ECMA Script 6

Created by :
Sangeeta Joshi

Agenda

Let

Const

Class

Arrow functions

Modules

Import ,Export

Overview

- ECMASCRIPT 2015

JAVASCRIPT

- **JavaScript** (**JS**) is a lightweight, interpreted or JIT compiled programming language with first-class functions
- Most well-known as the scripting language for Web pages
- Many non-browser environments also use it, such as node.js and Apache CouchDB.
- JS is a prototype-based multi-paradigm, dynamic scripting language, supporting object-oriented, imperative, and declarative (e.g. functional programming) styles.

What falls under the scope of ECMAScript?

- The ***JavaScript programming language*** is standardized by ***ECMA*** (a standards body like W3C) under the name ***ECMAScript***.

- Among other things, ECMAScript defines:

  - Language syntax – parsing rules, keywords, statements, declarations, operators, etc.
  - Types – boolean, number, string, object, etc.
  - Prototypes and inheritance
  - The standard library of built-in objects and functions – JSON, Math, Array methods, Object introspection methods, etc.

What falls under the scope of ECMAScript?

- ES6 is a major upgrade to the language.
- At the same time, while JS code will continue to work, ES6 was designed for maximum compatibility with existing code.
- In fact, many browsers already support various ES6 features, and implementation efforts are ongoing.
- This means most of JS code has *already* been running in browsers that implement some ES6 features!

ES 6

- ES6 is different.
- It's the product of years of harmonious work.
- It's a treasure trove of new language and library features, the most substantial upgrade for JS *ever*.
- The new features range from arrow functions and simple string interpolation, to new concepts like proxies and generators.
- ***ES6 will change the way we write JS code***.

Let is the new var

- ***let variables are block-scoped***: The scope of a variable declared with let is just the enclosing block, not the whole enclosing function.
- ***There's still hoisting with let***, but it's not as indiscriminate.
- ***Global let variables are not properties on the global object***: That is, we won't access them by writing window.variableName. Instead, they live in the scope of an invisible block that notionally encloses all JS code that runs in a web page.
- ***Loops of the form for (let x...) create a fresh binding for x in each iteration***.
- ***It's an error to try to use a let variable before its declaration is reached***. The variable is uninitialized until control flow reaches the line of code where it's declared.
- ***Re-declaring a variable with let is a Syntax Error***

Const

- ES6 also introduces a third keyword that can be used alongside let: ***const***.

- Variables declared with const are just like let except that we can't assign to them, except at the point where they're declared. It's a Syntax Error.

```
const MAX_CAT_SIZE_KG = 3000; //
MAX_CAT_SIZE_KG = 5000; // Syntax Error
MAX_CAT_SIZE_KG++; // nice try, but still a Syntax Error
```

- We can't declare a const without giving it a value.

```
const theFairest;  // Syntax Error
```

Classes

- Classes are "special functions“
- function declarations are hoisted and class declarations are not. We first need to declare class and then access it
- Class declaration:

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
}
```

Classes

Constructors:

- The constructor method is a special method for creating and initializing an object created with a class.

- There can only be one special method with the name "constructor" in a class.

- A Syntax Error will be thrown if the class contains more than one occurrence of a constructor method.

Classes

Static Methods:

- The static keyword defines a static method for a class.
- Static methods are called without instantiating their class and cannot be called through a class instance.
- Static methods are often used to create utility functions for an application.

Modules

- An ES6 module is a file containing JS code
- There's no special module keyword;
- A module mostly reads just like a script.
- ES6 modules are automatically strict-mode code.
- We can use import and export in modules.

Modules

- Everything declared inside a module is local to the module, by default.
- If we want something declared in a module to be public, so that other modules can use it, we must export that feature.
- We can export any top-level function, class, var, let, or const.

Modules

```
// module "my-module.js"
function cube(x) {
  return x * x * x;
}
const foo = Math.PI + Math.SQRT2;
export { cube, foo};
// in another file script.js import it as :

import { cube, foo, graph } from 'my-module';
```

USING THE DEFAULT EXPORT/IMPORT

- To export a single value or to have a fallback value for our module, we could use a default export:

```
export default function cube(x) {
    return x * x * x;
}
```

- Then, in another script, it will be straightforward to import the default export :

```
import cube from './my-module.js';
console.log(cube(3)); // 27
```

Arrow Functions

- An arrow function expression is a syntactically compact alternative to a regular function expression.
- However without its own bindings to : this, arguments, super
- Arrow function expressions are ill suited as methods.
- they cannot be used as constructors.

SIMPLE SYNTAX

```
(param1, param2, …, paramN) => { statements }
(param1, param2, …, paramN) => expression
// equivalent to: => { return expression; }

// Parentheses are optional when there's only one parameter
name:
(singleParam) => { statements }
singleParam => { statements }

// The parameter list for a function with no parameters should
be written with a pair of parentheses.
() => { statements }
```

ADVANCED SYNTAX

```
// Parenthesize the body of function to return an object
literal expression:
params => ({foo: bar})

// Rest parameters and default parameters are supported
(param1, param2, ...rest) => { statements }
(param1 = defaultValue1, param2, …, paramN =
defaultValueN) => {
statements }

// Destructuring within the parameter list is also supported
var f = ([a, b] = [1, 2], {x: c} = {x: a + b}) => a + b + c;
f(); // 6
```

Features

1.ES6 arrow functions
2.Composing Functions
3.Currying functions