
# modern JavaScript

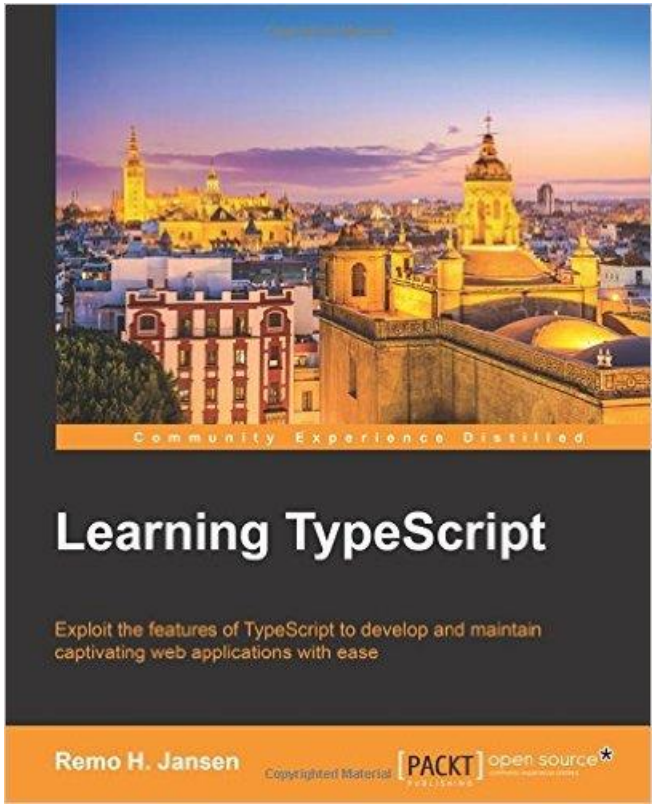
CENTRIQ TRAINING

# Prerequisites

- HTML
- CSS
- JavaScript programming experience

# Book

- **Learning TypeScript by** Remo H. Jansen
- Packt Publishing, September 29, 2015



# Resources

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

# Resources - secondary

- 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

TypeScript

- 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

TypeScript

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

# Use cases

TypeScript

- 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

- 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`

# Tools

- TypeScript playground (online compiler)
  - http://www.typescriptlang.org/Playground

# Types - static type notation

TypeScript

- var counter; // unknown (any) type
- var counter = 0; // number (inferred)
- var counter : number; // number
- var counter : number = 0; // number, initialized

- // not the same as a static class member

# Types - any

- single top type called **Any** type
- keyword **any**

- var notSure: any = 4;
- notSure = "maybe a string instead";
- notSure = false;
- var list:any[ ] = [1, true, "free"];

# Types - basic

TypeScript

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

- function noReturn(): **void** { }

# Types – 2.0

TypeScript

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

# enums

TypeScript

- enum Color {Red, Green, Blue};
- var c: Color = Color.Green;
- console.log(c);

# Scope – var vs let

TypeScript

- function / **lexical** scope
  - **var** mynum : number = 1;
- **block** scope
  - **let** mynum : number = 1;
  - **const** MYNUM: number = 1;

# Union types

TypeScript

- var numberOne: number | string = 1;
- numberOne = '1';

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

# 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

TypeScript

- type PrimitiveArray = Array<string|number|boolean>;
- type N = number;
  - var aNumber : N = 1;
- type NumString = number | string;
  - var aNumber : NumString = 1;

# Operators and flow

# Operators - arithmetic

TypeScript

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

# Operators – comparison, logical, bit

TypeScript

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

- &&, ||, !

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

# Operators - assignment

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

# Non-null assertion

TypeScript

- 2.0 - ! post-fix expression operator asserts operand is non-null and non-undefined in contexts where the type checker is unable to conclude that fact.
- Specifically, the operation x! produces a value of the type of x with null and undefined excluded.

# Flow control - branching

TypeScript

- **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 - iteration

TypeScript

- **do** { stuff } **while** ( condition );
- **for** ( var property **in** object) { do stuff }

- // use for-in with inheritance
- var numberLayer : any = { a:1, b:2, c:3 };
- for (var key in numberLayer) {
- if (numberLayer.**hasOwnProperty**(key)) {
- console.log(key + " is owned by this layer");
- }
- }

# Flow control - iteration

TypeScript

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

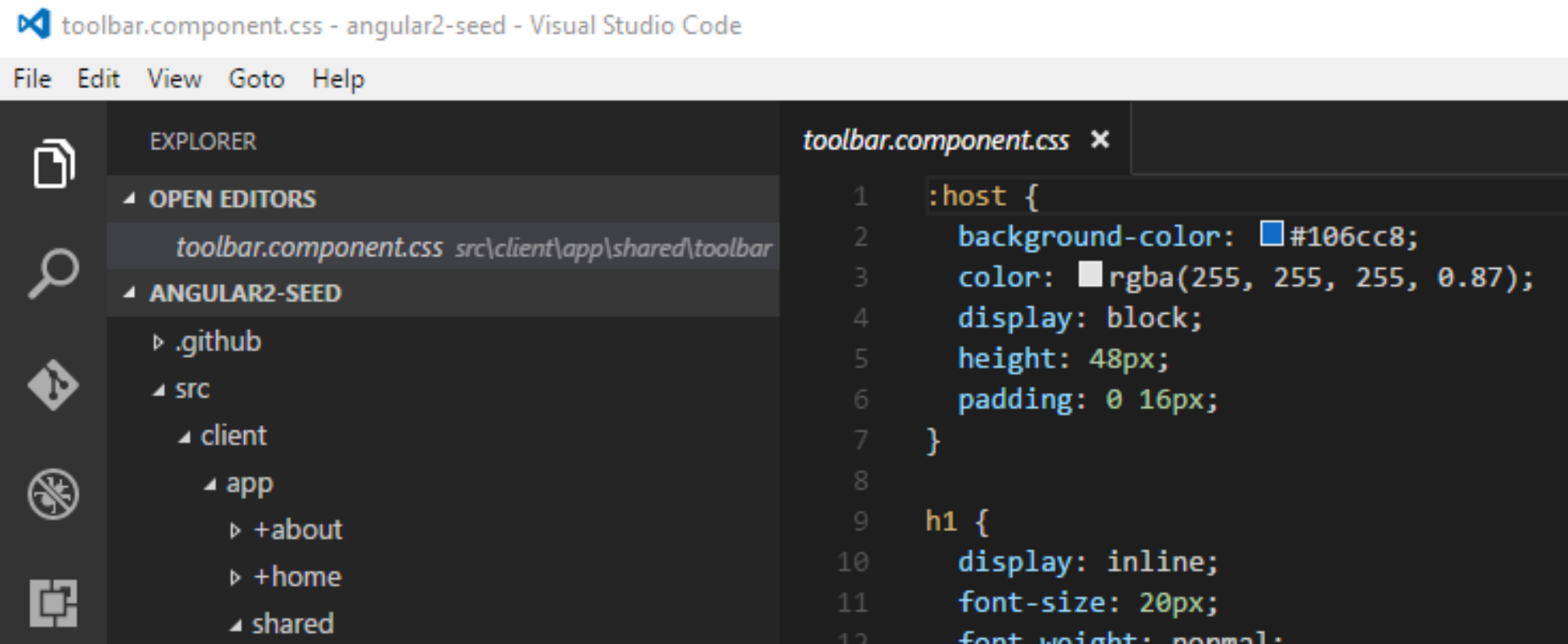
Type Script

Visual Studio Code

# Microsoft Visual Studio Code

# It doesn't look like VS

- an Electron project – not like VS



# Install C# option

- C# language support is now an optional install from the Marketplace.
- Install it from within VS Code by searching for 'C#‘
  - 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.

# Exercise

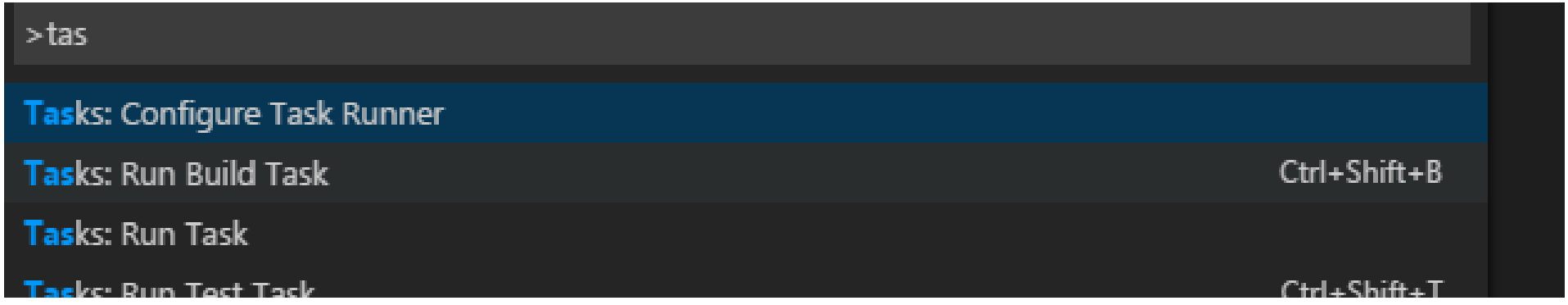
- Set up basic apps
- Set up command line environment

# TypeScript support modes

- Project scope
  - one folder
  - one **tsconfig.json** for compiler info, includes
  - **jsconfig.json** at root of project, multiples allowed
  - one **tasks.json** for task build info

# Set up project

- Create a directory for your files
- Open the directory in Code
- Write tsconfig.json
  - Start with empty object { }
- Generate tasks.json
  - F1, type “task”, select **Tasks: Configure Task Runner,** save



# jsconfig.json

TypeScript

- 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.
- run the **Reload JavaScript** command to ensure that everything is up to date

# Build project

- Run the build task
  - Ctrl+Shift+B
    or
  - F1, type build

# tasks.json

TypeScript

- for running any command line task
- {
- "version": "0.1.0",
- "command": "cmd",
- "isShellCommand": true,
- "args": ["/C"],
- "tasks": [ { see next slides }, { }, { }, … ]
- }

# tasks.json - tasks

- replacement for current build task
- {
- "taskName": "build",
- "args": ["tsc -p", "."],
- "isBuildCommand": true, // runs with Ctrl-Shift-B
- "suppressTaskName": true,
- "problemMatcher": "$tsc"
- }

# Run the file

- Open integrated terminal (Ctrl-`)
- Use node to run the file
  - **`node`** **<**filename with or without .js**>**

or

- Use a browser to run the file (or node)
  - Copy and paste the code into the browser console.

# tasks.json - tasks

- run currently selected file in node shell
- {
- "taskName": "run-js-node",
- "suppressTaskName": true,
- "args": ["**node** ${file}"]
- }

# <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

TypeScript

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

# tasks.json - tasks

- {
- "taskName": "tsd-init",
- "suppressTaskName": true,
- "args": ["**tsd** init"]
- }, {
- "taskName": " tsd-jquery",
- "suppressTaskName": true,
- "args": ["**tsd** install jquery --save"]
- }

# Running a test server

- Install lite-server with npm globally
  - `npm install -g lite-server`
- From Code, type **F1** and ‘**prom**’ <enter>
  or
  Control-Shift-C
- Type **`lite`**-server to load index.html
- live-server, http-server are other options that includes refresh on save

# Hiding .js files matching .ts files

TypeScript

- 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"}
  - }

# Resources

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

# 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

# Functions

# Function return types

TypeScript

- var getAOne  = function( )  { return 1; }
- var getAOne  = function( ) : **any** { return 1; }
- var getAOne  = function( ) : **number** { return 1; }
- function getNothing() : **void** {
-  return;
- }

# Function declaration types

TypeScript

- function declaration with name
  - **function getOne( ) : number** { return 1; }
- function declaration assigned to variable
  - let getADigit = **function getOne( ) : number** { return 1; }
- anonymous function declaration assigned to variable
  - let getAOne = **function ( ) : number** { return 1; }
- arrow function declaration assigned to variable
  - let getOneDigit = **( ) : number =>** { return 1; };

# Arrow / lambdas

TypeScript

- Arrow aka lambda function
  - a variation on the anonymous function

- function (radius) { return Math.PI * Math.pow(radius, 2); };
  - can be written
- (radius) => { Math.PI * Math.pow(radius, 2); }
- radius  => Math.PI * Math.pow(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

TypeScript

- Parameter with default value
  - function order(meal: string, **drink : string = 'water'**) : void {
  - console.log('You ordered', meal, drink);
  - }
  - order('a hamburger');
  - order('a cheeseburger', 'pepsi');
- Optional parameters with default value not allowed. Use || to default values with x? in setup of function

# 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');

# Template strings

TypeScript

- backticks delimit a string with ${ } variables
- function order (**meal**: string) : string {
- return **`Ordered a ${meal}`**;
- }

# Tag function – used with template…

Type Script

- function tagFunction( **literals : string[ ]**, **...placeholders**) {
- let result = ' ';
- placeholders.forEach( function (value, i) {
- result += literals[i] + value.replace(/\d/g, 'a digit');
- });
- result += literals[literals.length - 1]; // not tied to arg
- return result;
- }

# Tagged template

TypeScript

- var arg1 :string = '1';
- var arg2 :string= 'two';
- var taggedTemplateResult =
  **tagFunction** `literal1 ${arg1} literal2 ${arg2} literal3`;

- console.log(taggedTemplateResult);

# Union type parameters

Type Script

- function hitPoolBall(ball: string | number) : void {
- console.log( `You pocketed the ${ball} ball` );
- }
- hitPoolBall('blue and white')
- hitPoolBall(10);
- hitPoolBall(true);  // error

# 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

TypeScript

- function declarations are available in scope anywhere
  - console.log(guessWhereIAm());
  - // console.log(guessWhereIAmNow());
  - function guessWhereIAm(): string {
    return "I'm down here..."; }
- function variables are available after assignment
  - var guessWhereIAmNow = function(): string { return "I'm up here..."}
  - console.log(guessWhereIAmNow());
- TypeScript does not warn

# Restricting a function variable's return type

TypeScript

- function send(aMessage: string) : void { }
- function getString( ): string { return 'a string'; }

- var arrowTyped: ( ) => string;
  - // restricts to a return type of string
- arrowTyped = send;  // error
- arrowTyped = getString;

# Callbacks

TypeScript

- callback – the function argument passed to a function and executed when complete
- higher-order function – the function accepting a callback
- callback with arrow function
  - function thenPrintDone() : void {
  - console.log('Done.');  }
  - function doSomething(callback : ( ) => void) {
  - console.log('Did something.');
  - callback();  }

  - doSomething(thenPrintDone);

# Generics – functions, classes

TypeScript

- class Bird { }
- class Lizard { }
- function sellBirds( pets: Bird[ ]) : void { }
- function sellLizards (pets: Lizard[ ]) : void { }
- function sellPets<T> (pets : T[ ]) : void { }

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

# Classes and objects

# Class declaration, instance vars

TypeScript

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

# Scope

TypeScript

- default var scope in classes is public
  - there is no protected / friend
- **private** restricts access
  - class Gift {
  - **private** contents : string;
  - }
  - var xmasPresent: Gift = new Gift();
  - console.log(xmasPresent.contents);  // error

# Accessors

TypeScript

- requires ES5
- class Gift {
- private _contents : string;
- **get contents( )**: string { return this._contents; }
- **set contents(incoming: string)** { this._contents = incoming; }
- }
- var 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);

# Constructor

Type Script

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

# Interfaces

Type Script

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

# Inheritance

TypeScript

- multiple inheritance is not supported
- class Teacher extends Person {
  - constructor( name ) {
  - super(name);
  - }
- }

Type Script

Code units of namespaces, modules

# Structure

# Namespaces

TypeScript

- 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

TypeScript

- modules = external modules (TS 1.5 – ES6)
  - declared dependencies
  - better code reuse, stronger isolation, better tooling support
  - recommended
  - any file containing a top-level import or export is considered a module (ES 2015)
  - **export** from module, **import** into code = require( )
- Compile requires target of module loader

# Module config

- Add a target and module to compile to in tsconfig.json
  - "compilerOptions": {
  - "target": "ES5",
  - "module": "commonjs",
  - "emitDecoratorMetadata" : true
  - }

# Modules - export

TypeScript

- 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

TypeScript

- use the .ts file as the source – it’s a module
- import { Chien, Chat as Katze} from "./pets“
- import * as All from “./pets”

- var rover : Chien = new Chien();
- var kitty : Katze = new Katze();

# Split module

TypeScript

- One file with an interface
- 
```
module Person {
    export interface PersonInterface {
        name: string;
        hungry: boolean;
        feed();
    }
}
```

# Split module

- Second file refers to and adds to class body
- 
```
/// <reference path="Person.ts" />
module Person {
    export class Person implements PersonInterface {
    name: string;
    age: number;
    hungry: boolean = true;
    constructor(name: string, age?: number) {
        this.name = name; this.age = age;
    }
    feed() {
        this.hungry = false;
        return 'Yummy!';
    }
    }
}
```

# 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