ECMAScript Introduction

- JavaScript is one of the most popular programming languages in the world, created 25 years ago, it's gone a very long way since its humble beginnings.
- ECMA (also called ES) is the specification / standard and JavaScript, is a programming language that conforms to the ECMAScript specification.
- JavaScript is an implementation of the ECMAScript standard.
- In simple words, ECMAScript is the standard upon which JavaScript is based, and it's often abbreviated to ES.
- ECMAScript is different from the other JavaScript because it is compiled with Transpilers (Source to Source Compiler).
- ECMAScript is commonly used for client-side scripting on the World Wide Web, and it is increasingly being used for writing server based applications and services using Node.js.
- ES6 is a popular update to JavaScript that includes dozens of new features.
- Basically, it is a superset of ES5.
- ES.Next is a name that always indicates the next version of JavaScript.
- ECMAScript have file extensions ".js" & ".es6".

History Behind ECMAScript

Story 1

- In the year 1996, a standards organization called ECMA (European Computer Manufacturers Association) International carved out standard specification called ECMAScript (ES) which all browser vendors could implement.
- JavaScript is the most well-known implementations of ES, while ActionScript (from Macromedia/Adobe Systems) and JScript (from Microsoft) are other implementations of ES.
- ECMA International is a Swiss standards association who is in charge of defining international standards.
- When JavaScript was created, it was presented by Netscape and Sun Microsystems to ECMA and they gave it the name ECMA-262 alias ECMAScript.

Story 2

- JavaScript was originally named JavaScript in hopes of capitalizing on the success of Java.
- Netscape then submitted JavaScript to ECMA International for Standardization. (ECMA is an organization that standardizes information)
- This results in a new language standard, known as ECMAScript.
- ECMAScript is a standard. While JavaScript is the most popular implementation of that standard.

TC39(Technical Committee 39) Introduction

- TC39 is the committee that evolves JavaScript.
- The members of TC39 are companies involved in JavaScript and browser vendors, including Mozilla, Google, Facebook, Apple, Microsoft, Intel, PayPal, Salesforce and others.
- TC39 is the group of people (Brendan Eich and others) who develop the ECMA-262 standard.

Languages Implemented JavaScript

Beside JavaScript, other languages which implemented ECMAScript, including: -

a) ActionScript

ActionScript (the Flash scripting language), which is losing popularity since Flash will be officially discontinued in 2020.

b) JScript

JScript (the Microsoft scripting language), since at the time JavaScript was supported only by Netscape and the browser wars were at their peak, Microsoft had to build its own version for Internet Explorer.

note :- Besides ActionScript & JScript, JavaScript is the most popular and widely used implementation of ES.

Transpilers

- Transpilers, or source-to-source compilers, are tools that read source code written in one programming language, and produce the equivalent code in another language.
- A transpiler is a compiler like program, which convert ES6 JavaScript code to ES5 JavaScript code so that it can run in browsers.
- Transpilation is the process of converting the code from one language into its equivalent language.
- Babel transpiler can be used for converting ES6 JS code to ES5 JS code.
- Transpiler are also called transcompiler.
- A transpiler may translate a code from Pascal to C or C++ to C.

Babel JS

- BabelJS is a JavaScript transpiler which transpiles new features into old standard.
- BabelJS comes with a wide range of features in the form of plugins, presets, polyfills, etc.
- With Babel, developers can write their code using the new features in JavaScript. The users can get the codes transpiled using Babel; the codes can later be used in any browsers without any issues.

BabelJS manages the following two parts :-

a) Transpiling
b) Polyfilling

a) Babel - Transpiler

- Babel-transpiler converts the syntax of modern JavaScript into a form, which can be easily understood by older browsers.
- For example, arrow function, const, let classes will be converted to function, var, etc and the arrow function is converted to a normal function keeping the functionality same in both the cases.

b) Babel - Polyfill

- There are new features added in JavaScript like promises, maps and includes.
- These features when used and transpiled using babel will not get converted so, here we need a Babel-polyfill along with transpiling to make it work on older browsers.

Below there are some features that can be polyfilled :-

a) Promises
b) Map
c) Set
d) Weakmap
e) Weakset
f) include
g) Array.from, Array.of, Array.find, Array.buffer, Array.findIndex
h) Object.assign,Object.entries,Object.values

Transpilers Vs Compilers

a) Transpiler

- Transpiling is a specific term for taking source code written in one language and transforming into another language that has a similar level of abstraction.
- Transpiler, also known as a source-to-source compiler or a transcompiler is a specific type of compiler where input and output languages are at a similar level of abstraction.
- Transpilers are a subset of compilers which take in a source code file and convert it to another source code file in some other language or a different version of the same language.

For example,

- Emscripten: Transpiles C/C++ to JavaScript.
- Babel: Transpiles ES6+ code to ES5.

b) Compiler

- Compiling is the general term for taking source code written in one language and transforming into another.
- A compiler is a program that converts code from high-level programming language (e.g. C, C++, Java) to a low-level language (e.g. assembly language, byte-code, machine language).
- Compiler takes source code written in one language and produce a (or many) output file in some other language.

For example,

"gcc(Garbadge Collector Compiler)" which takes in C code as input and produces a binary executable (machine code) as output.

Advantages of Transpilers

- Transpilers enable developers to use their favorite language and then convert it to the target language which is used in the application.
- Transpiler can convert the additional features into code that is compatible with base language.
- Transpiler is convert legacy code into a newer version of the code.

Working with Transpilers

Method 1 - Install VSCode extension "Babel JavaScript".

Method 2 - Use Babel link from babel website

Babel Website Link -- https://babeljs.io/

Blocked Scope

- Until ES2015, "var" was the only construct available for defining variables.
- ES6 introduced the keywords 'let' and 'const' that enable us to declare variables much easier.
- Variables declared with 'let' and 'const' keywords are block-scoped, which means they are only accessible within the block where they were declared.
- In JavaScript, blocks are denoted by curly braces {} , for example the if else, for, do while, while, try catch and so on.

'let' keyword

- In ES5, when we declare a variable using the 'var' keyword, the scope of the variable is global if we declare it outside of a function or local in case you declare it inside a function.
- ES6 provides a new way of declaring a variable by using the 'let' keyword. The 'let' keyword is similar to the 'var' keyword, except that the variables it declares are block-scoped.
- Use var for top-level variables that are shared across many scopes. Use let for smaller scopes.

Example

```
let x = 10;
if (x == 10) {
    let x = 20;
    console.log(x); // 20:  reference x inside the block
}
console.log(x); // 10: reference at the beginning of the script
```

Example

```
if (true) {
 let a = 40;
 console.log(a); //40
}
console.log(a); // undeclared 'a' is not defined
```

Example -- Assignment with 'let'

```
var bac = "Hello";
let bac = "Hello World";
// bac is already defined, hence error
```

Example -- Variable Hoisting

```
console.log(a) // undefined
console.log(b) // ReferenceError
var a;
let b;
```

Example

```
let a = 50;
let b = 100;
if (true) {
 let a = 60;
 var c = 10;
 console.log(a/c); // 6
 console.log(b/c); // 10
}
console.log(c); // 10
console.log(a); // 50
```

Example

```
let x = "global(let)"
var y = "global(var)"

if (true) {
    var y = "blocked-scope(var)";
    let x = "blocked-scope(let)";

    console.log(y); //blocked-scope(var)
    console.log(x); //blocked-scope(let)
}

console.log(x); //global(let)
console.log(y); //blocked-scope(var)
```

Example -- Creating Global Properties

The global 'var' variables are added to the global object as properties.

Example

```
var counter = 0;
console.log(window.counter); //  0
```

Example

However, the let variables are not added to the global object

```
let counter = 0;
console.log(window.counter); // undefined
```

Redeclaration

Example -- With 'var'

```
var counter = 10;
var counter;
console.log(counter); // 10
```

Example -- With 'let'

```
let counter = 10;
let counter; // error
```

Example -- Reassigning

```
let counter = 10;
counter = 20;
console.log(counter); // 20
```

Looping in 'var' vs 'let'

Example

```
for (let i = 0; i < 5; i++) {
   console.log(i)
}
console.log(i)
```

Example -- Problem

- The intention of the code is to output numbers from 0 to 4 in the console every second. However, it outputs the number 5 five times.
- The reason is that after five iterations, the value of the i variable is 5. And the five instances of the callback function passed to the setTimeOut() function refers to the same variable i with the final value 5.

```
for (var i = 0; i < 5; i++) {
   setTimeout(function () {
      console.log(i); // 5, 5 , 5 , 5, 5
   }, 1000);
}
```

Example -- Solved

In ES5, you fix this issue by creating another scope so that each instance of the callback function refers to a new variable. And to create a new scope, you need to create a function.

```
for (var i = 0; i < 5; i++) {
   (function (j) {
      setTimeout(function () {
         console.log(j);
      }, 1000);
   })(i);
}
```

Example -- Solved

In ES6, the let keyword declares a new variable in each loop iteration, therefore, you just need to replace the var keyword by the let keyword to fix the issue.

```
for (let i = 0; i < 5; i++) {
    setTimeout(function () {
        console.log(i);
    }, 1000);
}
```

'const' keyword

- ES6 provides a new way of declaring a constant by using the 'const' keyword.
- The 'const' keyword works like the 'let' keyword. But the 'const' keyword creates block-scoped variables whose values can’t be reassigned.
- The variables declared by the 'let' keyword are mutual. However, variables created by the const keyword are immutable.

Example

```
const AGE = 25;
console.log(AGE) // 25
```

Example

```
const x = 10;
if (x == 10) {
   const x = 20;
   console.log(x); // 20:  reference x inside the lbock
}
console.log(x); // 10: reference at the begining of the script
```

Example -- Initializing / Redeclaration

```
const AGE; // SyntaxError
const AGE; // SyntaxError
```

Example -- Hoisting

```
console.log(a) // undefined
console.log(b) // ReferenceError
var a;
const b;
```

Example -- Reassigning

```
const AGE = 27;
AGE = 30; // TypeError
```

Example -- Conditionals

```
const a = 10;
if(a > 5)
{
   const a = 100;
   console.log(a); // 100
}
console.log(a); // 10
```

Example -- Looping

```
for (const i = 0; i < 5; i++) {
   console.log(i) //0 , TypeError
}
```