# **JavaScript Es6 Notes**

1. **For Each**
   1. Used to run each item in an array one at a time

<script>

var numbers = [1, 2, 3, 4, 5];

var sum = 0;

numbers.forEach(function(number) {

sum += number;

});

console.log(sum);

var output = document.getElementById('output');

output.innerHTML = sum;

</script>

1. **Map Helper**
   1. Used to create an array from another array

<script>

const numbers =[1,2,3,4,5,6];

let doubleNumbers = [];

doubleNumbers = numbers.map(function(number){

return number \*3;

});

console.log(doubleNumbers);

</script>

1. **Filter Helper**
   1. Used to Find and Item in an object

<script>

var products = [{

name: 'cucumber',

type: 'vegetable'

},

{

name: 'banana',

type: 'fruit'

},

{

name: 'celery',

type: 'vegetable'

},

{

name: 'orange',

type: 'fruit'

}

];

var myFilter = products.filter(function (product) {

return product.type == "fruit";

});

console.log(myFilter);

</script>

1. **Find Helper**
   1. Used to **finds ONLY the 1st element that matches** it does not continue so for example if there were 2 Carmen’s the find will only find the 1st

<script>

// To create an array from another array

var output = document.getElementById('output');

var output1 = document.getElementById('output1');

var users = [{

name: 'Henry'

},

{

name: 'Carmen'

},

{

name: 'Ethan'

}

];

// Note Find helper only finds the 1st element that matches it does not continue so for example if there were 2 Carmen’s the find will only find the 1st

var userExists = users.find(function (user) {

return user.name === 'Carmen';

});

console.log(userExists);

output.innerHTML = userExists.name;

</script>

1. **Every and Some Helper**
   1. The every is only true if all conditions are true in this example if all computers have more than 16 ram (Will return true or false)
   2. The some is only true if some conditions are true in this example if some computers have more than 16 ram (Will return true or false)

<script>

var output = document.getElementById('output');

var output1 = document.getElementById('output1');

var computers = [

{name: 'Apple', ram: 24},

{name: 'Compaq', ram: 4},

{name: 'Acer', ram: 32}

];

// The every is only true if all conditions are true in this exapmle if all computers have more then 16 ram

var allComputersCanRunProgram = computers.every(function(computer){

return computer.ram >16;

});

// The some is only true if some conditions are true in this example if some computers have more then 16 ram

var someComputersCanRunProgram =computers.some(function(computer){

return computer.ram >16;

});

console.log(allComputersCanRunProgram);

console.log(someComputersCanRunProgram);

</script>

1. **Reduce Helper**
   1. Used to Find the elements that match inside an array of strings //
      1. Note at end of function you can add either another array or a number as a starting point

<script>

var primaryColors = [{

color: 'red',

brand: 'bmw'},

{

color: 'white',

brand: 'ferrari'

},

{

color: 'blue',

brand: 'mazda'

},

];

var myNewArray = primaryColors.reduce(function (previous, primaryColor) {

previous.push(primaryColor.color);

return previous;

}, []);

var myBrands = primaryColors.reduce(function (previous, primaryColor) {

previous.push(primaryColor.brand);

return previous;

}, []);

console.log(myNewArray);

console.log(myBrands);

</script>

1. **Enhanced object Literals**
   1. Used to information inside and array of objects

<script>

// To create an array from another array

var output1 = document.getElementById('output1');

var output2 = document.getElementById('output2');

// IMPORTANT

// Note : All this code

// can be slightly reduce whit this code

function createBookShop(inventory) {

return {

inventory,

// inventory: inventory,

inventoryValue() {

//inventoryValue: function () {

return this.inventory.reduce((total, book) => total + book.price, 0);

},

priceForTitle (title) {

//priceForTitle: function (title) {

return this.inventory.find(book => book.title === title).price;

}

};

}

const inventory = [

{ title: 'Harry Potter', price: 10},

{title: 'Eloquent Javascript', price: 25},

{title: 'The alquimist',price: 14}

];

const bookShop = createBookShop(inventory);

console.log(bookShop.inventoryValue());

console.log(bookShop.priceForTitle('The alquimist'));

output1.innerHTML = `total book inventory is: ${bookShop.inventoryValue()}`;

output2.innerHTML = `Book cost is:${ bookShop.priceForTitle('The alquimist')}`;

</script>

### **RESULT IS:**

total book inventory is: 49

Book cost is:14

1. **Rest and Spread Operator**
   1. The spread operator ... allows you to include the variables without having to specify the amount and you can do in directly inside a new array or in a function argument.

<script>

// NOte

// the spread operator ... allows you to include the variables without having to specify the amount and you can do in directly inside a new array or in a function argument.

const defaultColors = ['red', 'green'];

const userFavColors = ['orange', 'yellow'];

const fallColors = ['fire red', 'fall orange'];

let newColors = ['green', 'blue', ...defaultColors, ...userFavColors, ...fallColors];

console.log(newColors);

</script>

1. **Destructing**
   1. Used to makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

<script>

var expense = {

type: 'business',

amount: '$45 USD'

};

// ES5 way to move the objects property into a variable

var type = expense.type;

var amount = expense.amount;

console.log(type);

console.log(amount);

console.log("------ the ES6 way -- Below -----");

// ES6 way to move the objects property into a variable

var expense1 = {

type1: 'business',

amount1: '$45 USD'

};

// const {type1} = expense1 ;

// const {amount1} = expense1;

// you an conbine the 2 lines avove to creat both at once

const {type1, amount1 } = expense1;

console.log(type1);

console.log(amount1);

</script>

1. **Generators** 
   1. Allow you to run a function on 2 different groups of objects

<script>

const testingTeam = {

lead: 'Amanda',

tester: 'Bill',

[Symbol.iterator]: function\* () {

yield this.lead;

yield this.tester;

}

};

const engineeringTeam = {

testingTeam: testingTeam,

// FOr line avove you can avoid repeatina and just type:

// testingTeam,

size: 3,

department: 'Engineering',

lead: 'Jill',

manager: 'Alex',

engineer: 'Dave',

[Symbol.iterator]: function\* () {

yield this.lead;

yield this.manager;

yield this.engineer;

yield\* this.testingTeam;

}

};

const names = [];

for (let name of engineeringTeam) {

names.push(name);

}

console.log(names);

</script>

1. **Promises and fetch**
   1. Used to access data and the data is receive then you can take one pat and of is not then u can take another path
      1. Promises can be resolve or rejected,
         1. if resolve then use. then
         2. if reject then use. catch

<script>

//Note

// Promises can be resolve or rejected

//flow if resolve then use .then

// flow if reject then use catch

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

//Note

// from resolve(); to reject(); to test

resolve();

//reject();

});

promise

.then(() => {

console.log("Finally Finished");

})

.then(() => {

console.log("I alson ran !!!");

})

.catch(() => {

console.log("uh ohh !!!");

})

</script>