ECMAScript 2015

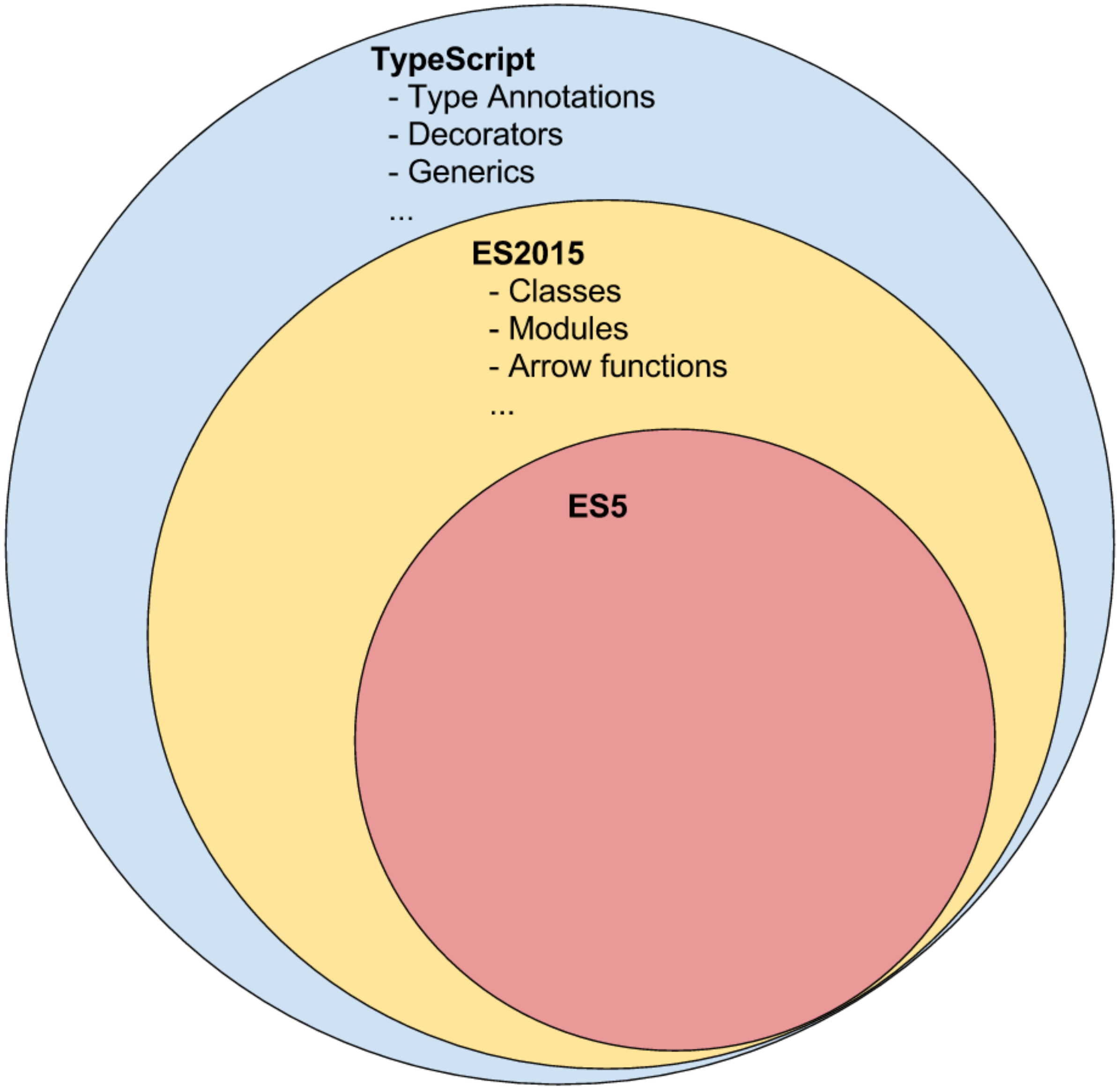
&

TypeScript

ECMAScript 2015 (ES2015)

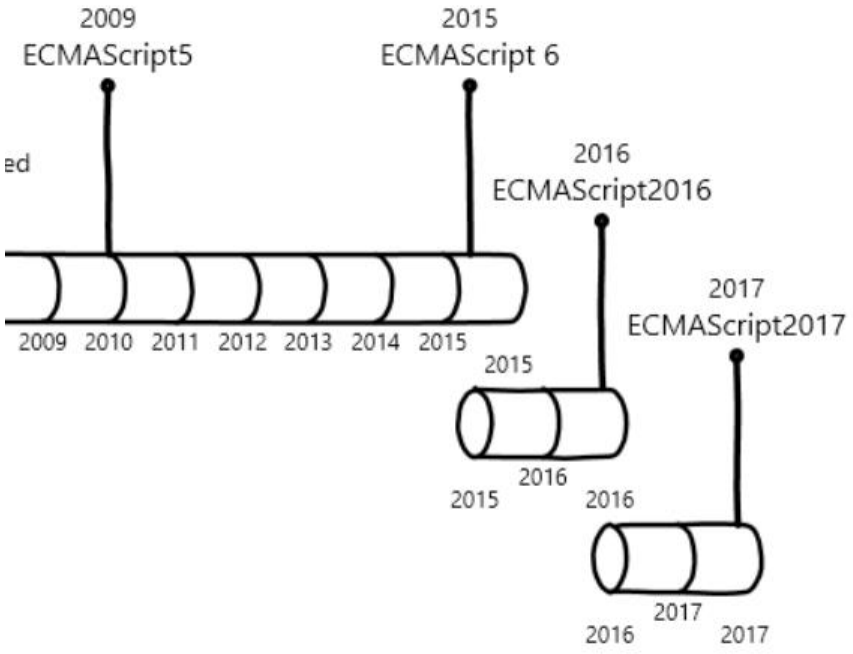
< Back

ES2015 & TypeScript Big Picture



ES2015 Status

- Approved June 17, 2015
- Largest Update in JavaScript's History
  - Future updates will be much smaller and more frequent



ECMAScript 2015 features

- Variables: var, let, const
- OOP: classes, inheritance, super, get/set
- Functions: generators, iterators, arrow functions, for-of
- Data Structures: set/weakset, map/weakmap
- Async operations: built-in promises
- Modules: imports, exports
- Objects: computed properties, shorthand properties, Object.is(), Object.assign(), proxies
- Others: templates, Math and Number extensions

ES2015 Variables

Back

ES2015 Variables

- ES2015 introduces new ways to declare variables:
  - let – creates a scope variable
    - Accessible only in its scope

```
for(let number of [1, 2, 3, 4]){
  console.log(number);
}
//accessing number here throws exception
```

- **const – creates a constant variable**
  - **Its value is read-only and cannot be changed**

```
const MAX_VALUE = 16;
MAX_VALUE = 15; // throws exception
```

For-of loop

- The for-of loop iterates over the values
  - Of an array

```
let sum = 0;
for(let number of [1, 2, 3])
  sum+= number;
```

- **Of An iteratable object**

```
function* generator(maxValue){
  for(let i = 0; i < maxValue; i+=1){
    yield i;
  }
}
let iter = generator(10);
for(let val of iter()){
  console.log(val);
}
```

Templated Strings in ES2015

- ES2015 supports templated strings
  - i.e. strings with placeholders:

```
let people = [new Person('Samir', 'Saghir'), … ];
for(let person of people){
   log(`Fullname: ${person.fname} ${person.lname}`);
}
```

Classes and Inheritance

The way of OOP in ES2015

Back

Classes and Inheritance in ES2015

- ES2015 introduces classes and a way to create classical OOP

```
class Person extends Mammal{
  constructor(fname, lname, age){
    super(age);
    this._fname = fname;
    this._lname = lname;
  }
  get fullname() {
    //getter property of fullname
  }
  set fullname(newfullname) {
    //setter property of fullname
  }
  // more class members…
}
```

Constructor of the class

Getters and setters

Arrow Functions

Also called LAMBDA expressions

Back

Arrow Functions

- Arrow functions easify the creation of functions:

```
numbers.sort(function(a, b){
  return b - a;
});
```

Becomes

```
numbers.sort((a, b) => b - a);
```

```
var fullnames =
   people.filter(function (person) {
     return person.age >= 18;
   }).map(function (person) {
     return person.fullname;
   });
```

Becomes

```
var fullnames2 =
  people.filter(p => p.age >= 18)
    .map(p => p.fullname);
```

Arrow Functions – Example

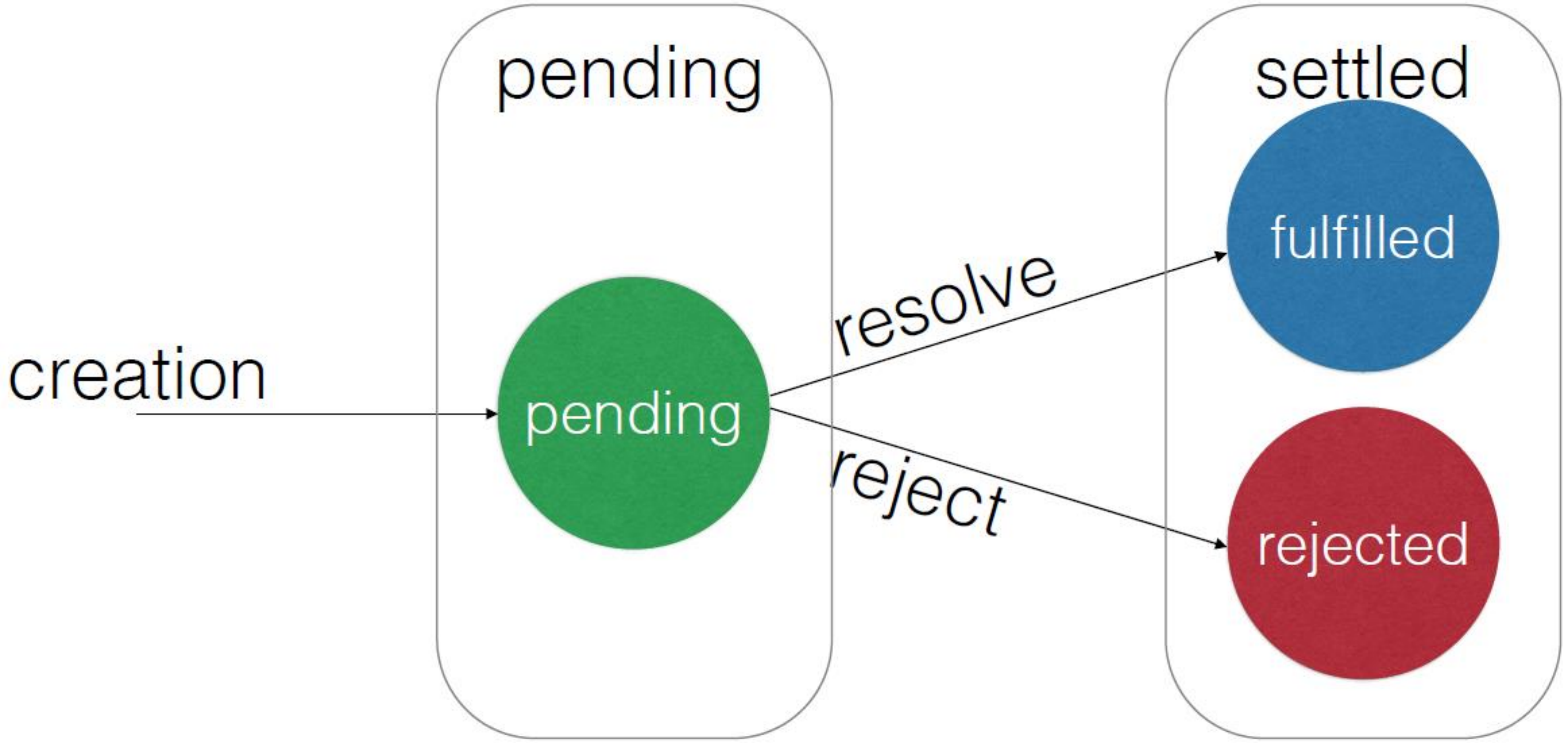
```
let arr = [1, 2, 3];
let sum = arr
   .map(x => x * 2)
   .reduce((sum, x) => sum + x);

log(sum); // ==> 12
```

Promises

- Object that represents a future value
- Has one of three states: pending, fulfilled, or rejected
- Immutable once fulfilled or rejected
- Producer returns a promise which it can later fulfill or reject
- Consumers listen for state changes with .then and .catch methods

State of a Promise



Promises – Example

```
// producer creates a promise, resolves when ready
function timeout(ms) {
  return new Promise(resolve => {
    setTimeout(resolve, ms);
  });
}

log("start");

// consumer gets a promise, is notified when resolved
let p = timeout(1000);
p.then(() => log("end"));
```

Object Literals

Back

Object Literals

- ES2015 adds a new feature (rule) to the way of defining properties:
  - Instead of

```
let name = 'Samir Saghir',
    age = 25;
let person = {
  name: name,
  age: age
};
```

- We can do just:

```
let name = 'Samir Saghir';
age = 25;
let person = {
  name,
  age
};
```

Destructuring Assignments

< Back

Destructuring Assignments

- Destructuring assignments allow to set values to objects in an easier way:
  - Destructuring assignments with arrays:

```
var [a,b] = [1,2]; //a = 1, b = 2
var [x, , y] = [1, 2, 3] // x = 1, y = 3
var [first, second, ...rest] = people;
```

- Swap values: `[x, y] = [y, x]`
- Result of method:

```
function get(){ return [1, 2, 3]; }
var [x, y] = get();
```

Destructuring Assignments

- Destructuring assignments allow to set values to objects in an easier way:
    - Destructuring assignments with objects:

```
var person = {
  name: 'Samir Saghir',
  address: {
    city: 'Doha',
    street: 'University'
  }
};

var {name, address: {city}} = person;
```

Maps and Sets

Back

Maps and Sets

- ES2015 supports maps and sets natively

```
let names = new Set();
names.add('Samir');
names.add('Fatima');
names.add('Mariam');
names.add('Ahmed');
names.add('Samir'); // won't be added
```

Using Iterators – Example

```
let arr = ['a', 'b', 'c'];
for(let item of arr) { log(item) }


let map = new Map();
map.set(1, 'a');
map.set(2, 'b');
for(let pair of map) { log(pair) }
for(let key of map.keys()) { log(key) }
for(let value of map.values()) { log(value) }
```

ES2015 Modules

Back

Modules

- ES2015 brings a module system to the table that enables us to write modular code.
    - Each JS file has its own scope (not the global)
    - Each file decides what to export from its module
- Export the objects you want from a module:

```
// Car.js
export class Car { ... }
export class Convertible extends Car { ... }
```

- Use the module in another file:

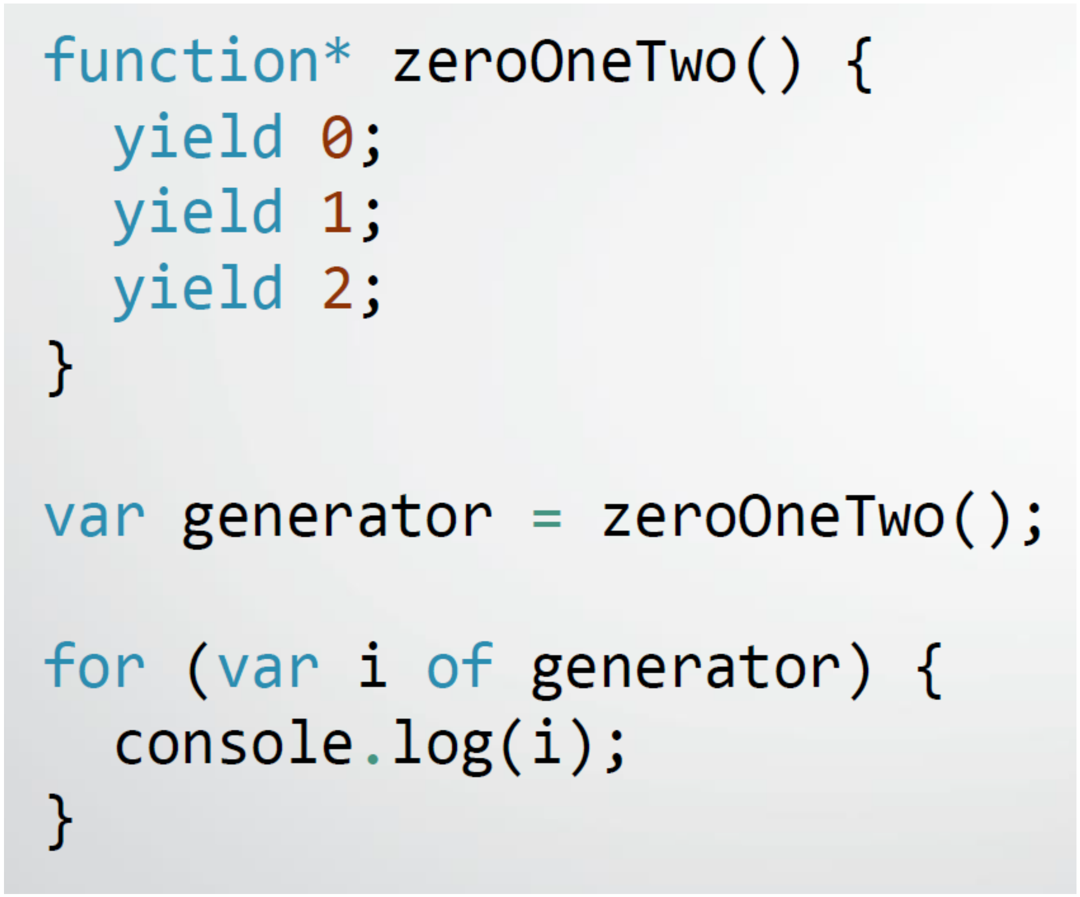
```
// App.js
import {Car, Convertible} from 'Car';
let bmw = new Car();
let cabrio = new Convertible();
```

Generators

< Back

Generators

- Suspending execution until someone calls next()
- **Run..Stop..Run**



```
function* zeroOneTwo() {
  yield 0;
  yield 1;
  yield 2;
}

var generator = zeroOneTwo();

for (var i of generator) {
  console.log(i);
}
```

Resources

- Best ES 2015 eBook

http://exploringjs.com/es6/

- Best ES 2015 Learning Resources

https://github.com/ericdouglas/ES2015-Learning

TypeScript

Back

What is TypeScript?

- Strongly Typed
- Classes
- Interfaces
- Generics
- Modules
- Type Definitions
- Compiles to JavaScript
- EcmaScript 6 & 7 Features

Type Annotations

- Type annotations provide optional static typing. Applied using **: T** syntax

```
var height:number = 6;
var isDone:boolean = true;
var name:string = 'thoughtram';

var list:number[] = [1, 2, 3];
var list:Array<number> = [1, 2, 3];

function add(x: number, y: number): number {
  return x+y;
}
```

Decorators

- A decorator is an **expression** that is evaluated after a class has been defined, that can be used to **annotate or modify** the class in some fashion.

```
import {Component, View} from 'angular2/core';


@Component({
  selector: 'contacts-app'
})
@View({
  template: 'Hello World!'
})
class ContactsApp {

}
```

Resources

- TypeScript Playground

http://www.typescriptlang.org/Playground

- TypeScript Handbook

http://www.typescriptlang.org/Handbook