modern JavaScript

CENTRIQ
TRAINING

Prerequisites

- HTML
- CSS
- JavaScript programming experience

Resources

- The TypeScript Handbook
  - http://www.typescriptlang.org/Handbook
- TypeScript Github
  - https://github.com/Microsoft/TypeScript
- Definitions repository - http://definitelytyped.org/

Resources - secondary

TypeScript

- Creating a TypeScript Workflow with Gulp, Dan Wahlin
  - http://weblogs.asp.net/dwahlin/creating-a-typescript-workflow-with-gulp
- Jonathan Turner's talk on TypeScript in Angular from ng-conf – March 2015
  - https://www.youtube.com/watch?v=Xw93oketp18
- What's new in TypeScript and TypeScript tooling for greater productivity

Intro

Creation

- Anders Hejlsberg (1960 -)
  - Borland - Turbo Pascal, Delphi
  - 1996 Microsoft – J++, 2000 C# lead architect
- Better large-scale development
  - static typed
    - more testable, better IDE support
  - outputs ES 3 or 5

Versions

- 0.8 – Oct 2012
- 1.0 – April 2014
- 1.8 – Jan/Feb 2016
  - 1.8.9 – Mar
- 1.9 – Apr 2016 dev
- 2.6.2 current

Usage

- Local development
  - install transpiler locally, convert JS code and deploy
- Remote development
  - use remote transpiler to convert JS code and deploy
  - download transpiler code, load web app, browser manages transpile

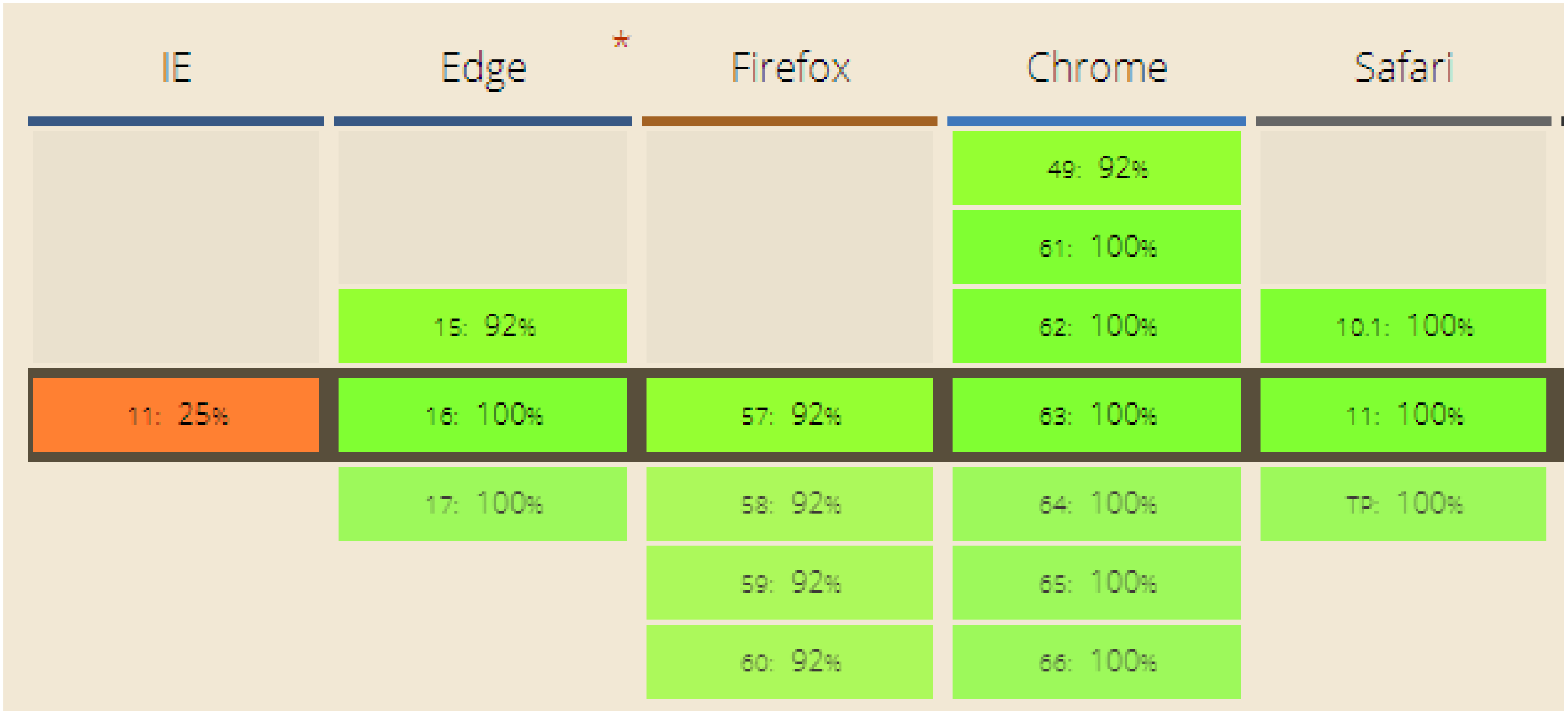
Install TypeScript

TypeScript

- http://www.typescriptlang.org/
- Install node.js to get npm
  - Download from https://nodejs.org
- Install typescript with npm
  - **`npm install -g typescript`**
- Compile a file from the command line
  - **`tsc helloworld`**
- Compile and watch for changes then recompile
  - **`tsc helloworld --watch`**

ECMAScript 6 - 2015

- aka ES6, ECMAScript 2015
  - Firefox supports JavaScript 1.8.5 (ES5)
- TypeScript will transpile ES6 to ES5



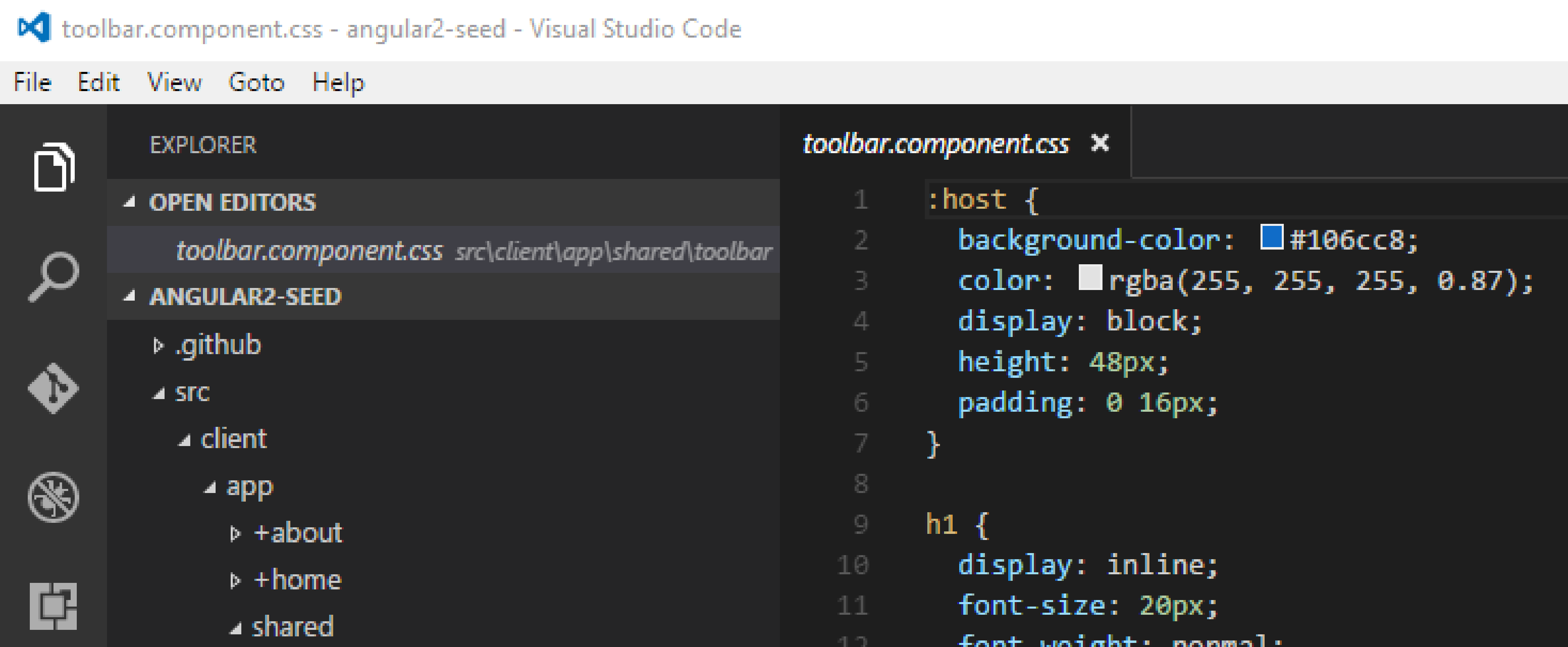
IE	Edge *	Firefox	Chrome	Safari
			49: 92%	
			61: 100%	
	15: 92%		62: 100%	10.1: 100%
11: 25%	16: 100%	57: 92%	63: 100%	11: 100%
	17: 100%	58: 92%	64: 100%	TP: 100%
		59: 92%	65: 100%	
		60: 92%	66: 100%	

Microsoft Visual Studio Code

It's not VS

Type Script

- an Electron project – not like VS



```
:host {
  background-color: #106cc8;
  color: rgba(255, 255, 255, 0.87);
  display: block;
  height: 48px;
  padding: 0 16px;
}

h1 {
  display: inline;
  font-size: 20px;
```

C# support

- C# extensions in Marketplace
  - if you already have a project with C# files, VS Code will prompt you to install the extension as soon as you open a C# file.
  - Lightweight development tools for .NET Core.
  - Great C# editing support, including Syntax Highlighting, IntelliSense, Go to Definition, Find All References, etc.
  - Debugging support for .NET Core (CoreCLR). NOTE: Mono and Desktop CLR debugging is not supported.
  - Support for project.json and csproj projects on Windows, macOS and Linux.

Exercise

- **Set up basic apps**
- **Set up command line environment**

TypeScript support modes

- Project scope
  - one folder
  - one **tsconfig.json** for compiler info, includes
  - **jsconfig.json** at root of project, multiples allowed
    - just about the same as tsconfig
  - one **tasks.json** for task build info
- File scope
  - no jsconfig.json

Set up project

TypeScript

- Create a directory for your files
- Open the directory in Code
- Optional
  - Code task runner scripts - tasks.json
    - F1, type “task”, select **Tasks: Configure Tasks** or **Tasks/Configure Tasks**
    - **Create tasks… Other** or another option

jsconfig.json

- Or use tsconfig.json for TypeScript projects
  - tsc --init
- Exclude lists
  - explicit lists
    - node exclude the node_modules folder
    - bower exclude the bower_components folder
    - ember exclude the tmp and temp folder
    - jspm exclude the jspm_packages folder
    - webpack then exclude the output folder, e.g., dist

Hiding .js files matching .ts files

- File / Preferences / Workspace Settings
  - opens a .vscode/settings.json file for the project
  - use User Settings for all projects
- Use this property in the object
  - "files.exclude": {
  -         "**/.git": true,
  -         "**/.DS_Store": true,
  -         "**/*.js.map": true,
  -         "**/*.js": {"when": "$(basename).ts"}
  -     }

Build project

- Set up includes and excludes in tsconfig.json
  - default includes: **/*
- "include": ["src/**/*"]
- "exclude": ["node_modules"]
- Select an internal build task – Control-Shift-B
  - tsc: build
  - tsc: watch

Run a .js file

- Open integrated terminal (Ctrl-`)
  - **node <**filename with or without .js**>**

or

- Use a browser to run the file
  - Copy and paste the code into the browser console.

or

- Install extension Code Runner
  - right click / Run Code

Run a .ts file

- Install VS Code extension Code Runner
- Install ts-node
  - npm install -g ts-node
- right click / Run Code

.vscode/tasks.json

TypeScript

Tasks

- Run npm tasks from Tasks/Run Task…

```
"tasks": [
    {
        "label": "node",
        "type": "shell",
        "command": "node ${file}",
        "problemMatcher": [
            "$eslint-stylish"
        ]
    }
]
```

<filename>.d.ts

- interface declarations for JavaScript objects
  - lib.d.ts contains definitions for built-in objects, DOM, BOM
- using jQuery with TypeScript
  - set up jQuery as normal
  - get jquery.d.ts from https://github.com/DefinitelyTyped/DefinitelyTyped
  - or nuget it from there with - Install-Package jquery.TypeScript.DefinitelyTyped
  - or the best is to use **tsd** (next slide)
  - compile and run

tsd

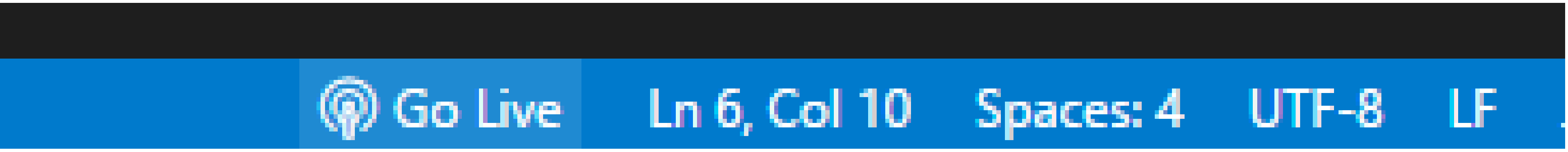
- Install package manager for TypeScript definitions
  - **`npm install tsd –g`**
- Create tsd.json file
  - **`tsd init`**
- Download jQuery definitions
  - **`tsd install jquery --save`**

Running a test server

- Open a terminal window
- Install lite-server with npm globally
  - **`npm install -g lite-server`**
- Type **`lite`**-server to load index.html
  - live-server, http-server are other options that includes refresh on save
- Used with Angular

Running a test server

- Install extension Live Server
- Launch with click on Go Live on bottom of screen



Resources

- https://code.visualstudio.com/blogs

My favorite extensions

- Live Server
- Beautify

- HTML CSS Support
- ESLint
- Easy SASS
- Easy LESS
- Code Runner

Exercises

- Set up Microsoft VS Code environment
- Set up local server

Scope – let

ES6

- lexical scope
  - bounded by function block
  - var mynum = 1;
- block scope
  - bounded by any block
  - let mynum = 1;

const

ES6

- constants
  - have block scope
  - are mutable!
    - like static on variables
  - can not be reassigned
  - not for a class, use static there
- const MYNUM: number = 1;

Types - static type notation

Type
Script

```
let counter;                            // unknown (any) type
let counter = 0;                        // number (inferred)

let counter : number;
let counter : number = 0;
```

Types - basic

```
let height:  number = 6;
let isDone:  boolean = false;
let name:    string = "bob";
let list:    number[ ] = [1, 2, 3];

function noReturn(): void {    }
```

Types - any

- single top type **: any**

```
let notSure: any = 4;
notSure = 'maybe a string instead' ;
notSure = false;
let list: any[ ] = [1, true, 'free'];
```

Types – never, null, undefined

- **never**
  - return type for functions that never return
  - variable type under type guards that are never true.
- **null** and **undefined**
  - types have the values null and undefined

Types - union

- let numberOne: number | string = 1;
- numberOne = '1’;

- let oneOrMany: string | string[ ] = 'a';
- oneOrMany = ['a','b','c'];

Types – string literals

TypeScript

```
type CurrencyCode = "USD" | "EUR" | "GBP" | "AUD";
function convertToUSD(
      amount:number, code: CurrencyCode = 'USD') {
  let exchangeRate = 1.0;
  if (code === "EUR")      { exchangeRate = 1.20673; }
  else if (code === "GBP") { exchangeRate = 1.35527; }
  else if (code === "AUD") { exchangeRate = 0.78609; }
  return amount/exchangeRate + ' ' + code;
}
console.log(convertToUSD(100));
console.log(convertToUSD(100,'EUR'));
```

Type guards

```
function addTwoTo(numberIn: number | string): any {
      if (typeof numberIn === 'number'){
            return numberIn + 2;
      } else {
            return numberIn + "2";
      }
}
console.log(addTwoTo(1));
console.log(addTwoTo('1'));
```

Type aliases

```
type PrimitiveArray =
      Array<string | number | boolean>;


type N = number;
```

- `let aNumber : N = 1;`

```
type NumString = number | string;
```

- `let aNumber : NumString = 1;`

Operators and flow control

Operators - arithmetic

- +, -, *, /
- %
- ++, --

Operators – comparison, logical, bit

ES6

- ==, ===, !=
- >, >=, <, <=

- &&, ||, !

- &, |, ^, ~, <<, >>, >>>

Operators - assignment

ES6

- =
- +=, -=, *=, /=
- %=

Operators - TypeScript

- Exponentiation

```
2 ** 6
Math.pow(2, 6);
```

Flow control - branching

ES6

```
if (condition) { then do this }
if (condition) { then do this } else { do this }

let x = (condition) ? value if true : value if false;

switch (variable) {
```

- `case <value of variable>:`
  - `do stuff;  break;`
- `default:`
  - `do stuff;`

```
}
```

Flow control - for

ES6

```
for (let i: number = 0; i < 9; i++) {
   console.log(i);
}
```

Flow control – for-in

ES6

- for (let property in object) { do stuff }
  - iterates over all enumerable properties of an object
- use with inheritance for just one layer

```
Object.prototype.objCustom = function() { };
let numberLayer = { a:1, b:2, c:3 };
for (let key in numberLayer) {
  if (numberLayer.hasOwnProperty(key)) {
    console.log(key + " is owned by this layer");
  } else {
       console.log(key + " isn't owned by this 
layer");
      } }
```

Flow control – for-of

ES6

- for (let property in object) { do stuff }
  - iterates over enumerable properties of an object, array, string
  - includes values, functions, superclass properties
  - array order not guaranteed
  - most flexible
- for (let property of iterable) { do stuff }
  - iterates over data specifically declared iterable
  - not for objects

Functions

Function return types

```
let getAOne  = function( )          { return 1; }

let getAOne  = function( ) : any    { return 1; }
let getAOne  = function( ) : number { return 1; }

let getZip   = function( ) : void   { return; }
```

ES6

Function declaration types

- function with name
  - `function getOne( ) { return 1; }`
- function assigned to variable
  - `let get1 = function getOne( ) { return 1; };`
- anonymous function assigned to variable
  - `let get1 = function ( ) { return 1; };`
- arrow function assigned to variable
  - `let get1 = ( ) => { return 1; };`
- arrow function assigned to variable, shortest
  - `let get1 = ( ) => 1 ;`

Arrow functions

ES6

- aka lambda function
- Use function syntax for readability

```
function (radius) { return Math.PI * radius ** 2; };

(radius) => { return Math.PI * radius ** 2; }

 radius  =>   Math.PI * radius ** 2 ;
```

Parameters - optional

- ? allows a nulllable type, aka optional parameter

```
function sendNumString (
   firstArg : number, secondArg?: string) : void {
      return;
}
sendNumString(1,'a');
sendNumString(1);
  // secondArg now has type Undefined
```

Parameters - default

- Parameter with default value
- Optional parameters with default value not allowed. Use || to default values with x? in setup of function

```
function order(
   meal: string, drink : string = 'water') : void {
      console.log('You ordered', meal, drink);
}
order('a hamburger');
order('a cheeseburger', 'Pepsi');
```

Rest parameter

TypeScript

- aka varargs

```
function order(
   meal: string, ...extras: string[]) : void {
   console.log('You ordered', meal, extras.join(',
'));
}


order('a hamburger');
order('a cheeseburger', 'pepsi', 'fries', 'onion
rings');
```

typescript/spread.ts

ES6

Spread operator - arrays

- Inverse of rest parameter
- Use in place of concat( )

```
let entrees:string[] = [‘hamburger’];
let sideOrders:string[] = [‘fries’,’onion rings’];
let order:string[] =
   [...entrees, ...sideOrders, ’Pepsi’];
```

typescript/spread.ts

ES6

Spread operator – immutable objects

```
let order1 = {
    entree: `hamburger`,
    drink: `Pepsi`,
    sideOrder: `fries`
};
let update = {sideOrder: `onion rings`};
let order2 = {...order1, ...update};
console.log(order2);

// similar to:
let order3 = Object.assign({}, order1, update);
console.log(order3);
```

Template strings

TypeScript

- backticks delimit a string with ${ } variables

```
function order (meal: string) : string {
      return `Ordered a ${meal}`;
}
```

Specialized overloading

TypeScript

```
function order (meal: number) : string;
function order (meal: string) : string;
function order (meal: any)    : string {
      switch (typeof meal) {
            case 'number':
                  return `Ordered meal #${meal}`;
            case 'string':
                  return `Ordered a ${meal}`;
            default:
                  return 'Ordered something.'
} }
console.log(order(5));
console.log(order(`General Tso's chicken`));
console.log(order(true));
```

Hoisting

ES6

- function declarations are available in scope anywhere
  - `console.log(whereIAm());`

  -
    ```
    function whereIAm(): string {
      return "I'm declared after..."; }
    ```
- function variables are available after assignment

  -
    ```
    let whereIAmNow = function(): string {
      return "I'm declared first..."}
    ```
  - `console.log(whereIAmNow());`
- TypeScript does not warn

Restricting a function variable’s return type

```
let f_arrowTyped: ( ) => string;
      // restricts to a return type of string

function getString( )     : string {
   return ‘a string’;
}
function send(msg: string): void { }

f_arrowTyped = getString;
f_arrowTyped = send;   // error
```

Callbacks

- the function argument passed to a function and executed when complete
- higher-order function – the function accepting a callback

```
function doSomething(callback : ( ) => void) {
    console.log('Did something.');
    callback();
}
function thenPrintDone() : void {
    console.log('Done.');
}
doSomething(thenPrintDone);
```

Generics – functions, classes

- useful with inheritance
  - also can implement on a class e.g. class Bag<T> { }

```
class Bird { }
class Lizard { }

function sellBirds( pets: Bird[ ]) : void { }
function sellLizards (pets: Lizard[ ]) : void { }

function sellPets<T> (pets : T[ ]) : void { }
```

Classes and objects

Class declaration, instance vars

- block scope for let and class members

```
{
class Dog {
      name: string;
      age: number;
}
let fido: Dog = new Dog();
console.log(fido);
}
```

Scope

- **private** restricts access

```
class Gift {
      private contents : string;
}
let xmasPresent: Gift = new Gift();
console.log(xmasPresent.contents);   // error
```

Accessors

- use related field/property names

```
class Gift {
   private _contents : string;
   get contents( ): string {
      return this._contents;
   }
   set contents(incoming: string) {
      this._contents = incoming; }
   }
let xmasPresent: Gift = new Gift();
xmasPresent.contents = "pair of socks";
console.log(xmasPresent.contents);
```

Static properties

- one value for all class objects

```
class MarshallsGift {
      static storeName : string = `Marshall’s`;
      giftName         : string;
}

console.log(MarshallsGift.storeName);
```

Static methods

ES6

- static method declaration in a class

```
Class.staticMethod()
```

typescript/DogWithConstructor.ts

Constructor

- initialize fields in constructor

```
class Dog {
      private name: string;
      private age: number;
      constructor(name? : string, age?: number) {
            this.name = name || 'Rover' ;
            this.age = age || 5;
      }
}
{
  console.log(fido);
  let fido: Dog = new Dog();
}
```

Interfaces

- Useful for enforcing class structures

```
interface HasBooleanCheck {
      result: boolean;
      isTrue(),
      isFalse(): boolean;
}
class ClassWithBooleanCheck implements
HasBooleanCheck {
      private result: boolean;
      constructor() { this.result = false;  }
      isTrue(): boolean { return this.result;}
      isFalse(): boolean { return this.result;}
}
```

typescript/interface-data.ts

Interfaces

- More useful for type checking data structures

```
interface Order{
  entree: string,
  drink?: string,
  sideOrder? : string,
  total: number
}
function putOrderIn(anOrder: Order){
  console.log(anOrder); }

putOrderIn({entree:'hamburger', total:4.95});
putOrderIn({entree:'cheeseburger', drink: 'Pepsi',
total:6.95});
putOrderIn({entree:'cheeseburger', sideOrder: 'fries', drink:
'Pepsi', total:8.95});
```

Inheritance

- multiple inheritance is not supported
- super( ) calls superclass constructor

```
class Teacher extends Person {
   constructor( name ) {
      super(name);
   }
}
```

Enums

- More restricted in generated code: const enum

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

let c: Color = Color.Green;
console.log(c);
```

Code units of namespaces, modules

Structure

Namespaces

- namespaces = internal modules , JavaScript objects
  - used with <script>
- all members are private
  - export makes unit public

```
namespace app {
  export class UserModel {  }
}
namespace app.entities { }
```

Modules

ES6

- better code reuse, stronger isolation, better tooling support
- any file containing a top-level import or export is a module
- **export** from module, **import** into code
- Compile requires target of module loader

Modules

ES6

```
//file module1.ts
export function
```

Module config

- Add a target and module preference to compile to in **tsconfig.json**

```
 "compilerOptions": {
            "target": "ES5",
            "module": "commonjs",
            "emitDecoratorMetadata" : true
    }
```

Modules - export

- Define and declare exports
  - // filename: pets.ts
  - class Dog{ }
  - class Cat{ }
  - export { Dog, Cat };
- Declare export at definition
  - export class Dog { }
- Use alias
  - export { Dog as Chien, Cat as Chat };

Modules - import

- use the .ts file as the source – it’s a module

```
import { Chien, Chat as Katze} from "./pets“
import * as All from “./pets”

let rover : Chien = new Chien();
let kitty : Katze = new Katze();
```

End

- André Staltz – All JS Libraries Should Be Authored in TypeScript – Mar 2016
  - http://staltz.com/all-js-libraries-should-be-authored-in-typescript.html