say User.js, will have to import the function from the Message module by including the following −

import printMsg from './Message.js' or import {ele1, ele2 …} from './Message.js'

**steps:**

>npm install -g es6-module-transpiler

>compile-modules convert -I <scripts dir> -o <out dir> Message\_module.js

consume\_module.js -format commonjs

## Navigator

JavaScript navigator object includes a child object called plugins. This object is an array, with one entry for each plug-in installed on the browser.

Eg1: checking for plugin

for (i = 0; i<navigator.plugins.length; i++) {

document.write(navigator.plugins[i].name);