**ES6 tutorial (Birth 2015)**

ECMA script

European Computer Manufacturers Association (**ECMA**)

Why you need ES6 ?

It is next level of JavaScript mostly used in front end JS frameworks like React, Angular, Vue etc…

I’m an front end engineer but I don’t have much knowledge in ES6, do you think still I need to get deep into it ?

Of course not, but in some cases you can solve the queries at ease if you know ES6.

*Note :- if you want to learn other JS frameworks at ease then this tutorial will be useful for you.*

Happy coding…

1. Choosing code editors & installing node.js

* It’s really up to you, you can choose any code editor
* If you are confused, use either visual studio or atom
* My personal choice is VS
* Also click here to get installed node.js

1. How JS transformed to ES6
   * JavaScript is small, lightweight, object-oriented language
   * JavaScript can handle client as well as server side
   * Client-side by accessing DOM(Document Object Model)
   * Server-side by accessing Servers and connecting databases

The **Document Object Model (DOM)**gives full access from JavaScript to dynamically create HTML with specific methods that modify the object representations of HTML elements

DOM is an interface between HTML and XML documents read by browser

1. What is new in Ecma Script ?

* Arrow functions, enhanced collections, generators
* Assignments in ES6

**Module 1 – Basics (Level – Beginner)**

**Le 1) Variables in ES6**

|  |  |
| --- | --- |
| **In Java Script** | **In ES6** |
| var limit = 100 | let limit = 100 |
|  | const limit = 100 |

Instead of ~~var~~, **let** keyword is used

const keyword is used for unchanged variables (if you try to replace values of constant variable you will hit an error)

**Le 2) Blocks**

What is Blocks?

Anything that provides the value of the variables with its limit is called as blocks.

Example 1:

let a = 5 global scope

let b = 4

{

let a =3 local scope

console.log(a,b) output=> a = 3, b =4

}

console.log(a) output=> a=5

In global scope while printing ‘a’ it gives the correct value, then what is wrong with local scope

In local scope it prints b -> which is global value

Local scope first checks the value within itself and if doesn’t found the value it searches globally

Example 2:

let a = 5

{

let b = 4

}

console.log (a,b) output => result’s in error because, you can’t print local variable outside it’s scope

*Note : - we can access global variable inside local scope, but we can’t access local variable in global scope*

**Le 3) $ operator**

$ operator is used to join the variables

let a = ‘happy’

let b = a+’ learning’ (or) let b = a.concat(‘ learning’)

console.log( b ) output=> happy learning

let b = `$a learning` Same output

if we use + or concat lot of spaces will be wasted, it will be like you will be using lot of spaces

instead you can use $ opertaor as join operator

note :- while using $ operator you should use **grave accent** ( **`** ) not **apostrophe ( ‘ )**

**example : `$a learning` not ‘$a learning’**

**Le 4) Operators & Destructuring Assignment**

1. Spread Operator: { … } => 3 dots
2. Rest Parameter: function(…) {}

Destructuring Assignment

Simplifies extracting data on array and objects into distinct variables

1. **Spread Operator with example**

Consider we have 2 arrays and we want to join them togther and print as output

Solution :- Let’s try the possible ways

let a = [1,2,3]

let b = [4,5,6]

1. let c= [a,b]

console.log( c ) output=> [Array(3), Array(3)]

it’s displayed in object form

ii let a = [1,2,3]

let b = [a,4,5,6]

console.log(b) output=>[Array(3), 4, 5, 6]

and again we can conclude that if we are using raw array variable in another array it may result as Array with It’s length as shown above

Here comes the superman operator Spread to solve the problem

1. let a = [1,2,3]

let b = […a,4,5,6]

console.log(b) output=> [1, 2, 3, 4, 5, 6]

1. let a = [1,2,3]

let b = [4,5,6]

let c = […a,…b]

console.log(c) output=> [1, 2, 3, 4, 5, 6]

**ii) Rest Parameter**

It is opposite to spread operator

If we want to send an array/ list of items to a function, we can use Rest parameter

1. function new(a)

{

console.log(a)

}

new(1,2,3) output=> 1

here we will get the first element of the group of elements

while passing (1,2,3) to function new(a = 1,2,3), a = 1 where 2 and 3 will be rejected automatically

1. function new(…a)

{

console.log(a)

}

new(1,2,3) output=>1,2,3

**Destructuring Assignments**

let a = [1,2,3]

let x = a[0]

let y = a[1]

console.log(x,y) output=> 1,2

This is the typical example of the traditional way of designing assignments

let a = [1,2,3]

let [x,y] = a

console.log(x,y) output=> 1,2

Modern way of destructuring assignments

Consider in array ‘a’ we want only 1,3 element

let a = [1,2,3]

let [x, ,y] = a

console.log(x,y) output=> 1,3

by leaving empty space inbetween x and y it will neglate the second element

let [x, ,y] =a

x=1,

‘ ’=2,

y=3

where 2 is neglated since it is assigned to an empty variable

In Complex Scenario

let detail = {name:’Mars’, id:1}

let name = detail.name

let id = detail.id

console.log(name,id) output=> Mars 1

Traditional way

let detail = {name:’Mars’,id:1}

let {name,id} = detail

console.log(name,id) Same output Modern way

**Module 2 – New Functions (Level – Beginner)**

**Arrow functions**

It is similar to normal function but difference with syntax

|  |  |
| --- | --- |
| **JavaScript** | **ES6** |
| function()  {  //…  } | ( ) =>  {  //…  } |

Arrow functions ( ) => can be anonymous (function without name)

In default Arrow function is anonymous.

Anonymous arrow function

Syntax

( ) => {

…//

}

Using Anonymous in onClick event

<button onClick = { ( ) => {concole.log(“ I’m anonymous function “)}>

Anonymous Button

</button>

Tell me whether the function is Anonymous or not

let val = ( ) => {

console.log( ‘ print ‘)

}

val()

yes no

if yes : - yes it is anonymous because the function doesn’t have name

if no :- it’s wrong, you may get confused by seeing let val = (), here let val is variable and ( ) = > is the function, where the empty(anonymous) function is stored in variable (val)

**Map function**

Map function is also called as loop function which is similar to for each function.

Printing number in an array

let value = [1,2,3]

let printValue = value.map ( (n) = > n )

console.log(printValue) output => [1,2,3]

let’s decode the map function

value . map ( (n) => n )

value = variable which contains array

map = loop function

n = each number is stored as n

What happens in map function ?

let printValue = value [1,2,3] . map ( (n[1]) => n[1] )

[1] = [1,2,3] . map ( (1) => 1)

let printValue = value [1,2,3] . map ( (n[2]) => n[2] )

[1,2] = [1,2,3] . map ( (2) => 2)

let printValue = value [1,2,3] . map ( (n[3]) => n[3] )

[1,2,3] = [1,2,3] . map ( (3) => 3)

When map function doesn’t see the next value, it returns stops looping and return the value produced by it.

Example : - Print the value by adding 2 to each individual value.

let value = [1,2,3]

let addTwo = value.map ( (n) => n +2 )

console.log(addTwo) output => [3,4,5]

let addTwo = value [1,2,3] . map ( (n[1]) => n[1 + 2] )

[3] = [1,2,3] . map ( (1) => 1 + 2)

let addTwo = value [1,2,3] . map ( (n[2]) => n[2 + 2] )

[3,4] = [1,2,3] . map ( (2) => 2 + 2)

let addTwo = value [1,2,3] . map ( (n[3]) => n[3 + 2] )

[3,4,5] = [1,2,3] . map ( (3) => 3 + 2)

**Filter Function**

As the name says we are filtering the values based on the given condition.

For example : - the values be [1,2,3,4,5]. I need the values which is greater than 3.

This will be example of filter function. Odd values, even values, divided by 2, divided by 3 are some of the examples for filter function.

Example 1 :- Print all odd and even numbers.

let value = [1,2,3,4,5]

let oddNumber = value.filter ( (n) => n%2 ! = = 0 ) output=> [1, 3, 5]

let evenNumber = value.filter ( (n) => n%2 = = = 0 ) output=> [2, 4]

console.log(`odd number: $oddNumber, even number $evenNumber`

Example 2 :- Print the numbers which is greater than and equal to 3.

let value = [1,2,3,4,5]

let greaterNumber = value.filter ( (n) => n >= 3 ) output=> [3, 4, 5]

console.log(`greater than nd equal to 3 $greaterNumber`)

Example 3 :- Print the numbers which is greater than 2 and lesser than 5.

let value = [1,2,3,4,5]

let greaterNumber = value.filter ( (n) => n >2 && n<5 ) output=> [3, 4]

console.log(`greater than 2 nd lesser than 5 $greaterNumber`)

**Helper Methods**

***.repeat( )***

Repeats the value for given n times

for example :-

let truth = `this session is b{$o.repeat(10)} ring’

(or)

let truth = ‘this session is b’ +”o”.repeat(10)+ ‘ring’

***.startsWith( ) & .endsWith( )***

It is used to check our starting/ending value which matches with existing value and returns true if matches.

let value = ‘pancake’

value.startsWith(‘pan’) output=> true

value.endsWith(‘cake’) output=> true

**Import and Export methods / Modules**

It enables you to import or export the modules from one to another.

Create two files names.js and output.js

The file structure will be

src >

names.js

output.js

names.js

const names = [‘abc’, ‘def’ , ‘ghi’]

export {names}

output.js

import {names} from ‘./output’;

console.log(names)

Example :- Creating our own functions using module concepts.

File structure will be

src>

easyMaths.js

solve.js

easyMaths.js

const add = ( a, b ) => {

return a + b

}

const minus = ( a, b ) => {

return a - b

}

export {add, minus}

solve.js

import {add, minus} from ‘./easyMaths’

let n = add(1,2)

let m = minus(5,2)

console.log(`addition ${n} , subtraction ${m} `**)**

**Classes**

Package where the objects, methods are kept and reused.

Class firstGrade{

Id;

Name;

Section;

}

Let’s consider this simple example.

Let me compare the class with Grades in schools.

Assuming firstGrade as class, if you want to admit a student into it, he must have Id, Name and Section. This template can be reused for n students.

Example

class firstGrade {

constructor (id, name, section){

this.id = id;

this.name = name;

this.section = section;

}

}

let Harry = new firstGrade(1,’Harry’,’A’)

let Tom = new firstGrade(2,’Tom’,’A’)

let Jerry = new firstGrade(3,’Jerry’,’B’)

Likewise we can add n students list by using template firstGrade.

Parameter values can be anything but while calling you should call the variable name inside the constructor.

constructor (a,b,c){

this.id = a;

this.name = b;

this.section = c;

}

let Tim = new firstGrade(4,’Tim’,’C’)

console.log(Tim.name, Tim.section)

Using function inside class

class firstGrade {

constructor(id,name,section){

this.id = id;

this.name = name;

this.section = section;

}

details( ){

console.log (this.id, this.name, this.section)

}

}

let Tom = new firstGrade(1,’Tom’,’A’)

let Jerry = new firstGrade(2,’Jerry’,’B’)

Tom.details( ) output=> 1 Tom A

Jerry.details( ) output=> 2 Jerry B

Inheritance

Extending assets to child class or giving child class permission to access the properties of parent class. We will be using ‘super’ method which intimate the parent class to get accessed.

class Grade

constructor (id, name, section){

this.id = id

this.name = name

this.section = section

}

details( ){

console.log (this.id, this.name, this.section)

} }

class firstGrade extends Grade{

constructor (id, name, section){

super(id, name, section)}

details ( ) { console.log (this.id, this.name, this.section) }

}

class secondGrade extends Grade{

constructor (id, name, section){

super(id, name, section)}

details ( ) { console.log (this.id, this.name, this.section) }

}

let Man = new firstGrade (1,’Man’,’A’)

let Wan = new secondGrade(2,’Wan’,’B’)

Like wise you can extend upto n number of classes using inheritance.

You can also add additional parameter to inheritance.

class highGrade extends Grade{

constructor (id, name, section, marks){

super(id, name, section)

this.marks = marks;

}

details ( ) { console.log( this.id, this.name, this.section, this.marks) }

}

Using Static method in class

Static method is used to call the call the function without creating a new instances.

class Grade{

static add(a,b)

{

return a+b

}

static add(a,b)

{

return a+b

}

static sub(a,b)

{

return a-b

}

}

let a = Grade.add(1,2)

let b = Grade.sub(1,2)

Using static method there is no need of creating new object. It also has disadvantages, it can’t be overload or override which doesn’t mean if we try to overload or override it results in error.

Prototypes

JavaScript is based on prototypal inheritance model.

Data Structures

In JS data structures are most used in solving problems like searching, sorting with optimization.

1)SET

A special function which is allergic to duplicates.

Which mean it can’t take duplicates into it.

In other words, a function which doesn’t allow/ removes duplicates if added to it.

let variable = new Set( )

variable.add(1)

It can add numbers

variable.add(‘string’)

It can add string

variable.add({id:1, name: ‘hai’})

It can add objects

console.log(variable) output=> { 1, ‘string’, { id: 1, name: ‘hai’ } }

Example :-

let a = new Set( )

a.add(1)

a.add(1)

a.add(2)

a.add(3)

a.add(2)

console.log(a) output =>{1, 2, 3}

console.log(a.size) output =>3

note : we are adding 5 values say 1 1 2 3 2 but while printing it prints only 3 values by removing duplicates which means the size is also 3 not 5.

To check whether it has the value or not.

console.log( a.has(1) ) output =>true

console.log( a.has(5) ) output =>false

To delete the value.

a.delete(1)

console.log(a) output => {2,3}

To convert array into set.

let arr = [1, 2, 3, 4, 5]

let set\_ = new Set(arr)

console.log(set\_) output => {1, 2, 3, 4, 5}

To convert set into array

let newArr = […set\_]

Maps

The diference between ‘map’ and ‘Map’

map = used to iterate the values

Map = used to key pair the values

Map data structure contains key and value, where we can get value using unique key.

let a = new Map( )

a.set(1,’Ram’)

a.set(2, ‘Rom’)

console.log(a) output=> {1 => ‘Ram’, 2 => ‘Rom’}

Finding value using key.

let a = new Map( )

a.set(1, ’Ram’)

a.set(2, ‘Rom’)

a.set(3, ’Cam’)

a.set(4, ‘Tom’)

let k = 3

for ( let [key, value] in a.entries( ) ){

if ( key === k ){

console.log( value ) output=> ‘Cam’

}

}

**Closure**

A function through which we can access private state is called as closure.

Let’s assume that we have function like this.

let hidden = ( ) => {

let node = ‘101101’

}

console.log(node) output=> Result’s in error since node is not declared globally.

let hidden = ( ) => {

let node = ‘101101’

let inner = ( ) =>{

console.log(node)

}

return inner;

}

const secret = hidden( )

secret( )

We are creating an inner function where we the **node** variable is accessed.

We are storing **hidden( ) to constant** to prevent immortal of function.

If  JavaScript did not have closures, then more state would have to be passed between functions explicitly, making parameter lists longer and code noisier.

So, if you want a function to always have access to a private piece of state, you can use a closure.

**Function Factories**

Using Closures we can create Function Factories, it takes one or more argument and returns new function.

It calls a function through variable and other function called through newly created variable.

(or)

Function factories create functions based on returning inner functions that manipulate its own arguments and the arguments of the outer function.

let add = (a) = >{

let plus = (b) =>{

return a + b

}

return plus

}

let x = add( 1 )

let y = x( 2 )

console.log( y ) output=> 3

we can rewrite without inner function

let add = (a) =>{

return (b) = > { a+b }

}

let x = add(1)

let y = x(2)

console.log(y) output=> 3

Generator

Generator is a function which controls the code execution.

It can pause and resume the line of execution.

In a normal function, there is only **one** entry point: the invocation of the function itself. A **generator** allows **you** to pause the execution of a function and resume it later. **Generators are** useful when dealing with iterators and **can** simplify the asynchronous nature of Javascript.

syntax

function\* function\_name( ){

yield ‘value’;

yield ‘value’;

}

function\_name( ).next( ).value

function\* denotes declaration of Generators

yield is used to pause

next() is used to resume

function \* gen( ){

yield ‘x’;

yield ‘y’;

}

let play = gen( )

console.log(play.next( ).value)

output => x

console.log(play.next( ).value)

output=> x y

Controlling Generators

function\* gen( ){

let i = 0

while(true) {

i = i+1

if( i%5 = = = 0 ){

yield i

}

}

}

let play = gen( )

console.log(c.next( ).value)

console.log(c.next( ).value)

output => 5 10

We can pause/ stop using break but we can’t resume once after it’s done.

Asynchronous Programming

Synchronous Programs

Run in sequence without blocking

Asynchronous Programs

Run by firing an event

In **programming**, **synchronous** operations block instructions until the task is completed, while **asynchronous** operations can execute without blocking other operations. **Asynchronous** operations are generally completed by firing an event or by calling a provided callback function

Example : UI and browser are asynchronous

It helps in accessing databases, server and API interaction

**Promise**

Promise is an object which is used in asynchronous functions.

It has 3 stages

Fulfilled : resolve( )

Rejected : reject( )

Pending : not yet fulfilled or rejected

If a promise is resolved or rejected it moved into settled state.

A promise will be settled if it is not pending.

Once promise is settled it can’t be recalled.

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

resolve (‘solved’)

}

).then (res => console.log(res)) //fulfilled

.catch(res => console.log(res)) // rejected

Output => solved

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

reject(‘rejected’)

}

).then (res => console.log(res)) //fulfilled

.catch(res => console.log(res)) // rejected

Output => rejected

CRUD operations using API call

CRUD mean Create Read Update and Delete.

CRUD operation is key king in each app.

An app contains 85% of CRUD operations and 15% of others.

CRUD is Diamond among Golds.

There are more libraries used for crud operations.

Axios is one of the best library.

Four operations

GET - display

POST - add

PUT - update

DELETE – remove

**GET**

const [items,setItems] = useState([ ])

axios.get('https://jsonplaceholder.typicode.com/posts')

.then( res => setItems (res.data) )

.catch(err => (console.log(err)) )

{ items.map (item => (<div key ={item.id}>

<p> {item.id} </p>

<p> {item.title} </p>

<p> {item.body} </p>

</div>) }

**Mini Project ….**

**Getting Google books api and posting**

import React, { useState, useEffect } from 'react'

import axios from 'axios'

import {Button,Form,FormGroup, Card, CardDeck} from 'react-bootstrap'

import { CardText } from 'react-bootstrap/Card'

export default function Vote(){

const [val,setVal] = useState(0)

const [records,setItem] = useState([])

const [link,setLink] = useState()

const search = (e) =>{

    e.preventDefault()

    axios.get('https://www.googleapis.com/books/v1/volumes?q='+val)

    .then(res=>setItem([res.data])).catch(err=>setItem(['item not found']))

}

const key = (e) =>{

    if(e.key==='Enter'){

        search(e)

    }

}

    return(<div> <br/> <br/> <br/>

<Form onSubmit={search}>

  <FormGroup controlId="formBasicEmail" >

    <Form.Label>Search Books</Form.Label>

    <Form.Control type="text" placeholder="Enter Book Name" onChange={(e)=>setVal(e.target.value)} onKeyPress={key} />

  <Button variant="primary" type="submit" className="glyphicon glyphicon-search" value="submit">

    &nbsp;Search

  </Button>

    <Form.Text className="text-muted">

      Find your favourite books here

    </Form.Text>

  </FormGroup>

  </Form>

    <div className="container-fluid " >

  <CardDeck className="col-auto mb-3" style={{height:400, width:400}}>

        <Card>

        <Card.Body>

    { records.map(record=>(<div1 key={record.kind}>{record.kind}

    {record.items.map(sub=>(<div1 key={sub.id}>

    {[sub.volumeInfo].map(sub1=>(<div1 key={sub1.title} >  <Card.Title>{sub1.title}</Card.Title>

    <Card.Text>

    <br/>{sub1.publisher}<br/>{sub1.publishedDate}<br/>

    {[sub1.imageLinks].map(sub2=>(<div1 key={sub1.publisher}>   <Card.Img variant="top" src= {sub2.thumbnail}  style={{height:200, width:200}} /> </div1>))}

    <Card.Footer>

    <a href={sub1.previewLink} target="\_blank">preview</a> <br/>

     country :{sub.saleInfo.country}<br/> <a href={sub.saleInfo.buyLink} value="buy" target="\_blank">link</a>

     </Card.Footer>     </Card.Text> <hr/>

    </div1>))}

     </div1>))}

    </div1>))}

    </Card.Body>

    </Card>

 </CardDeck>

 </div>

    </div>)

}