**Introduction**

All about the new features added to the javascript language by the ECMA script standard (ES6 specification).

* ECMAScript is a Standard for scripting languages
* ECMA means European Computer Manufacturer’s Association

**Traceur**

Google project which can run in a browser thereby allowing to use most of the ES6 features.

**traceur.js :** parse ES6 syntax to ES5

**BrowserSystem.js :**

**bootstrap.js :** single line of code to instantiate traceur and tell it to run on this webpage

**Google CDN**

<script src="https://google.github.io/traceur-compiler/bin/traceur.js"></script>

<script src="https://google.github.io/traceur-compiler/bin/BrowserSystem.js"></script>

<script src="https://google.github.io/traceur-compiler/src/bootstrap.js"></script>

Once we have this code in place we can open <script> and write some ES6 code in it.

<script type="module">  
 let add = (x,y) => x+y;  
 document.write(add(5,3));  
</script>

**Browser Compatibility**

Just google **es6 compatibility table** and you will find it.

**Variables and Parameters**

* **let**
  + Used to declare variables, unlike **var** keyword which has just local / global scope, **let** has something called a block scope.
  + function check\_let(flag) {  
     if(flag) {  
     let x = "Hello World";  
     return x;  
     }  
     }  
     document.write(check\_let(true));
  + In the above function if we move the **x** outside the if block, it will throw an error as the variable x is only available inside the if block.
* **const**
  + Create and initialise a readonly variable, a variable that will hold a constant value that will never change.
  + function check\_let(flag) {  
     const MAX\_SIZE = 10;  
     //MAX\_SIZE = 12; // throws readonly error  
       
     const x = 10;  
     if(flag) {  
     var x = 12; // throws duplicate declaration error, if you move const x out of function block it will work fine.  
     return x;  
     return MAX\_SIZE;  
     }  
     v}  
     document.write(check\_let(true));
* **Destructuring**
  + JavaScript expression that makes it possible to extract data from arrays or objects into distinct variables.
  + **Array**  
     var numb = [2, 3];  
     var [x, y] = numb; // storing values from array to variables x , y, [] is because numb is an array, [x,y] is not an array, is just an expression  
       
     var numb = function() {  
     return [1, 2, 3];  
     };  
     var [,a,b] = numb(); // can use comma to ignore values  
     alert(x+', '+y);  
     alert(a+', '+b);
  + **Object**  
     let retObj = function() {  
     return {  
     firstname:'Arun',   
     location:'US',   
     email:'arun.gopan@marlabs.com',  
     social:{  
     facebook:'fb',  
     twitter:'tweet',  
     google:'search'  
     }  
     };  
     }
    - **for different variable names**  
       let {  
       firstname : first\_name,  
       location:loc,  
       email:email\_id,  
       social:{facebook:fb\_var},  
       social:{twitter:tw\_var},  
       social:{google:go\_var}  
       } = retObj();  
       alert(first\_name+', '+loc+', '+email\_id+', '+fb\_var+', '+tw\_var+', '+go\_var);
    - **for same variable names as the object**  
       let {  
       firstname,  
       location,  
       email,  
       social:{facebook},  
       social:{twitter},  
       social:{google}  
       } = retObj();  
       alert(firstname+', '+location+', '+email+', '+facebook+', '+twitter+', '+google);
* **Default parameters**
  + Allows you to specify default values for functions in javascript
    - **default way of specifying default values**  
       function alert\_me(name) {  
       name = name || "Arun";  
       alert(name);  
       }  
       alert\_me();
    - **new way of specifying default values**  
       function alert\_me(name="Arun") {  
       alert(name);  
       }  
       alert\_me('Marlabs');
* **Template literals**
  + Used to concatenate strings in a better way.
  + var name = 'Arun',  
     location = 'United States';  
     var res = 'My name is '+name+', and i am currently in '+location;  
     var res2 = `My nams is ${name}, and i am currently in ${location}`;  
     alert(res);  
     alert(res2);

**Classes**

* **Defining a class**
  + class Users {  
     alertMe(name="Arun") {  
     alert(name);  
     }  
     alertMyLoc(loc="India") {  
     alert(loc);  
     }  
     }  
    var user1 = new Users();  
    user1.alertMe('Marlabs');  
    user1.alertMyLoc();
  + **behind the scenes it is using the same protoype method.**  
    Users.prototype.alertMe();
* **Constructor**
  + Function used for initializing variables and is automatically invoked when use create an instance of this class
  + class Users {  
     constructor(name="Arun", location="India") {  
     this.name = name;  
     this.location = location;  
     }  
     alertMe() {  
     alert(this.name);  
     }  
     alertMyLoc() {  
     alert(this.location);  
     }  
     }  
     var user1 = new Users('Nanda', 'United States');  
     user1.alertMe();  
     user1.alertMyLoc();  
     var user2 = new Users();  
     user2.alertMe();  
     user2.alertMyLoc();
* **Inheritance**
  + Uses **extends** keyword to inherit from another class.
  + class Users {  
     constructor(name="Arun", location="India") {  
     this.name = name;  
     this.location = location;  
     }  
     alertMe() {  
     alert(this.name);  
     }  
     alertMyLoc() {  
     alert(this.location);  
     }  
     }  
       
     // Students inherits from Users, in other Users in the super class for Students  
     class Students extends Users {  
     rollNo(rollNo=1) {  
     alert(`Roll no of ${this.name} is ${rollNo}`);  
     }  
     }   
       
     var student1 = new Students();  
     student1.alertMe();  
     student1.alertMyLoc();  
     student1.rollNo();
* **super**
  + When used in a constructor, the super keyword appears alone and must be used before the this keyword can be used.
  + This keyword can also be used to call functions on a parent object.
  + class Users {  
     constructor(name="Arun", location="India") {  
     this.name = name;  
     this.location = location;  
     }  
     alertMe() {  
     alert(this.name);  
     }  
     alertMyLoc() {  
     alert(this.location);  
     }  
     }  
       
    //**Super function will make the call the parent constructor and sets the name and location values**  
     class Students extends Users {  
     constructor(address, name, location) {  
      
    **// if called without parameters it will get the values from the parent function**  
     super(name, location);  
      
     this.address = address;  
     }  
     rollNo(rollNo=1) {  
     alert(`Roll no of ${this.name} is ${rollNo}`);  
     }  
     sayAddr() {  
     alert(`My name is ${this.name}, and my location is ${this.location}, and my address is ${this.address}`);  
     }  
     }   
       
     var student1 = new Students('Piscataway, New Jersey, 08854', 'Nanda', 'Canada');  
     student1.alertMe();  
     student1.alertMyLoc();  
     student1.rollNo();  
     student1.sayAddr();
* **Overriding**
  + Using super keyword we can call a function in parent class.
  + class Users {  
     constructor() {}  
     alertMe() {  
     return 'Arun';  
     }  
     }  
       
     class Students extends Users {  
     constructor() { super(); }  
     alertMe() {  
     **//calling alertMe() from parent class using super**  
     return super.alertMe()+', United States';  
     }  
     }  
     var student1 = new Students();  
     alert(student1.alertMe());
* **Getters and setters**
  + As the name suggests used to get and set properties
  + class user {  
     constructor(firstname, lastname) {  
     this.\_firstname = firstname;  
     this.\_lastname = lastname;  
     }  
       
     get firstname() {  
     return this.\_firstname;  
     }  
       
     get lastname() {  
     return this.\_lastname;  
     }  
       
     set name(value) {  
     value = value.split(' ');  
     this.\_firstname = value[0];  
     this.\_lastname = value[1];  
     }  
       
    }  
      
    var user1 = new user('Arungopan', 'Gopakumar');  
      
    document.write(user1.firstname+', '+user1.lastname+'<br />');   
      
    user1.name = 'Nandakumar purohit';  
      
    document.write(user1.firstname+', '+user1.lastname);

**Functional Programming**

* **Arrow function**
  + // single line function to 2 parameters  
    let add = (x,y) => x + y;  
    alert(add(2,3));
  + // function with 1 parameter  
    let square = x => x\*x;  
    alert(square(3));
  + // function with no parameter  
    let sayHello = () => "Say Helloo";  
    alert(sayHello());
  + // function with multiple lines of code  
    let multiLineFn = (x,y) => {  
     let temp = x\*y;  
     return temp;  
    }  
    alert(multiLineFn(5,4));
  + **Using inside forEach loop**  
    var numbers = [5,6,7,8,9];  
    var sum = 0;  
    numbers.forEach(function(item) {  
     sum += item;  
    });  
    alert(sum);  
    var sum2 = 0;  
    numbers.forEach(n => sum2+= n);  
    alert(sum2);

**Bult-in Objects**

Covers the changes in the built-in objects and some new objects ES6 has introduced.

* **Numbers**
  + Introduced **parseInt** and **parseFloat** as part of the number object
  + var num = parseInt("3.4"); // old format  
    var num = Number.parseInt("3.4"); // new format  
    alert(num);  
      
    var num2 = Number.parseFloat("4.555"); // new format  
    alert(num2);
  + **isInteger**
    - if(Number.isInteger(1)) { alert(true)} ;
* **Arrays**
  + **find** (returns the first matching element )
  + **findIndex** (return the first matching index)  
      
    var arr = [5,7,8,9,10];  
    var match = arr.find(item=>item>8);  
    alert(match);
  + **entries** (iterates through an array)  
    - var arr = ['arun', 'nanda', 'naveen'];  
      var entries = arr.entries();  
      var firstentry = entries.next().value;  
      var firstIndex = firstentry[0];//index  
      var firstValue = firstentry[1];//value  
      alert(firstIndex+" => "+firstValue);  
         
      var secondentry = entries.next().value;  
      var secondIndex = secondentry[0];//index  
      var secondValue = secondentry[1];//value  
      alert(secondIndex+" => "+secondValue);
  + **keys** (iterates through keys of an array)
    - var arr = ['arun', 'nanda', 'naveen'];  
      var keys = arr.keys();  
      var firstkey = entries.next().value;  
      var secondkey = entries.next().value;
* **Sets**
  + The Set object lets you store unique values of any type
  + var set1 = new Set();  
     var obj = {"name":"Arun", "loc":"United States"};  
     var arr = ["lekshmi", "Appu"];  
     set1.add('Arun');  
     set1.add(obj);  
     set1.add(obj);  
     set1.add(arr);  
     set1.add(arr);  
     console.log(set1);
  + Above example we added array and object twice and only set contains only one of them.
  + You can pass values the moment that you create a new set
    - var set1 = new Set(["Marlabs", {"name":"Arun"}]);
  + **Clear**
    - Clears all the values of a set
    - set1.clear();
  + **Size**
    - Returns size of a set
    - set1.size
  + **Delete**
    - set1.delete(obj);  
      set1.delete(arr);
  + **forEach** to iterate through a set
    - set1.forEach(function(value) {  
       console.log(value);  
      });
  + **Values** : returns values of a set.
    - var set\_value = set1.values();  
      var setValue = set\_value.next();  
      var setValue = set\_value.next();  
      console.log(setValue.value);
* **Maps**
  + Simple key/value map very similar to objects. Any value may be used as either a key or a value
  + Like sets keys and values has to be unique
  + Iterates in the insertion order
  + Create a map using **new Map()**
  + **set / get functions**
    - var map = new Map();  
      var keyObj = {"company":"Marlabs", "location":"United States"};  
        
      map.set("name", "Arun");  
      **// passing object as key in map**  
      map.set(keyObj, "yes");  
         
      var mapValue = map.get(keyObj);  
      console.log(mapValue);
  + **Storing values while you create a map**
    - var map = new Map([["name", "Arun"], ["location", "US"], ["email", "[arun.gopan@marlabs.com](mailto:arun.gopan@marlabs.com)"]]);
  + **Clear :** clears all the values in the map
    - **map.clear();**
  + **Delete :** delete an item from the map
    - map.delete(key)
  + **forEach :** to iterate through a map
    - map.forEach(function(key, value) {  
       console.log("key = "+key+", value = "+value);  
       });
  + **for :** iterates through an map
    - for(var [key, value] of map) {  
       console.log("key = "+key+", value = "+value);  
      }

**Promise**

The Promise object is used for deferred or asynchronous computations. A Promise represents a value which may be available now, or in the future, or never.

var promise = new Promise(function(resolve, reject) {  
 //resolve('Hello, this is the response from resolve function.');  
 reject('Hello, this is the response from reject function.')  
 });  
 promise.then(function(data) {  
 console.log(data);  
 },  
 function(data) {  
 console.log(data);  
 });

In the above example both **resolve and reject** are methods which indicates whether the promise is resolved or rejected.

**then(),** has two function, first function for the handling resolved promise and second for rejected promise.

**Passing promise object to resolve function :**

var promise1 = new Promise(function(resolve, reject) {  
 resolve(3);  
 });  
   
 var promise2 = new Promise(function(resolve, reject) {  
 resolve(promise1);  
 });  
   
 promise2.then(function(data) {  
 console.log('Success data = '+data);  
 },  
 function(data) {  
 console.log('Error data = '+data);  
 });

**Chaining promise**

function getOrder(orderId) {  
 var orderPromise = new Promise(function(resolve, reject) {  
 resolve({"userId":27});  
 });  
 return orderPromise;  
 }  
 function getUser(userId) {  
 var userPromise = new Promise(function(resolve, reject) {  
 resolve({"companyId":18});  
 });  
 return userPromise;  
 }  
 function getCompany(companyId) {  
 var companyPromise = new Promise(function(resolve, reject) {  
 resolve({"name":"Marlabs"});  
 });  
 return companyPromise;  
 }  
  
 getOrder(3).then(function(order) {  
 return getUser(order.userId)  
 }).then(function(user) {  
 return getCompany(user.companyId);  
 }).then(function(company) {  
 console.log(company.name);  
 });

**Promise.all()**

Returns a promise that resolves when all of the promises in the iterable argument have resolved, or rejects with the reason of the first passed promise that rejects

**Promise.race()**

Returns a promise that resolves or rejects as soon as one of the promises in the iterable resolves or rejects, with the value or reason from that promise.

var promise\_1 = new Promise(function(resolve, reject) {  
 resolve("1");  
 });  
 var promise\_2 = new Promise(function(resolve, reject) {  
 resolve("2");  
 });  
 var promise\_3 = new Promise(function(resolve, reject) {  
 resolve("3");  
 });  
   
 Promise.all([promise\_1, promise\_2, promise\_3]).then(function(data) {  
 console.log('Promise all success = '+data);  
 },  
 function(data) {  
 console.log('Promise all error = '+data);  
 });

**Modules**

Javascript by default does not have modules, but with the use of libraries like CommonJS and AMD (Asynchronous Module Definition), you can create modules in javascript.

ECMA script brings officially brings modules in javascript.

ES6 uses **export keyword** to export a module and **import keyword** to import the same.

export default class Users {  
 constructor(name) {  
 this.name = name;  
 }  
 formatName() {  
 return 'Username id '+this.name;  
 }  
};  
  
export function sayHello() {  
 alert('Hellooo');  
};  
  
export var userObj = {  
 "name":"Arun",  
 "loc":"India",  
 "email":"arun.gopan@marlabs.com"  
};  
  
export var user\_arr = ["Arun", "Nanda", "Naveen"];

//export {user\_arr, userObj, sayHello, Users};

**Note:** Either you can specify export with all the statements you want to export or in one single line

import {Users, user\_arr, userObj, sayHello} from './users';  
   
var user1 = new Users('Arungopan');  
console.log(user1.formatName());  
console.log(user\_arr);  
console.log(userObj);  
sayHello();

**Note : You can also give a different name while import / export times.**

**default :** You can always specify a default statement you can to export using the default keyword along the export.

export default var user\_arr = ["Arun", "Nanda", "Naveen"];

import defaultVal from './users';

**Note : import everything in the module using the \* symbol**

import \* as moduleConent from 'users';

var user1 = new moduleConent.Users('Arungopan');  
console.log(user1.formatName());  
console.log(moduleConent.user\_arr);  
console.log(moduleConent.userObj);  
moduleConent.sayHello();