//slide1

Hello everyone, my name is Yan, and today I would like to talk about such an important topic for newbies in programming as ECMAScript standard.

//slide2

ECMAScript 2015, also known as ES6, is a fundamental version of the ECMAScript standard.

Since this long time passed between ES5.1 and ES6, the release is full of important new features and major changes in suggested best practices in developing JavaScript programs. To understand how fundamental ES2015 is, just keep in mind that with this version, the specification document went from 250 pages to ~600.

//slide3

The most important changes in ES2015 include:

1. let, const and block scoping

2. Arrow Functions

3. Default Function Parameters

4. Spread / Rest Operator

5. Object Literal Extensions

6. Octal and Binary Literals

7. Array and Object Destructuring

8. super in Objects

9. Template Literal and Delimiters

10. for...of vs for...in

11. Map and WeakMap

12. Set and WeakSet

13. Classes in ES6

14. Symbol

15. Iterators

16. Generators

17. Promises

Let's take a quick look at each of them.

//slide4/1

**1. let, const and block scoping**

let allows you to create declarations which are bound to any block, called block scoping.

//slide4/2

Another form of block-scoped declaration is the const, which creates constants. In ES6, a const represents a constant reference to a value. In other words, Object's and Array's contents may change, only the re-assignment of the variable is prevented

A few things to keep in mind:

1. Hoisting of let and const vary from the traditional hoisting of variables and functions. Both let and const are hoisted, but cannot be accessed before their declaration

2.let and const are scoped to the nearest enclosing block.

3. When using const with fixed strings or values, CAPITAL\_CASING might be appropriate

4.const has to be defined with its declaration

5. Always use const over let, unless you plan on re-assigning the variable.

//slide5/1

**2. Arrow Functions**

Arrow functions are a short-hand notation for writing functions in ES6. The arrow function definition consists of a parameter list ( ... ), followed by the => marker and a function body

**Behold! There is more...**

//slide5/2

Arrow functions don't just make the code shorter. They are closely related to this binding behavior.

Each function in JavaScript defines its own this context but arrow functions capture the this value of the nearest enclosing context

//slide6

**3. Default Function Parameters**

ES6 allows you to set default parameters in function definitions. Here is a simple illustration.

//slide7

**4. Spread / Rest Operator**

operator is referred to as spread or rest operator, depending on how and where it is used.

When used with any iterable, it acts as to "spread" it into individual elements:

When we gathering all arguments together into an array ,it acts call "rest"

//slide8

**5. Object Literal Extensions**

ES6 allows declaring object literals by providing shorthand syntax for initializing properties from variables and defining function methods. It also enables the ability to have computed property keys in an object literal definition.

//slide9

**6. Octal and Binary Literals**

ES6 has new support for octal and binary literals.

//slide10

**7. Array and Object Destructuring**

Destructuring helps in avoiding the need for temp variables when dealing with object and arrays.

//slide11

**8. super in Objects**

ES6 allows to use super method in (classless) objects with prototypes.

//slide12

**9. Template Literal and Delimiters**

ES6 introduces an easier way to add interpolations which are evaluated automatically.

• `${ ... }` is used for rendering the variables.

• `` Backtick is used as delimiter.

//slide13

**10. for...of vs for...in**

for...of iterates over iterable objects, such as array.

for...in iterates over all enumerable properties of an object.

//slide14/1

**11. Map and WeakMap**

Now, we actually use maps in JavaScript all the time. In fact every object can be considered as a Map.

An object is made of keys (always strings) and values, whereas in Map, any value (both objects and primitive values) may be used as either a key or a value.

//slide14/2

**WeakMap**

A WeakMap is a Map in which the keys are weakly referenced, that doesn’t prevent its keys from being garbage-collected. That means you don't have to worry about memory leaks.

In WeakMap as opposed to Map *every key must be an object*.

A WeakMap only has four methods delete(key), has(key), get(key) and set(key, value).

//slide15/1

**12. Set and WeakSet**

*Set* objects are collections of unique values. Duplicate values are ignored. The values can be primitive types or object references.

//slide15/2

Similar to WeakMap, the WeakSet object lets you store weakly held *objects* in a collection. An object in the WeakSet occurs only once; it is unique in the WeakSet's collection.

//slide16/1

13. Classes in ES6

ES6 introduces new class syntax. One thing to note here is that ES6 class is not a new object-oriented inheritance model. They just serve as a syntactical sugar over JavaScript's existing prototype-based inheritance.

Functions defined using the static keyword implement static/class functions on the class.

//slide16/2

**extends and super in classes**

extends allow child class to inherit from parent class in ES6. It is important to note that the derived constructor must call super().

Also, you can call parent class's method in child class's methods using super.parentMethodName()

//slide17

**14. Symbol**

A Symbol is a unique and immutable data type introduced in ES6. The purpose of a symbol is to generate a unique identifier but you can never get any access to that identifier.

If a symbol is used as a property/key of an object, it’s stored in a special way that the property will not show up in a normal enumeration of the object’s properties.

To retrieve an object’s symbol properties, use Object.getOwnPropertySymbols

//slide18

**15. Iterators**

An iterator accesses the items from a collection one at a time, while keeping track of its current position within that sequence. It provides a next() method which returns the next item in the sequence. This method returns an object with two properties: done and value.

ES6 has Symbol.iterator which specifies the default iterator for an object. Whenever an object needs to be iterated (such as at the beginning of a for..of loop), its *@@iterator* method is called with no arguments, and the returned iterator is used to obtain the values to be iterated.

//slide19

**16. Generators**

Generator functions are a new feature in ES6 that allow a function to generate many values over time by returning an object which can be iterated over to pull values from the function one value at a time.

A generator function returns an **iterable object** when it's called. It is written using the new \* syntax as well as the new yield keyword introduced in ES6.

//slide 20/1

**17. Promises**

ES6 has native support for promises. A *promise* is an object that is waiting for an asynchronous operation to complete, and when that operation completes, the promise is either fulfilled(resolved) or rejected.

//slide 20/2

The standard way to create a Promise is by using the new Promise() constructor which accepts a handler that is given two functions as parameters. The first handler (typically named resolve) is a function to call with the future value when it's ready; and the second handler (typically named reject) is a function to call to reject the Promise if it can't resolve the future value.

Every Promise has a method named then which takes a pair of callbacks. The first callback is called if the promise is resolved, while the second is called if the promise is rejected.

//slide 20/3

Returning a value from then callbacks will pass the value to the next then callback.

When returning a promise, the resolved value of the promise will get passed to the next callback to effectively chain them together. This is a simple technique to avoid "callback hell".

//slide 21

It was a brief description of all the new features that appeared with the release of ES6. Let's learn and use these features correctly. Thank you and good luck to all.