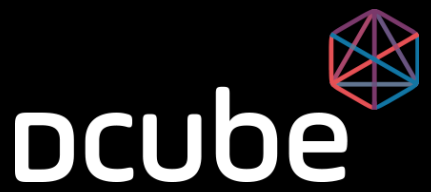
TS best practices

Félix Billon



Tech lead

Co-orga

Member

Microsoft
CERTIFIED
Solutions Developer
Web Applications

@felix_billon

felixbillon

http://shakedatcode.com

Sommaire

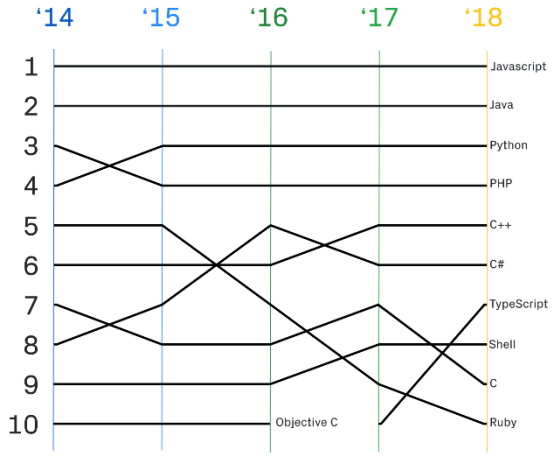
- Introduction
- Architecture
- Transpilation
- Typing
- Migration JS → TS
- Conclusion

Introduction

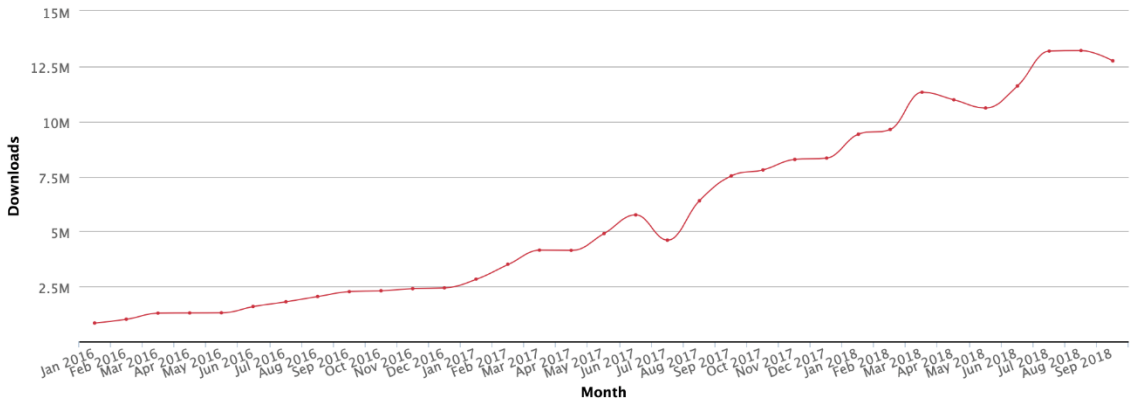
Raise of TypeScript

Popularity

Stackoverflow



Github



npm

TypeScript in open source world

Visual Studio Code

Angular

Ionic / Stencil

RxJS

NativeScript

TensorFlow.js

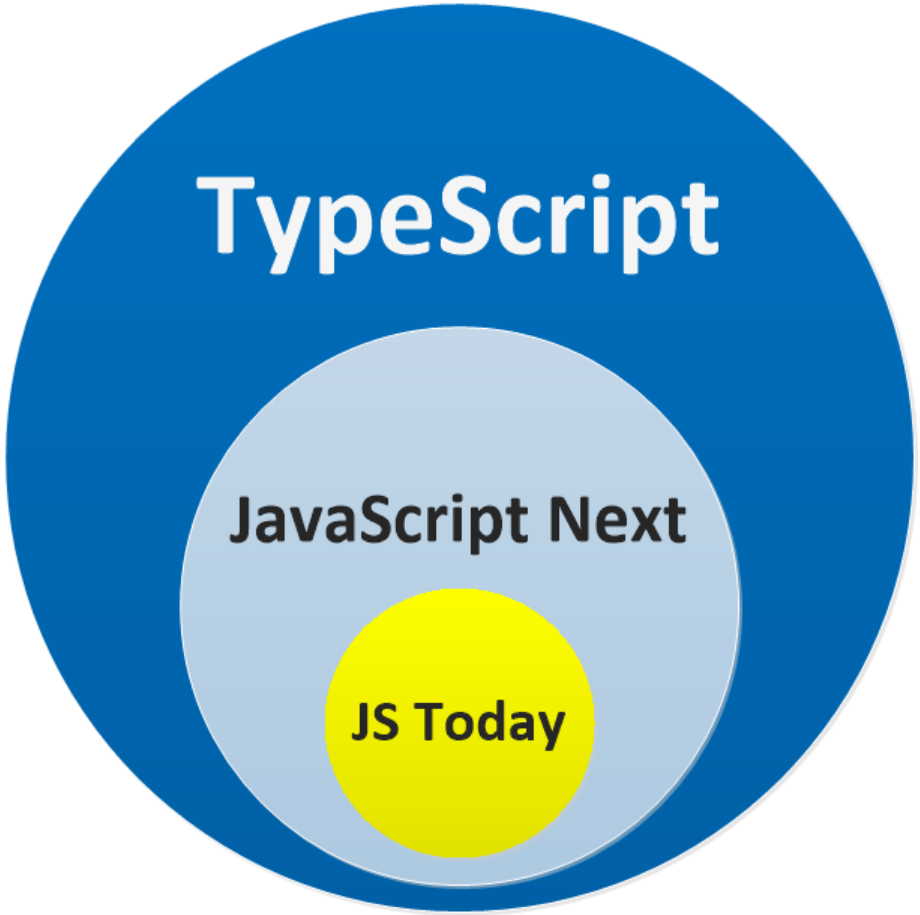
NestJS

Vue.js

In progress

TypeScript ?

- Superset of Javascript.
- Made in Microsoft.
- Open source on GitHub.
- 2014 v1.0 -> today v3.2.
- Main features :
  - Transpilation -> generates Javascript.
  - Typing -> only useful during compilation.



Why this talk ?

- No official style guide !
- Maybe coding guidelines on TypeScript’s Github ?

Coding guidelines

Daniel Rosenwasser edited this page on 11 May · 18 revisions

STOP READING IMMEDIATELY

THIS PAGE PROBABLY DOES NOT PERTAIN TO YOU

These are Coding Guidelines for *Contributors to TypeScript*

This is *NOT* a prescriptive guideline for the TypeScript community

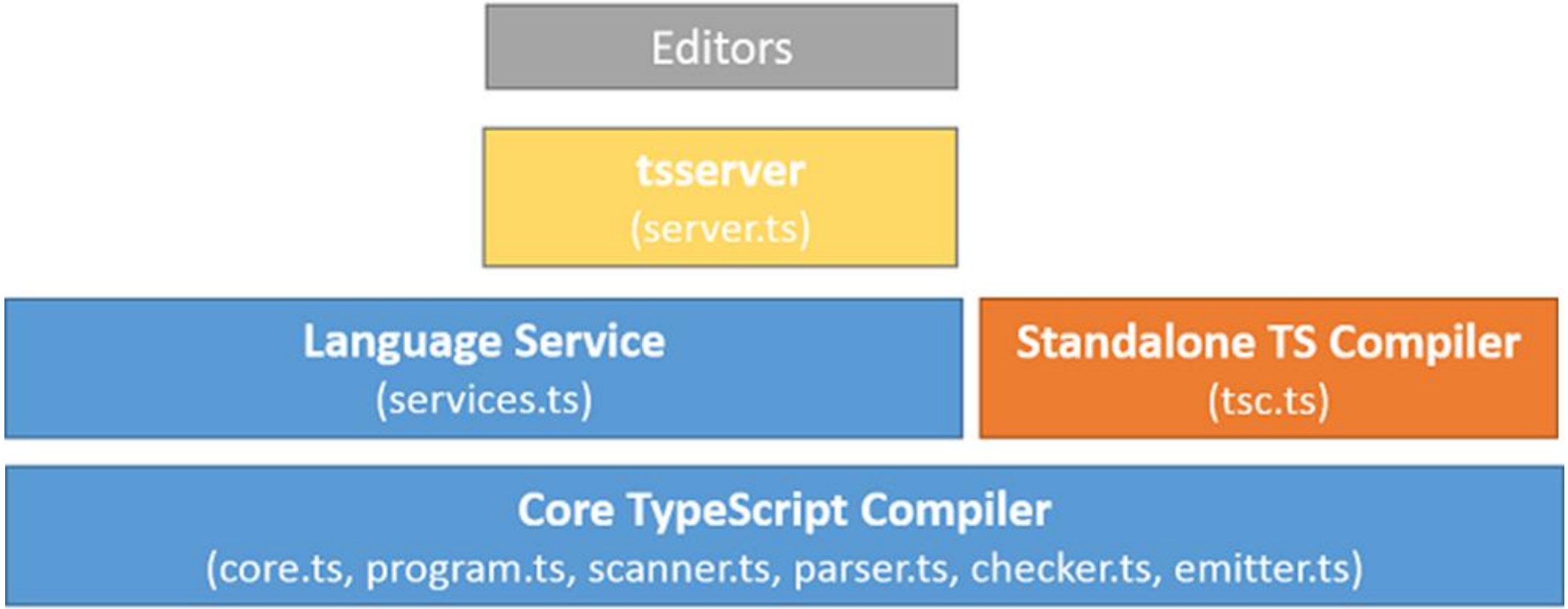
These guidelines are meant for contributors to the TypeScript project's codebase.

Be careful with best practices

“Don't blindly adhere to any old advice”

Architecture

Architecture



CLI tsc

- Files + options → tsc → core compiler → JavaScript files.

- Use options :
  - Command line :

```
tsc **/*.ts --target=es5 --sourcemap=true
```

  - Configuration file aka tsconfig.json :

```
{
  "compilerOptions": {
      "target": "es5",
      "module": "es2015",
      "removeComments": true,
      "sourceMap": true
},
  "include": ["src/**/*"]
}
```

CLI tsc

- Use tsconfig.js in preference to command line.

- Initialize it this way : `tsc --init`

tsconfig.json \.\C:\Projects\talk\tsconfig.json\...

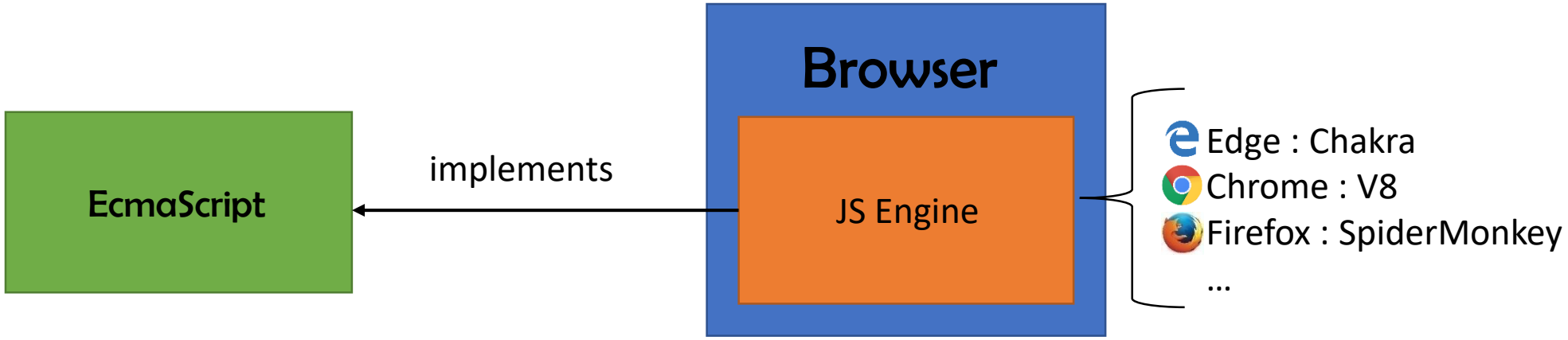
```
{
  "compilerOptions": {
    /* Basic Options */
    "target": "es5",                          /* Specify ECMAScript target version: 'ES3' (default), 'ES5', 'ES2015', '
    "module": "commonjs",                     /* Specify module code generation: 'none', 'commonjs', 'amd', 'system', '
    // "lib": [],                             /* Specify library files to be included in the compilation. */
    // "allowJs": true,                       /* Allow javascript files to be compiled. */
    // "checkJs": true,                       /* Report errors in .js files. */
    // "jsx": "preserve",                     /* Specify JSX code generation: 'preserve', 'react-native', or 'react'. *
    // "declaration": true,                   /* Generates corresponding '.d.ts' file. */
    // "declarationMap": true,                /* Generates a sourcemap for each corresponding '.d.ts' file. */
    // "sourceMap": true,                     /* Generates corresponding '.map' file. */
    // "outFile": "./",                       /* Concatenate and emit output to single file. */
    // "outDir": "./",                        /* Redirect output structure to the directory. */
    // "rootDir": "./",                       /* Specify the root directory of input files. Use to control the output d
    // "composite": true,                     /* Enable project compilation */
    // "removeComments": true,                /* Do not emit comments to output. */
    // "noEmit": true,                        /* Do not emit outputs. */
    // "importHelpers": true,                 /* Import emit helpers from 'tslib'. */
    // "downlevelIteration": true,            /* Provide full support for iterables in 'for-of', spread, and destructur
    // "isolatedModules": true,               /* Transpile each file as a separate module (similar to 'ts.transpileModu

```

Transpilation

ECMAScript

- ECMAScript = standard for scripting languages.

- ECMAScript implementation : javascript, actionscript, …

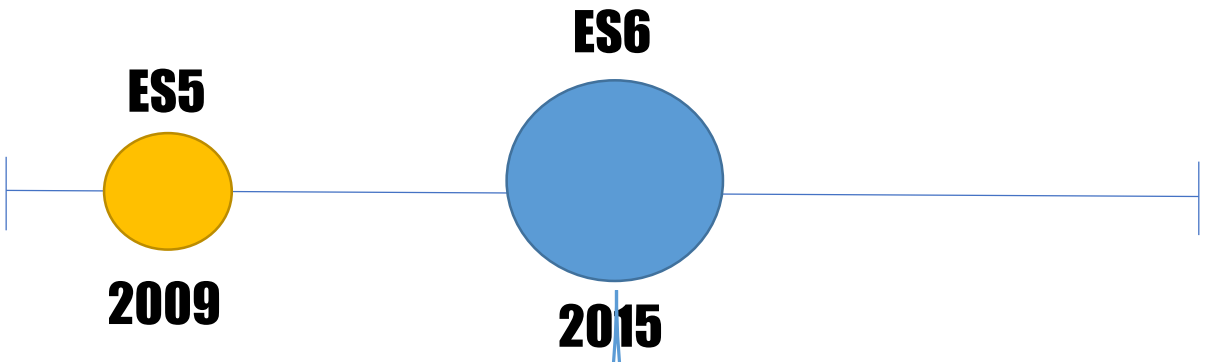


ECMAScript : historical

TC 39

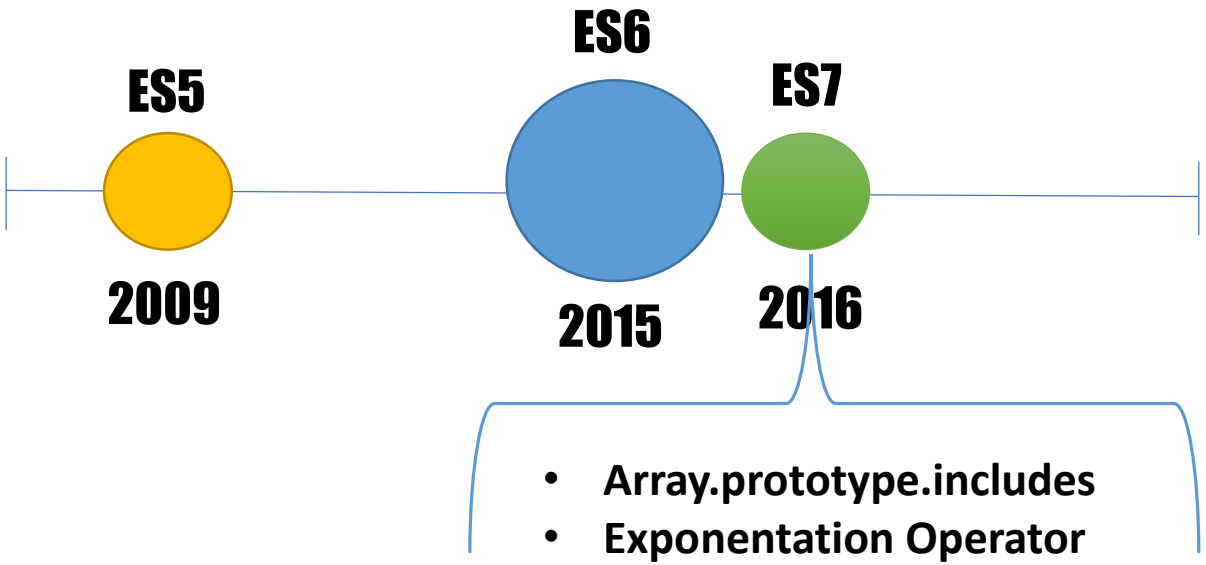
ECMAScript : historical

TC 39

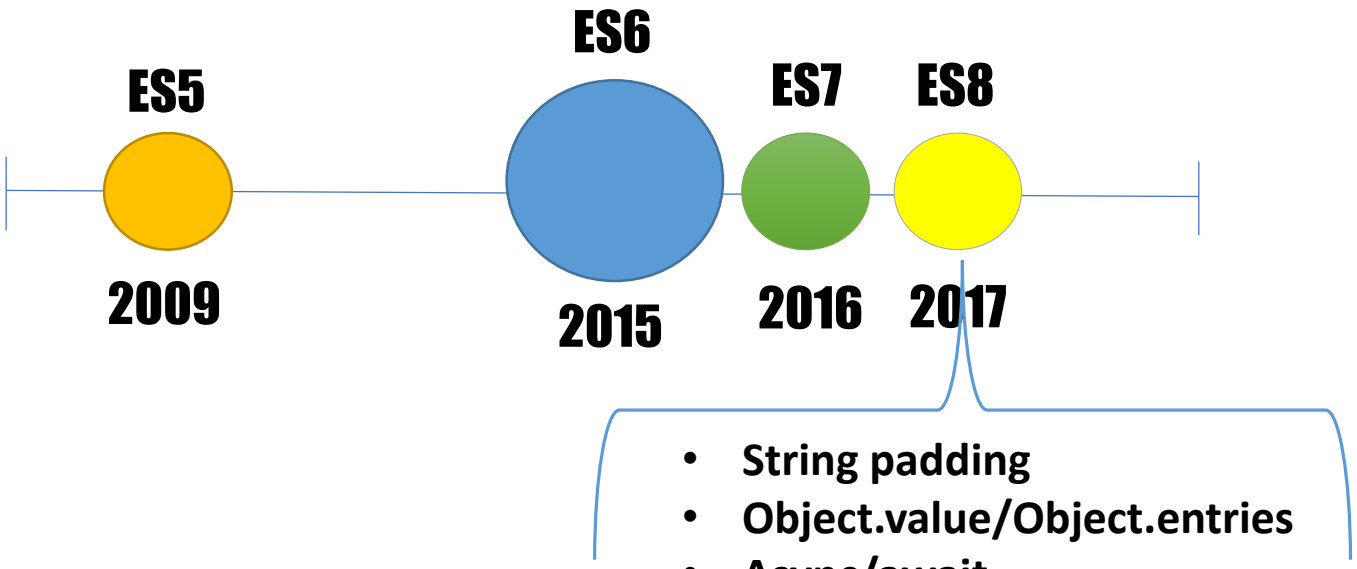


- Classes
- Arrow function
- Promise
- Destructuring
- Constants
- Modules
- Template Literals
- Map/Set
- Iterators
- Generators
- Symbol type
- …

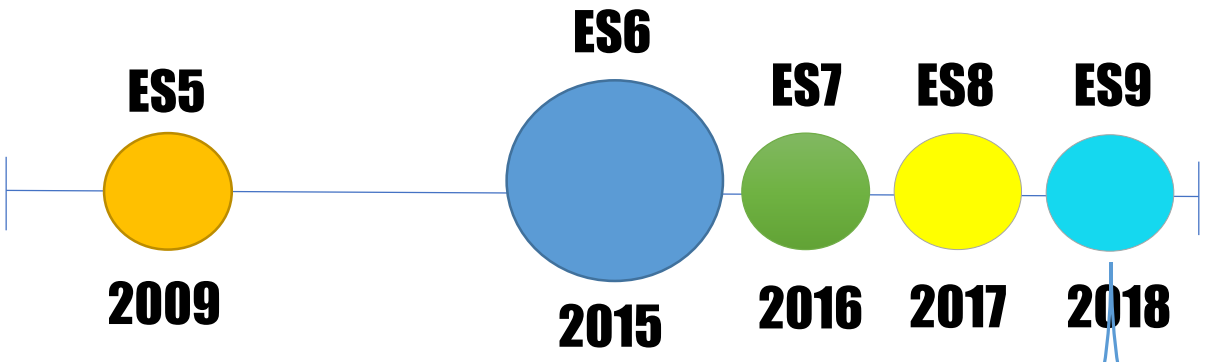
ECMAScript : historical



ECMAScript : historical



ECMAScript : historical



- Asynchronous Iteration
- Rest/Spread properties
- Improve Regex
- Promise.prototype.finally
- Template Literal Revision

Implementation rate d’ES7+

ECMAScript | 5 | 6 | 2016+ | next | intl | non-standard | compatibility table

Flattr | by kangax & webbedspace & zloirock | Fork 687

Sort by Engine types | Show obsolete platforms | Show unstable platforms

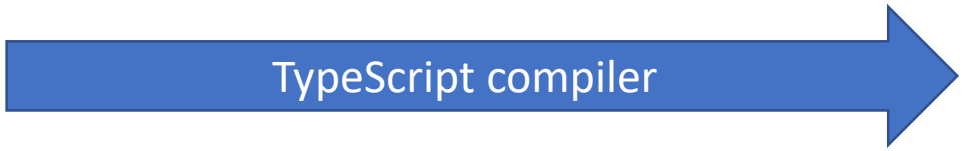
V8 | SpiderMonkey | JavaScriptCore | Chakra | Other

Minor difference (1 point) | Small feature (2 points) | Medium feature (4 points) | Large feature (8 points)

		Compilers/polyfills					Desktop browsers										Servers/runtimes					
	100%	6%	56%	52%	55%	10%	0%	52%	58%	78%	78%	78%	100%	100%	83%	90%	2%	19%	77%	100%	8%	
Feature name	Current browser	Traceur	Babel 6 + core-js	Closure 2018.10	Type-Script + core-js	es7-shim	IE 11	Edge 17	Edge 18	FF 60 ESR	FF 62	FF 63	CH 69, OP 56	CH 70, OP 57	SF 11.1	SF 12	PJS	Node >=6.5 <7[2]	Node >=8.10 <9[2]	Node >=10.13 <11[2]	DUK 2.2	
exponentiation (**) operator	3/3	2/3	3/3	3/3	2/3	0/3	0/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	0/3	0/3	3/3	3/3	2/3	
Array.prototype.includes	3/3	0/3	3/3	2/3	3/3	2/3	0/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	3/3	0/3	3/3	3/3	3/3	0/3	
2016 misc																						
generator functions can't be used with "new"[7]	Yes	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	
generator throw() caught by inner generator[8]	Yes	No	No	Yes	Yes[9]	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	
strict fn w/ non-strict non-simple params is error[10]	Yes	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	
nested rest destructuring, declarations[11]	Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	
nested rest destructuring, parameters[12]	Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	
Proxy, "enumerate" handler removed[13]	Yes	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	
Proxy internal calls, Array.prototype.includes	Yes	No	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	
2017 features																						
Object static methods	4/4	0/4	4/4	3/4	4/4	3/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	0/4	4/4	4/4	0/4	
String.padding	2/2	0/2	2/2	2/2	2/2	2/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	0/2	2/2	2/2	0/2	
trailing commas in function syntax	2/2	0/2	2/2	2/2	2/2	0/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	0/2	0/2	2/2	2/2	0/2	

Transpilation

file.ts



file.js

```ts
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}

let greeter = new Greeter("world");
```

```js
var Greeter = /** @class */ (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
}());
var greeter = new Greeter("world");
```

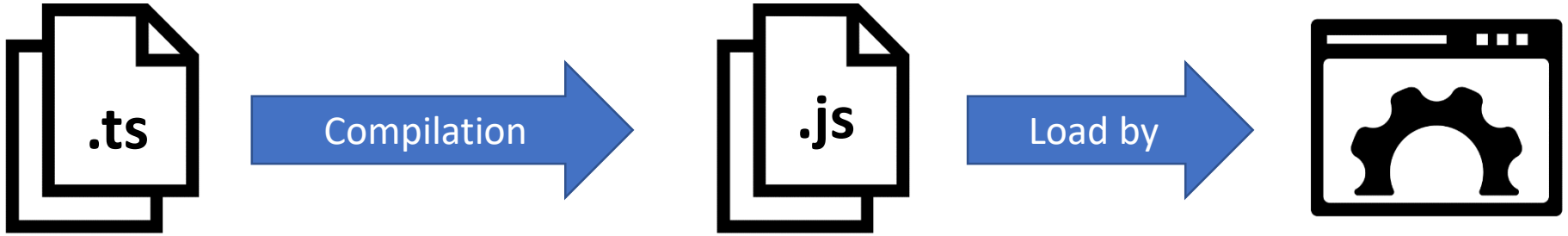
Transpilation : configuration

- Compilation option :

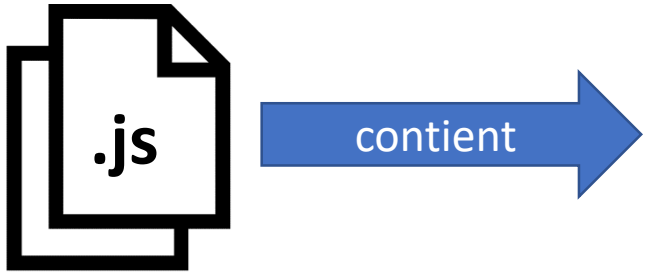
`--target` → ES3 (default), ES5, ES2015, ES2016, ES2017,ES2018 or ESNEXT

Transpilation vs Polyfill

Transpilation ->



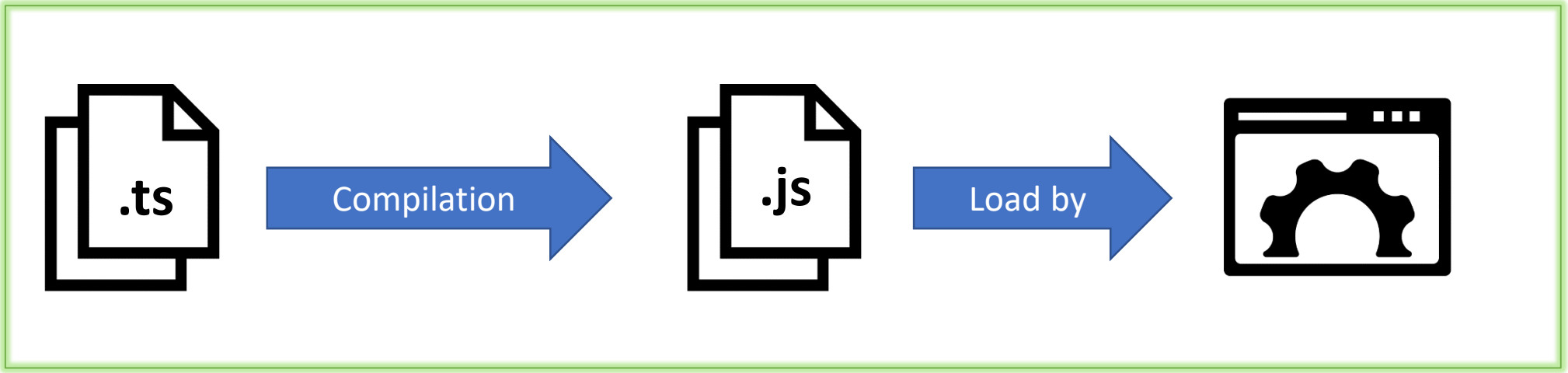
Polyfill ->



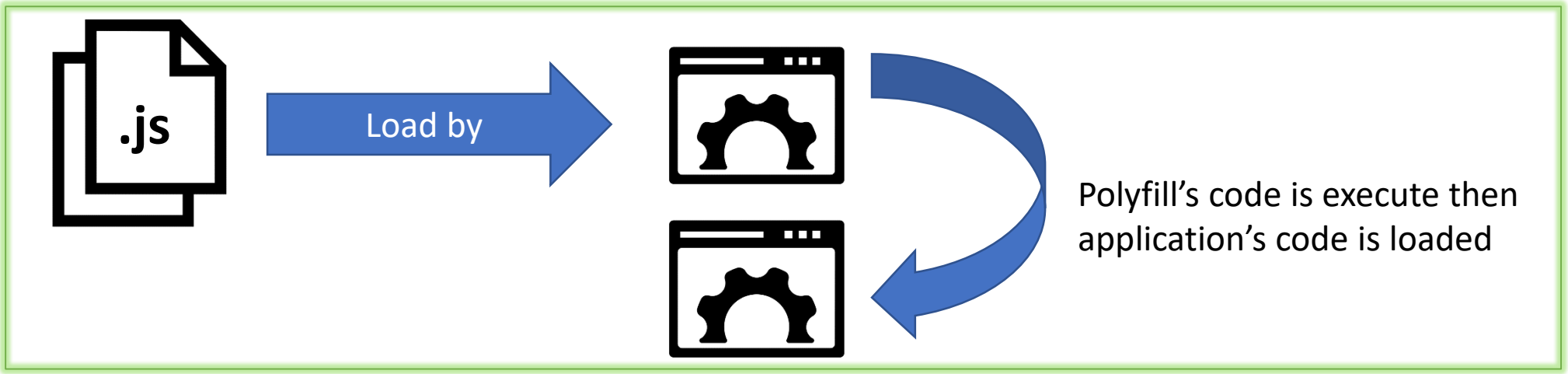

```
if(typeof Array.prototype.filter !== "function") {
  Array.prototype.filter = function() {
    // implementation goes here
  };
}
```

Transpilation vs Polyfill

Transpilation ->



Polyfill ->



Transpilation

- Adapt transpilation level to targeted browsers.

- TypeScript don’t transpile everythings, solution :
  - TypeScript + polyfills library (core-js, es6-shim, …)
  - TypeScript + Babel = ♡

Module

- Export ES2015

```
export class Animal {
    // …
}
```

- Import ES2015

```
import { Animal } from "./animal";
```

- Before : AMD, CommonJS, UMD, System, ES2015.
- Over time there will be only one : ES2015

Module : configuration

- Compilation option :

```
--module
```

none, commonjs, amd, system, umd, es2015, or ESNext

Module

- Use ES2015 -> transpile if needed.
- To prevent ugly import : `import { Animal } from "../../../../../../../core/animal";`

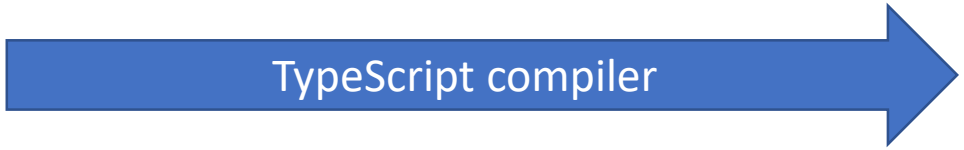
1. In tsconfig.json use aliases path :

```
{
    "compilerOptions": {
        "baseUrl": "./src",
        "paths": {
         "@myProject/utils/*": ["app/utils/*"],
         "@myPorject/core/*": ["app/core/*"]
        }
    }
}
```

2. Don’t forget to also configure this aliases into your bundler’s config file.
3. Result : `import { Animal } from “@myProject/core/animal";`

Enum

color.ts



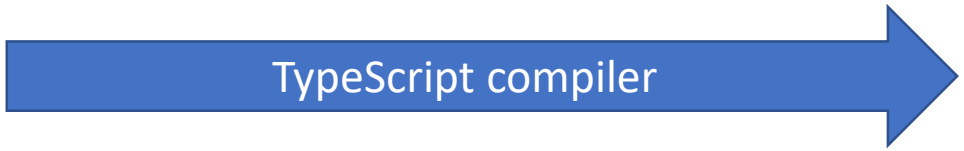
color.js

```typescript
enum Color {
  Red,
  Blue,
  Green
}

let foo: Color = Color.Red;
let bar: string = Color[Color.Red];
```

```js
"use strict";
var Color;
(function (Color) {
    Color[Color["Red"] = 0] = "Red";
    Color[Color["Blue"] = 1] = "Blue";
    Color[Color["Green"] = 2] = "Green";
})(Color || (Color = {}));
let foo = Color.Red;
let bar = Color[Color.Red];
```

Constant Enum

color.ts



color.js

```
const enum Color {
  Red,
  Blue,
  Green
}

let foo: Color = Color.Red;
```

```
"use strict";
let foo = 0 /* Red */;
```

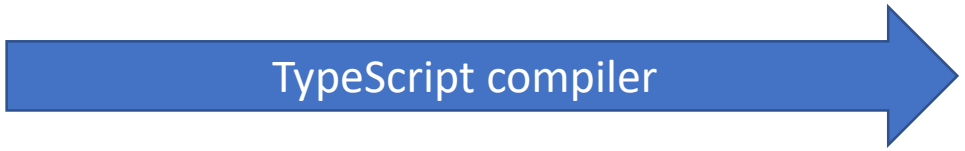
```
let bar: string = Color[Color.Red];
```

Enum

- Use ***const enum*** as much as possible.
- Be careful with this option : `--preserveConstEnums`
- If you acces Enum via index, thinks of Map/Set, Object, …

TypeScript Helper

file.ts



file.js

```ts
class Greeter {
  greeting: string;
  constructor(message: string) {
    this.greeting = message;
  }
  greet() {
    return "Hello, " + this.greeting;
  }
}

class PoliteGreeter extends Greeter {
  //...
}
```

```js
var __extends = (this && this.__extends) || (function () {
    var extendStatics = function (d, b) {
        extendStatics = Object.setPrototypeOf ||
            ({ __proto__: [] } instanceof Array && function (d, b) { d.__proto__ = b; }) ||
            function (d, b) { for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p]; };
        return extendStatics(d, b);
    }
    return function (d, b) {
        extendStatics(d, b);
        function __() { this.constructor = d; }
        d.prototype = b === null ? Object.create(b) : (__.prototype = b.prototype, new __());
    };
})();
var Greeter = /** @class */ (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
}());
var PoliteGreeter = /** @class */ (function (_super) {
    __extends(PoliteGreeter, _super);
    function PoliteGreeter() {
        return _super !== null && _super.apply(this, arguments) || this;
    }
    return PoliteGreeter;
}(Greeter));
```

TypeScript Helper : the trap

- Many helpers exists :

```
function __assign(t: any, ...sources: any[]): any; // Helper de Object.Assign
function __spread(...args: any[]): any[]; // Helper de l'opérateur spread
//...
```

- Generate in each file where are needed -> increase bundle size !!!

TypeScript Helper

- To prevent helpers to proliferate :

1. Install **tslib** : `npm install tslib`
2. Use the following compilation options :

```
--noEmitHelpers
```

```
--importHelpers
```

3. Once done, TypeScript compiler only imports helpers from **tslib**

Typing

Basic typing

- boolean, number, string, array, void, null, undefined, object, any et unknow.
- Example

```
let name: string;

let list: number[] = [1, 2, 3];

function fn(param: boolean): void {
    // Do something
}
```

Basic typing

- Use ***any*** as less as possible !

- Prefer ***unknow*** to ***any*** :

```
let myAny : any = "toto" ;
let myUnknown: unknown = "toto";

let foo: string = myAny;
let bar: string = myUnknown;

myAny.mehtod();
myUnknown.mehtod();
```

Basic typing

- Don’t have to type everythings, let TypeScript compiler inference do !
- Reminder : types are useful only during compilation not at runtime !

Classe and interface (1/2)

```
interface Ninja {
    nbShuriken: number;
    throwShuriken: () => void;
}
```

```
class NinjaTurtle implements Ninja {
    nbShuriken: number;

    constructor() {
        this.nbShuriken = 6;
    }

    throwShuriken(): void {
        // Throw shuriken
    }
}
```

```
let leonardo: Ninja = new NinjaTurtle();
let donatelo: NinjaTutle = new NinjaTurtle();
```

Classe and interface (2/2)

- Optional field :

```
interface Animal {
    name: string;
    say?: () => void;
}
```

- Readonly field :

```
interface Animal {
    readonly name: string;
    say?: () => void;
}
```

Union and intersection

- Union :

```
class Ninja {
    nbShuriken: number;
    throwShuriken: () => void;
}

class Samurai {
    nbKunai: number;
    throwKunai: () => void;
}
```

```
function throwAttack(human: Ninja | Samurai) {
    if (human instanceof Ninja) {
        human.throwShuriken();
    } else {
        human.throwKunai();
    }
}
```

- Intersection :

```
assign<T, U>(target: T, source: U): T & U;
```

Type alias

```
class Ninja {
    nbShuriken: number;
    throwShuriken: () => void;
}

class Samurai {
    nbKunai: number;
    throwKunai: () => void;
}
```

```
type Fighter = Ninja | Samurai;

function throwAttack(human: Fighter) {
    if (human instanceof Ninja) {
        human.throwShuriken();
    } else {
        human.throwKunai();
    }
}
```

Classe vs Interface vs Alias

- Which one use ?
  - Need implementation -> Classe.
  - Need union or intersection -> Alias.
  - Otherwise -> Interface or Alias, make a choice and stick to it ☺

Structural typings vs nominal typings

```
class Order {
  id: number;
}

class User {
  id: number;
  name: string;
}

function processOrder(order: Order) {
  // Do something
}
```

```
const order = new Order();
const user = new User();

processOrder(order);
processOrder(user);
```

Do nominal typgins

- “On TypeScript’s roadmap !” -> Investigation.
- One of many hack to force nominal typings :

```
class Order {
  private __nominal: void;
  id: number;
}

class User {
  private __nominal: void;
  id: number;
  name: string;
}

function processOrder(order: Order) {
  // Do something
}
```

```
const order = new Order();
const user = new User();

processOrder(order);
processOrder(user);
```

Enable stricter type checking (1/2)

- Compilation option : `--strict`

- Master option that enable following sub-options :

```
--noImplicitAny
--noImplicitThis
--alwaysStrict
--strictNullChecks
--strictFunctionTypes
--strictPropertyInitialization
```

Enable stricter type checking (1/2)

- Enable immediately on new project, by default when use : `tsc --init`

- Enable incrementally on existing project :

```
{
  "compilerOptions": {
    "strict": true,
    // "noImplicitAny": false,
    "strictNullChecks": false,
    "strictFunctionTypes": false,
    "strictPropertyInitialization": false,
    "noImplicitThis": false,
    "alwaysStrict": false
  }
}
```

Definition file

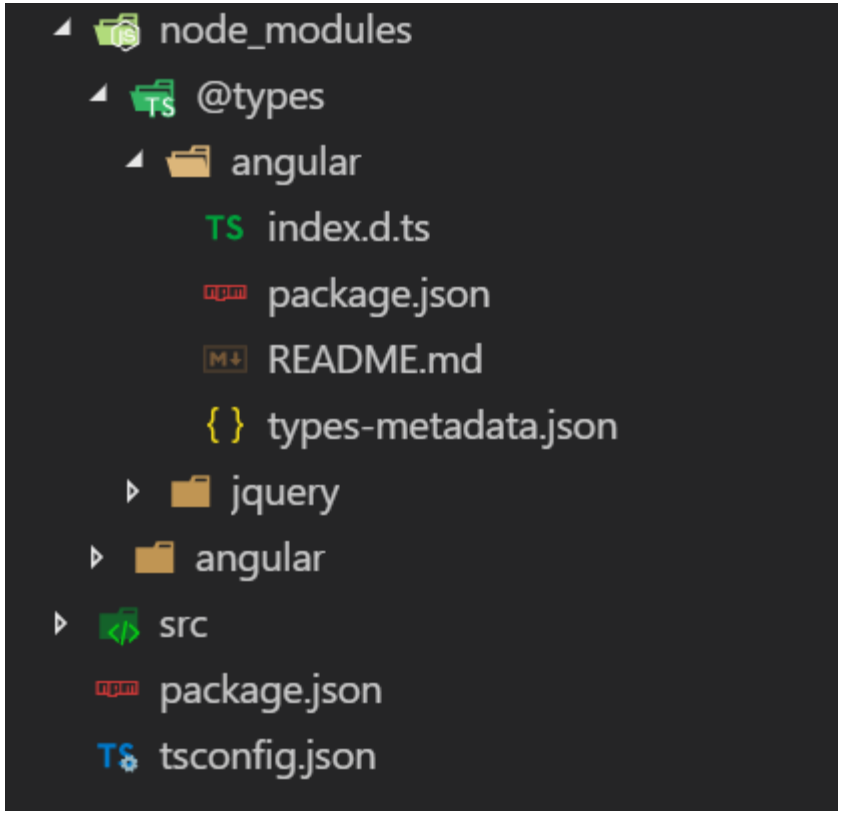
- Describe and type Javasript code.
- NPM package and TypeScript:
  - Write in TypeScript -> 👌
  - Write in JavaScript but ship with his definition file -> 👌
  - Write in JavaScript without his définition file -> ✋
    - Install NPM package définition file (@types).
- TypeScript compiler use definition files for native JavaScript : lib.d.ts

Definition file

- Install : `npm install --save-dev @types/angular`

- package.json :

```
{
    "name": "angularjs-with-dts",
    "version": "1.0.0",
    "dependencies": {
        "angular": "1.5.8"
    },
    "devDependencies": {
        "@types/angular":"1.5.20"
    }
}
```



Definition file

- Alaway install .d.ts files in devDependencies.

- Specify composition of lib.d.ts file according to the native Javascript features you use :

```
{
  "compilerOptions": {
      "target": "es5",
      "lib": [
          "dom",
          "es5",
          "es2015.collection",
          "es2015.iterable"
    ]
  }
}
```

Migration JS -> TS

JavaScript → TypeScript : solution 1

- Analysis JavaScript via TypeScript
  - One file : // @ts-check
  - Globally (tsconfig.json): `--checkJs`

```
function add(numbers) {
  return numbers
   .reduce(function(previous, next) {
      return previous + next;
   });
}

var result = add([true, 2, "3"]);
console.log(result); // 33
```

```
// @ts-check
/**
* @param {number[]} numbers
*/

function add(numbers) {
  return numbers
   .reduce(function(previous, next) {
      return previous + next;
   });
}

var result = add([true, 2, "3"]);
console.log(result); // 33
```

JavaScript → TypeScript : solution 2

- Incremental migration :
  1. Create tsoncifg.config thanks to CLI tsc : `tsc --init`
  2. Set the following compilation option : `--allowJs`
  3. Adapt transpilation level.
  4. Rename gradually file.js → file.ts

Migration JavaScript → TypeScript

- Prefer solution 2, if you can.
- TypeScript can transpile even if errors are decteded.

Conclusion

Conslusion

- Essential to master TypeScript → Compilation options !

- Many subject not addressed :
  - Options needed to use react, angular,…
  - Mapped type and Conditional type
  - …