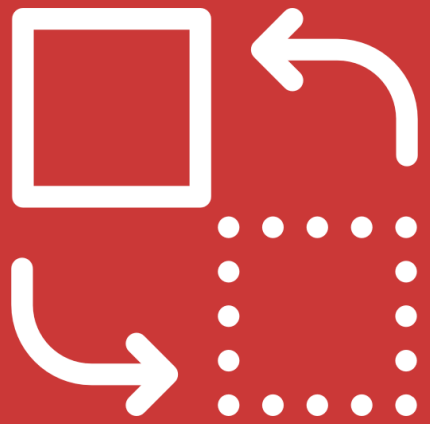
Introduction to

ECMAScript 6

- Rania Hany

Transpilers



A transpiler takes ES6 source code and generates:

- ES5 code
- The source map files

Two main alternatives to transpile ES6 till now

BABEL

tc

What's New?

- Let
- Constants
- Creating Objects
- Destructuring Assignment
- Default parameters and values

- Rest operators
- Classes
- Arrow Functions
- Template Literals
- Modules
- .....

Let …

ES5

Var keyword

```
function getEmployeeName(emp) {
    // var name;
    if (emp.isManager) {
        var name = 'Mr. ' + emp.name;
        return name;
    }
    // name is still accessible here
    return emp.name;
}
```

Remember Hoisting?

ES6

Using let

```
function getEmployeeName(emp) {

    if (emp.isManager) {
        let name = 'Mr. ' + emp.name;
        return name;
    }
    // name is not accessible here
    return emp.name;
}
```

Variables are now restricted to their block.

Constants

ES6 introduces const to declare… constants!
It's only declared at the block level. (Not Hoisted)

```
const minSalary = 1200;
```

- It has to be initialized

```
minSalary = 1200; //Syntax Error
```

- You can't assign another value later.

Constants (cont.)

Constants with objects and arrays

- you can initialize a constant with an object and later modify the object content.

```
const user = {};
user.name = 'hamada'; // works
```

- But you can't assign another object

```
user = {name : 'hamada'}; // Syntax Error
```

Constants (cont.)

Constants with objects and arrays

- Same thing with arrays

```
const users = [];
users.push({name = 'hamada'}); // works
```

```
users = []; // Syntax Error
```

Creating Objects

ES5

```
function createCar() {
  const name = 'BMW';
  const color = 'blue';
  return { name: name, color: color };
}
```

Can this be simplified?

Creating Objects (Cont.)

ES6

```
function createCar() {
  const name = 'BMW';
  const color = 'blue';
  return { name, color };
}
```

Simplified shortcut! when the object property you want to create has the same name as the variable used as the value.

Destructuring Assignment

- Assigning variable from object or an array

ES5

```
var httpOptions = { timeout: 2000, isCache: true };
// later
var httpTimeout = httpOptions.timeout;
var httpCache = httpOptions.isCache;
```

ES6

```
const httpOptions = { timeout: 2000, isCache: true };
// later
const {timeout:httpTimeout, isCache:httpCache}= httpOptions;
```

Destructuring Assignment (Cont.)

What if the variable you want to assign has the same name as the property?

```
const httpOptions = { timeout: 2000, isCache: true };
const { timeout, isCache } = httpOptions;
// you now have a variable named 'timeout'
// and one named 'isCache' with correct values
```

Destructuring Assignment (Cont.)

Same concept with arrays

```
const salaries= [1000, 2000, 3000];
const [low, medium, high] = salaries;
// now we have variable named low' = 1000
// and 'medium' = 2000, 'high' = 3000,
```

```
const salaries
      = [1000, 2000, 3000];
const [low , , high] = salaries;
// 'low' = 1000
// 'high' = 3000
```

```
const salaries
      = [1000, 2000, 3000];
const [low, medium] = salaries;
// 'low' = 1000
// 'medium' = 2000
```

Destructuring Assignment (Cont.)

One interesting use of this can be for multiple return values of functions

```
function randomStudent() {
    const student = { name: 'Hamada' };
    const track = 3;
    // ...
    return { student , track};
}
const { track, student } = randomStudent();
```

Destructuring Assignment (Cont.)

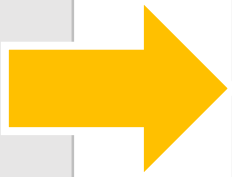
if you don't care about the track, you can write:

```
function randomStudent() {
    const student = { name: 'Hamada' };
    const track = 3;
    // ...
    return { student , track};
}
const { student } = randomStudent();
```

Default Parameters

ES5

```
function getBook(price, pages)
{
    price = price || 10;
    pages = pages || 100;
    // ...
}
```



```
getBook(5, 20);
getBook();
//same as getBook(10, 100);
getBook(15);
//same as getBook(15, 100);
```

Was it obvious that getBook has an optional parameters with default values, without reading its body?

Default Parameters (Cont.)

ES6

```
function getBook(price = 10, pages = 100)
{
    //  ...
}
```

More clear now?

Default Parameters (Cont.)

Default parameter as a function

```
function getBook(price= defaultPrice(), pages= 100)
{
        // ...
}
```

```
function getBook(price = defaultPrice(), pages = price * 10) {
        // ...
}
```

```
function getBook(price = pages/10, pages = 100) {
        // ...
}
```

Rest Operator

Passing extra arguments to a function

ES5

Using arguments

```
function addBooks() {
  for (var i = 0; i <
  arguments.length; i++) {
     myBooks.push(arguments[i]);
  }
}
addBooks('es5', 'es6');
```

ES6

Using …

```
function addBooks(...books) {
    for (let book of books) {
        myBooks.push(book);
    }
}
addBooks('es5', 'es6');
```

Rest Operator (Cont.)

Rest operator can also work when destructuring data:

```
let racers = ["Cat", "Dog", "Hamster"];

const [winner, ...losers] = racers;
// 'winner' will have the first animal,
// and 'losers' will be an array of the
other ones
```

Spread Operator

looks awfully similar to Rest operator! But the spread operator is the opposite

```
const prices = [12, 3, 4];
const price = Math.min(...prices);
```

it takes an array and spreads it in variable arguments.

Arrow Function ⇒

```
var getSpeed = function() {
      return 10;
}
console.log(getSpeed());
```

```
var getSpeed = () => 10;
console.log(getSpeed());
```

```
var getSpeed = function(level) {
      return level + 5;
}
console.log(getSpeed());
```

```
var getSpeed = (level) => level+ 5;
console.log(getSpeed());
```

```
var getSpeed = function(level) {
      console.log(level);
      return level + 5;
}
console.log(getSpeed());
```

```
var getSpeed = (level) => {
      console.log(level);
      return level + 5;
}
console.log(getSpeed());
```

Arrow Function ⇒ (Cont.)

Arrow functions don't have a new this !

```
var maxFinder = {
    max: 0,
    find: function (numbers) {
        // let's iterate
        numbers.forEach(
        function (element) {
            // if the element is greater,
            set it as the max
            if (element > this.max) {
                this.max = element;
            }
        });
    }
};
maxFinder.find([2, 3, 4]);
// log the result
console.log(maxFinder.max);
```

```
const maxFinder = {
    max: 0,
    find: function (numbers) {
        numbers.forEach(element => {
            if (element > this.max) {
                this.max = element;
            }
        });
    }
};
maxFinder.find([2, 3, 4]);
// log the result
console.log(maxFinder.max);
```

Template Literals `

Composing strings has always been painful in JavaScript, as we usually have to use concatenation:

ES5

```
const fullname = 'Miss ' + firstname + ' ' + lastname;
```

ES6

```
const fullname = `Miss ${firstname} ${lastname}`;
```

- Templating system
- Multiline support

Angular Component :

```
const template =
        `
        <div>
        <h1>Hello</h1>
        `
        </div>`;
```

Classes

ES6 introduces classes to JavaScript!

Classes

```
class Car {
   constructor(color) {
      this.color = color;
   }
   toString() {
      return `${this.color} car`;
   }
}
const blueCar = new Car('blue');
console.log(blueCar.toString()); // blue car
```

Constructor

Methods

Creating Objects

Classes (Cont.)

It can also have static attributes and methods:

```
class Car {
   constructor(color) {
      this.color = color;
   }
   static defaultSpeed() {
      return 10;
   }
}
const speed = Car.defaultSpeed();
```

Static Method

“If you have **Classes**
You also have
Inheritance”

Classes (Inheritance)

Simply Using the 'extends' keyword

Base Class

Derived Class

```
class Animal {
   speed() {
      return 10;
   }
}
class Cat extends Animal {
}
const cat = new Cat();
console.log(cat.speed());
```

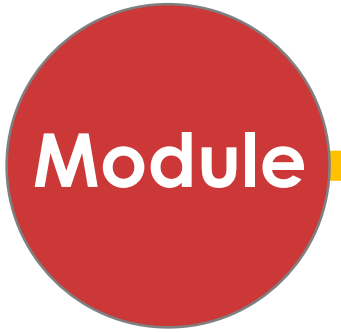
Classes (Inheritance) (Cont.)

```
class Animal {
   speed() {
      return 10;
   }
}
class Cat extends Animal {
   speed() {
      return super.speed() + 10;
   }
}
const cat = new Cat();
console.log(cat.speed());
```

Modules

A standard way to organize functions in namespaces and to dynamically load code in JS

Modules



game_service.js

```
export function loadGame(level, lifes) {
// ...
}
export function newGame() {
// ...
}
```

Here we are importing the two functions, and we have to specify the filename containing these functions:

another file

```
import { loadGame, newGame } from './game_service';
```

Modules

You can import only one method if you need, you can even give it an alias

```
import { loadGame as load } from './game_service';
```

Later, you can use the functions with its new alias

```
load(5,3);
```

Modules

if you need to import all the methods from a module, use a wildcard '*'.

```
import * as gameService from './game_service';
```

In this case, alias is a must

```
gameService.load(race, pony1);
gameService.start();
```

If your module exposes only one function or value or class, you don't have to use named export

```
// book.js
export default class Book {
}
// races_service.js
import Book from './book';
```

Thank You