



ES6

Part 1

JS

Agenda



- Before We Start ?
 - Terminology
 - What is ECMAScript ?
 - History
 - Goals for ES6
 - Backward Compatibility
 - How to use ES6 today ?
 - Live Coding Session
- 1) Let Variable and Constants
 - 2) Block Scoping
 - 3) Destructuring
 - 4) Spread Operator
 - 5) Arrow Function
 - 6) Default Arguments in Functions
 - 7) Template String/Literal



Before We Start

a small quiz ...



Terminology



- **ECMAScript** : A language standardized by ECMA International
- **JavaScript** : The commonly used name for implementations of the ECMAScript standard
- **TC 39** : The group of people who developed the ECMA standard
- **ECMAScript 6** : The 6th edition of ECMA



What is ECMAScript ?

Let's go back in time ...



History

- **ECMAScript 1** : 1997
- **ECMAScript 2** : 1998
- **ECMAScript 3** : 1999
- **ECMAScript 4** : Abandoned
- **ECMAScript 5** : 2009
- **ECMAScript 6** : 2015
- **ECMAScript 7** : 2016
- **ECMAScript 8** : 2017
- **ECMAScript 9** : 2018

Goals for ES6



- **Better Support for**
 - Complex Applications
 - Libraries
 - Code Generators

Let Variable & Constants



`let` and `const` allows you to declare variables that are limited in scope to the block, statement, or expression on which it is used. This is unlike the `var` keyword, which defines a variable globally, or locally to an entire function regardless of block scope.



Variables

ECMAScript 5

var

Function-Scoped

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

console.log(i); //Outputs 5

ECMAScript 6

let

Block-Scoped

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

console.log(i); //Outputs Undefined

Let Variable

```
function test() {  
  let foo = 1;  
  
  if (foo === 1) {  
    let foo = 22;  
    console.log(foo);  
  }  
  
  if (foo === 22) {  
    let foo = 33;  
    console.log(foo);  
  }  
  
  console.log(foo);  
}  
  
test();
```

Const : Block-Scope

```
const msg = 'hello world';  
  
var msg = 123;  
  
msg = 123;
```

- Only be defined once (within their scope)
- Cannot change the value once set

The **destructuring assignment** syntax is a JavaScript expression that makes it possible to extract data from arrays or objects using a syntax that mirrors the construction of array and object literals.

Inshort : Ability to extract values from objects or arrays into variables.

Creating Variables from Object Properties

ECMAScript 5

```
var bootcamp = {  
  group: 'ttn',  
  title: 'ECMA6',  
  venue: 'Cool Space'  
};  
  
var group = bootcamp.group;  
var title = bootcamp.title;  
var venue = bootcamp.venue;  
  
console.log(group); // ttn  
console.log(title); // ECMA6  
console.log(venue); // Cool Space
```

ECMAScript 6

```
var bootcamp = {  
  group: 'ttn',  
  title: 'ECMA6',  
  venue: 'Cool Space'  
};  
  
let {group,title,venue} = bootcamp;  
  
console.log(group); // ttn  
console.log(title); // ECMA6  
console.log(venue); // Cool Space
```

Creating Variables from Array Elements

ECMAScript 5

```
let brews = [  
  "Corona",  
  "Foster's",  
  "Budweiser"  
];  
  
let x = brews[0];  
let y = brews[1];  
let z = brews[2];  
  
console.log(x); // Corona  
console.log(y); // Foster's  
console.log(z); // Budweiser
```

ECMAScript 6

```
let brews = [  
  "Corona",  
  "Foster's",  
  "Budweiser"  
];  
  
let [x, y, z] = brews;  
  
console.log(x); // Corona  
console.log(y); // Foster's  
console.log(z); // Budweiser
```

Can also change the name of the local variable

```
var bootcamp = {  
  group: 'ttn',  
  title: 'ECMA6',  
  venue: 'Cool Space'  
};  
  
let {group: _group, title: _title, venue: _venue} =  
bootcamp;  
  
console.log(_group); // ttn  
console.log(_title); // ECMA6  
console.log(_venue); // Cool Space
```

Functions with Multiple Return Values

```
function getDate() {  
  let date = new Date();  
  return [date.getDate(), date.getMonth() + 1,  
    date.getFullYear()];  
}
```

```
let [day, month, year] = getDate();
```

```
console.log(day); // 23  
console.log(month); // 2  
console.log(year); // 2017
```


Swapping Values

ECMAScript 5

```
let a = 1;  
let b = 2;  
let temp;  
  
temp = a;    // a = a+b;  
a = b;       // b = a-b;  
b = temp;    // a = a-b;  
  
console.log(a); // 2  
console.log(b); // 1
```

ECMAScript 6

```
let a = 1;  
let b = 2;  
  
[b, a] = [a, b]  
  
console.log(a); // 2  
console.log(b); // 1
```

Spread Operator



ECMAScript 5

```
function addStrings(a, b) {  
  return a + " " + b;  
}  
  
let vals = ['Go', 'Vegan'];  
  
addStrings(vals[0], vals[1])  
  
// returns "Go Vegan"
```

ECMAScript 6

```
function addStrings(a, b) {  
  return a + " " + b;  
}  
  
let vals = ['Go', 'Vegan'];  
  
addStrings(...vals)  
  
// returns "Go Vegan"
```

Arrow Functions



Fat arrow functions, also known as just arrow functions are a brand new feature in ECMAScript 2015 (formerly ES6). Rumor has it that the `=>` syntax was adopted over `->` due to CoffeeScript influence and the similarity of sharing this context.

Arrow functions serve two main purposes:

- > more concise syntax (a shorthand way of declaring functions)
- > sharing lexical this with the parent scope. (shares the scope with the parent)

Fat Arrow Functions in ECMAScript 6

```
let add = (a, b) => a + b;
```

```
add(1,2); // 3
```

```
let addOne = a => a + 1;
```

```
addOne(1); // 2
```

```
let add = (a, b) => a + b;
```

```
let pi = () => 22/7;
```

```
pi(); // 3.142857142857143
```

```
add(1,2); // 3
```

Functions

ECMAScript 5

```
var addAndLog = function(a, b) {  
  let c = a + b;  
  console.log(a, '+' ,b , '=' ,c);  
  return c;  
};
```

```
addAndLog(1, 2);  
// 1 + 2 = 3 (returns 3)
```

ECMAScript 6

```
let addAndLog = (a, b) => {  
  let c = a + b;  
  console.log(a, '+' ,b , '=' ,c);  
  return c;  
};
```

```
addAndLog(1, 2);  
// 1 + 2 = 3 (returns 3)
```

Fat Arrow Functions Shares the Scope of Parent

ECMAScript 5

```
function Person() {  
  this.age = 23;  
  
  setTimeout(function() {  
    console.log(this.age); // undefined  
  }, 1000);  
}  
  
var p = new Person();
```

```
var that = this;  
  
setTimeout(() => {  
  console.log(that.age); // 23  
, 1000);
```

ECMAScript 6

```
function Person() {  
  this.age = 23;  
  
  setTimeout(() => {  
    console.log(this.age); // 23  
  }, 1000);  
}  
  
var p = new Person();
```

Default Parameters

ECMAScript 5

```
function add(a, b) {  
  b = typeof b !== 'undefined' ? b : 1;  
  return a + b;  
}  
add(1); // returns 2
```

ECMAScript 6

```
function add(a, b = 1) {  
  return a + b;  
}  
add(1); // returns 2
```



Template String Literals

Template strings are string literals allowing embedded expressions. You can use multi-line strings and string interpolation features with them.

- back tick character (`) is used to build multi-line strings
- `${}` syntax to reference variables and expressions

m

String Literals

```
let phrase = `If nothing goes right, go to sleep.`;  
  
console.log(phrase);
```

```
let multiLine = `I can't hear you,  
so I'll just laugh and hope it wasn't a question.`;  
  
console.log(multiLine);
```

```
let back = `Who the hell uses \` in a sentence?`;   
  
console.log(back);
```

String Substitution

```
let name = 'akash';  
  
console.log(`Hello ${name}`); // Hello akash
```

```
var bootcamp = {  
  group: 'ttn',  
  title: 'ECMA6',  
  venue: 'Cool Space'  
};  
  
console.log(`We are ${bootcamp.group},  
learning ${bootcamp.title} at a ${bootcamp.venue}`);  
  
// We are ttn,  
// learning ECMA6 at a Cool Space
```

Expressions

```
console.log(`Value of pi is ${22/7}`);
```

```
// Value of pi is 3.142857142857143
```

```
console.log(`Today is ${new Date()}`);
```

```
// Today is Thursday Feb 23 2017 10:10:10 GMT+0530  
(IST)
```

Backward Compatibility



When can I use what ?

- Compability tables: <https://kangax.github.io/compat-table/es6>
- Follow @esdiscuss
- <http://caniuse.com>

How to use ES6 today ?



Transpilers

- Babel : <http://babeljs.io>
- Traceur : <https://github.com/google/traceur-compiler>
- TypeScript (Superset of JavaScript that aims to align with ECMAScript)
<http://www.typescriptlang.org>

How to use ES6 today ?



Module Bundlers

- Webpack : <http://webpack.github.io>
- Browserify : <http://browserify.org>
- Rollup : <http://rollupjs.org>

Live Coding Session



JS

Thank You

Any queries? Open to Q&A



Now we leave and you all write some next-gen JavaScript!