TypeScript For C# Developers

Jesse Liberty
@Jesseliberty
http://jesseliberty.com

Getting Acquainted with TypeScript

Anders Hejlsberg

- Delphi
- Turbo Pascal
- C#
- TypeScript

Key Features

- Static Typing
- Classes
- Constructors, Properties, Methods
- Interfaces
- Arrow Functions (Lambdas)
- Encapsulation

Key Features

Supports Standard JavaScript code

Open Source

Runs Anywhere

Excellent Tool Support

Intellisense

Syntax Checking

TypeScript is JavaScript

TypeScript is a typed superset of Javascript

TypeScript is a first class language

TypeScript is a natural for JavaScript and C# programmers

Risks for C# Programmers

Syntax is similar but not the same

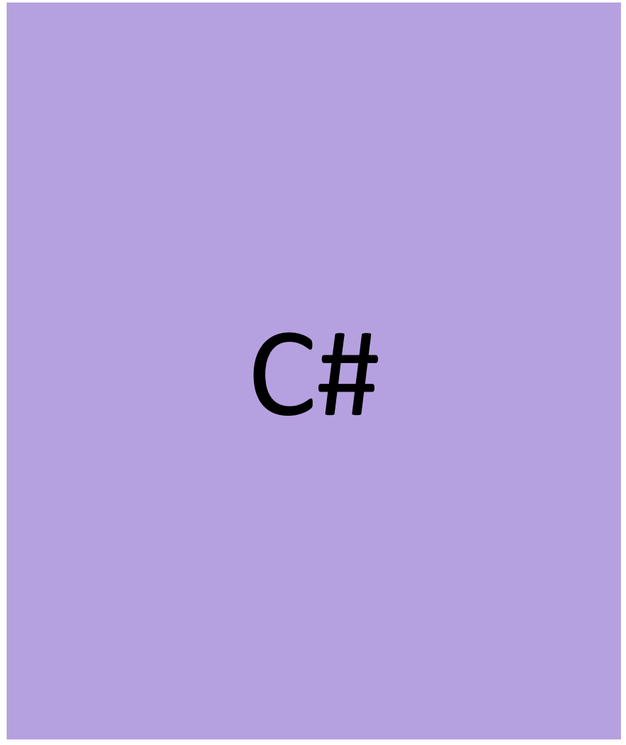
- Watch for subtle differences (e.g., = = vs. = = =)

Classes and Interfaces are similar but not the same

This is a *scripting* language

TypeScript is, ultimately JavaScript with a C#-ish shell

What do you need to know?



Why use TypeScript?

TypeScript Type Safety

- Variables can hold one type
- Variable types can *not* change during run-time
- Only explicit type conversion
- Easy to maintain

Tooling

1

Visual Studio

It just works

2

Otherwise…

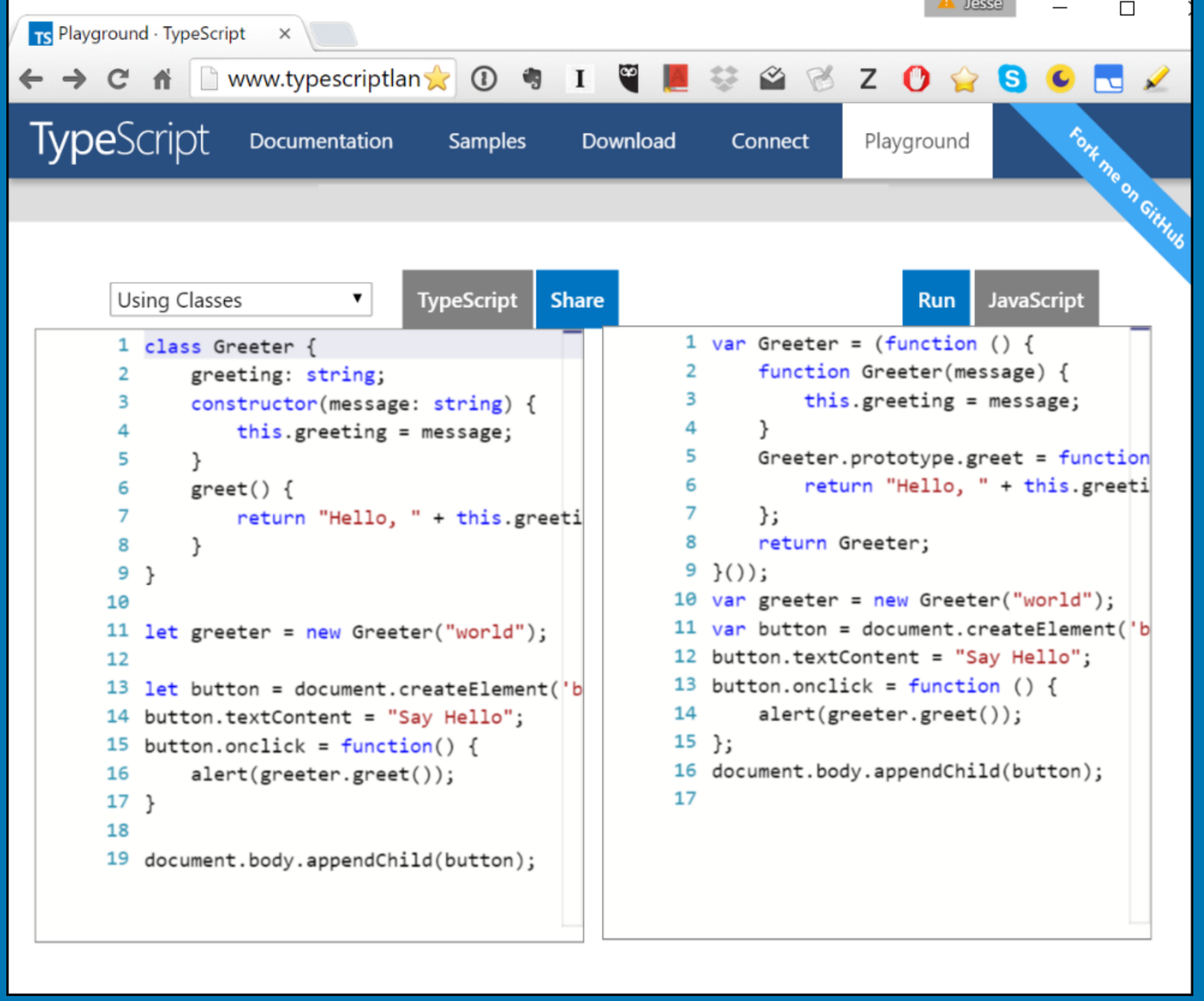
Any browser

Any editor

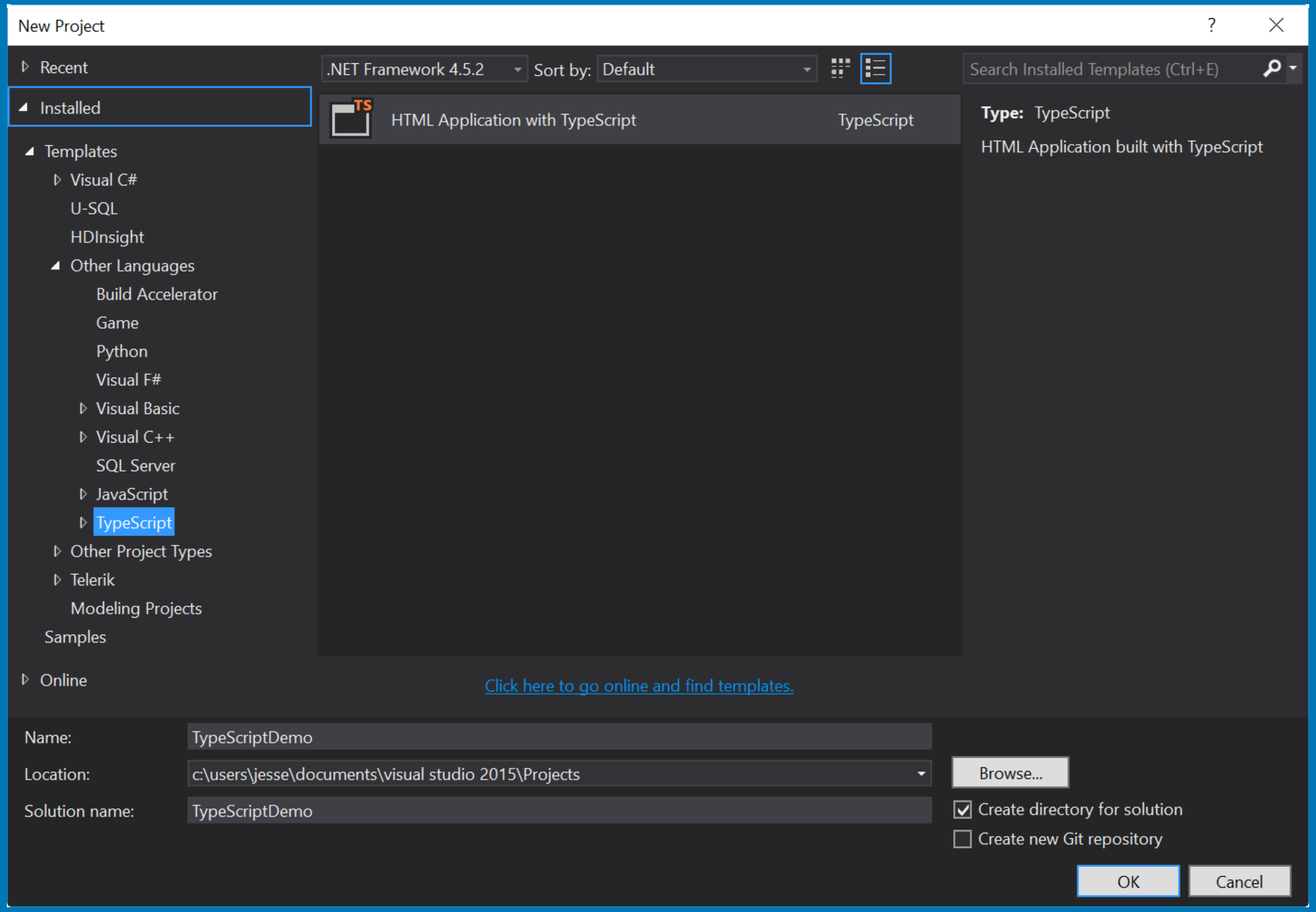
Any OS

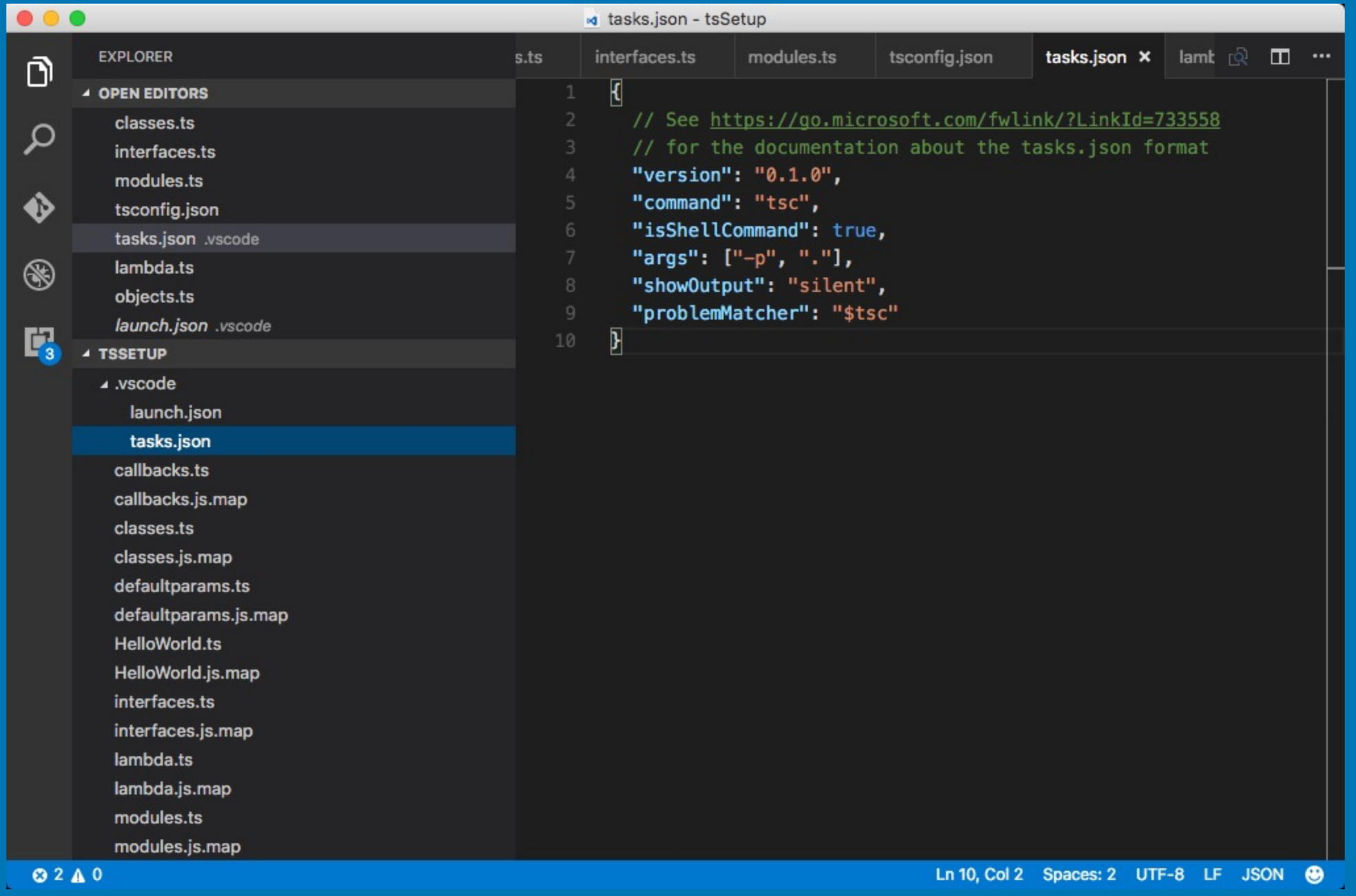
3

Playground



http://typescriptlang.org


New Project
Recent
Installed
Templates
Visual C#
U-SQL
HDInsight
Other Languages
Build Accelerator
Game
Python
Visual F#
Visual Basic
Visual C++
SQL Server
JavaScript
TypeScript
Other Project Types
Telerik
Modeling Projects
Samples
Online
.NET Framework 4.5.2
Sort by:
Default
Search Installed Templates (Ctrl+E)
HTML Application with TypeScript
TypeScript
Type: TypeScript
HTML Application built with TypeScript
Click here to go online and find templates.
Name:
TypeScriptDemo
Location:
c:\users\jesse\documents\visual studio 2015\Projects
Browse...
Solution name:
TypeScriptDemo
Create directory for solution
Create new Git repository
OK
Cancel


tasks.json - tsSetup
EXPLORER
OPEN EDITORS
classes.ts
interfaces.ts
modules.ts
tsconfig.json
tasks.json .vscode
lambda.ts
objects.ts
launch.json .vscode
TSSETUP
.vscode
launch.json
tasks.json
callbacks.ts
callbacks.js.map
classes.ts
classes.js.map
defaultparams.ts
defaultparams.js.map
HelloWorld.ts
HelloWorld.js.map
interfaces.ts
interfaces.js.map
lambda.ts
lambda.js.map
modules.ts
modules.js.map
s.ts
interfaces.ts
modules.ts
tsconfig.json
tasks.json
lamb
{
// See https://go.microsoft.com/fwlink/?LinkId=733558
// for the documentation about the tasks.json format
"version": "0.1.0",
"command": "tsc",
"isShellCommand": true,
"args": ["-p", "."],
"showOutput": "silent",
"problemMatcher": "$tsc"
}
2 0
Ln 10, Col 2
Spaces: 2
UTF-8
LF
JSON

VS Code - Setup

- Download and install VS Code (https://code.visualstudio.com/)
- Download and install Node (http://nodejs.org)
- Run VS Code from terminal by typing *code*

VS Code - Setup

- Add *tsconfig.json* file (http://jliberty.me/tsconfig)
- Open Command Palette (shift-command-p)
  type in *Configure Task Runner*
- Click on *TypeScript – tsconfig.json*

User Preferences

- In VS Code, enter Shift-Command-P
- Type Pref and choose Open User Settings
- On left are the default settings
- On right are the user settings

User Preferences

```
"editor.fontFamily": "Calynda",
"editor.fontSize": 14,
"files.trimTrailingWhitespace": true,
"editor.formatOnType": true,
"editor.tabSize": 3,
"editor.wrappingColumn": 120,
"files.autoSave": "afterDelay",
"files.autoSaveDelay": 1000,
```

User Preferences

```
"files.exclude": {
    "**/.git": true,
    "**/.DS_Store": true,
    "**/*.js": {"when":"$(basename).ts"},
    "**/*.js.map":true
  },
```

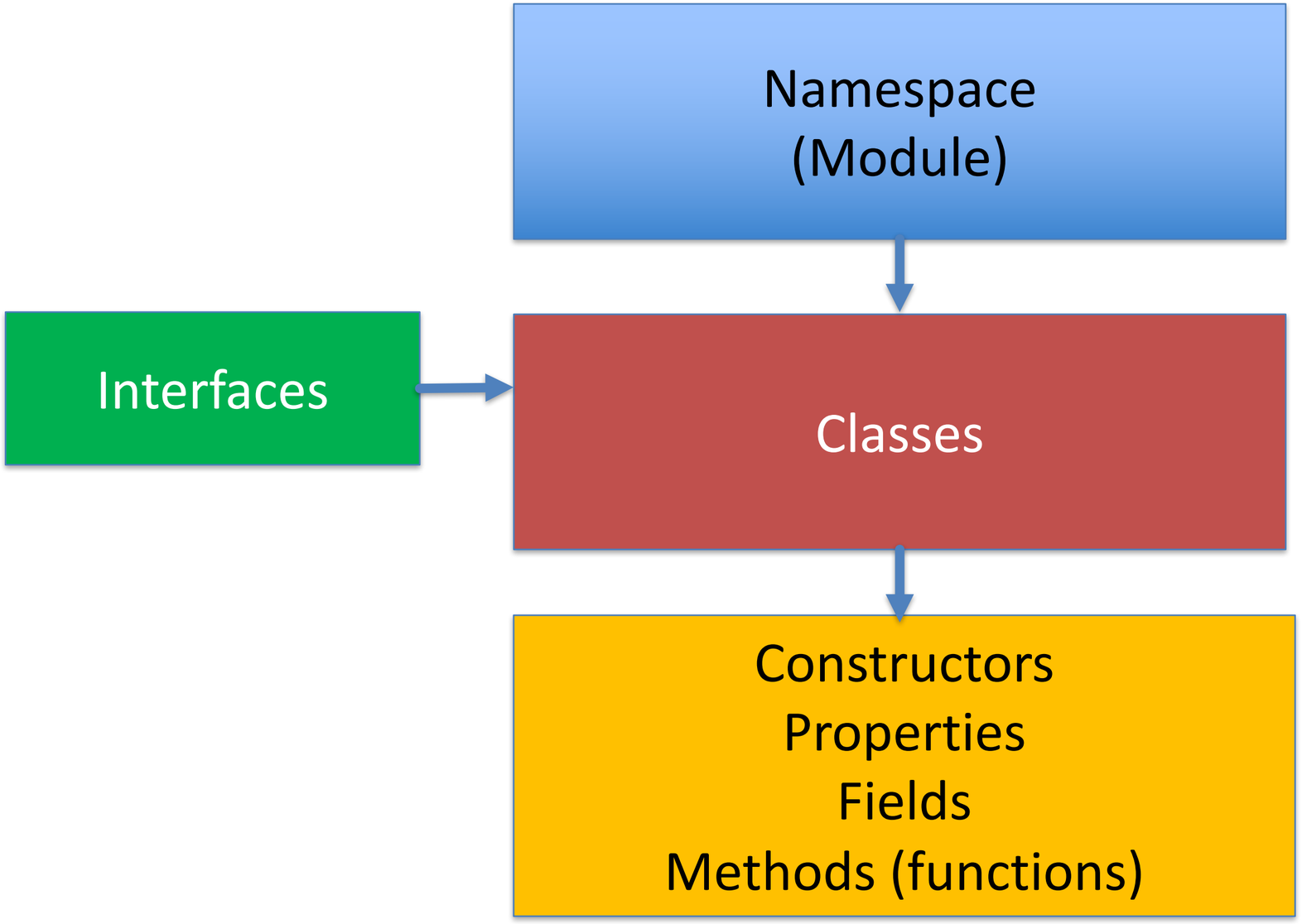
Hello World Demo – note intellisense

Hello World

```
export class HelloWorld{
    SayHi() {
      console.log("hello world”)
    }
}

var helloWorld = new HelloWorld();
helloWorld.SayHi();
```

Classes, Properties and Methods; oh my!


Namespace
(Module)
Interfaces
Classes
Constructors
Properties
Fields
Methods (functions)

TypeScript and Types

TypeScript and Types

- Boolean (true or false)
- Number (6, 5.3, 200.47)
- String ("Hello World")
- Enum (temperature.hot)
- Array (myArray[4])

TypeScript and Types

- Void (function does not return a value)
- Any (undermines type safety)
- Undefined (much like null)

Hoisting

- Var - scoped to the function – or if no function: globally (yuck)
- Let – scoped to the block
- const – scoped to the block

Inferring and setting types

```
let age;                    // type: any
let age = 21;               // type: number (inferred)
let age: number;            // type: number  (explicit)
let age: number = 21;       // type: number
```

Arithmetic Operators

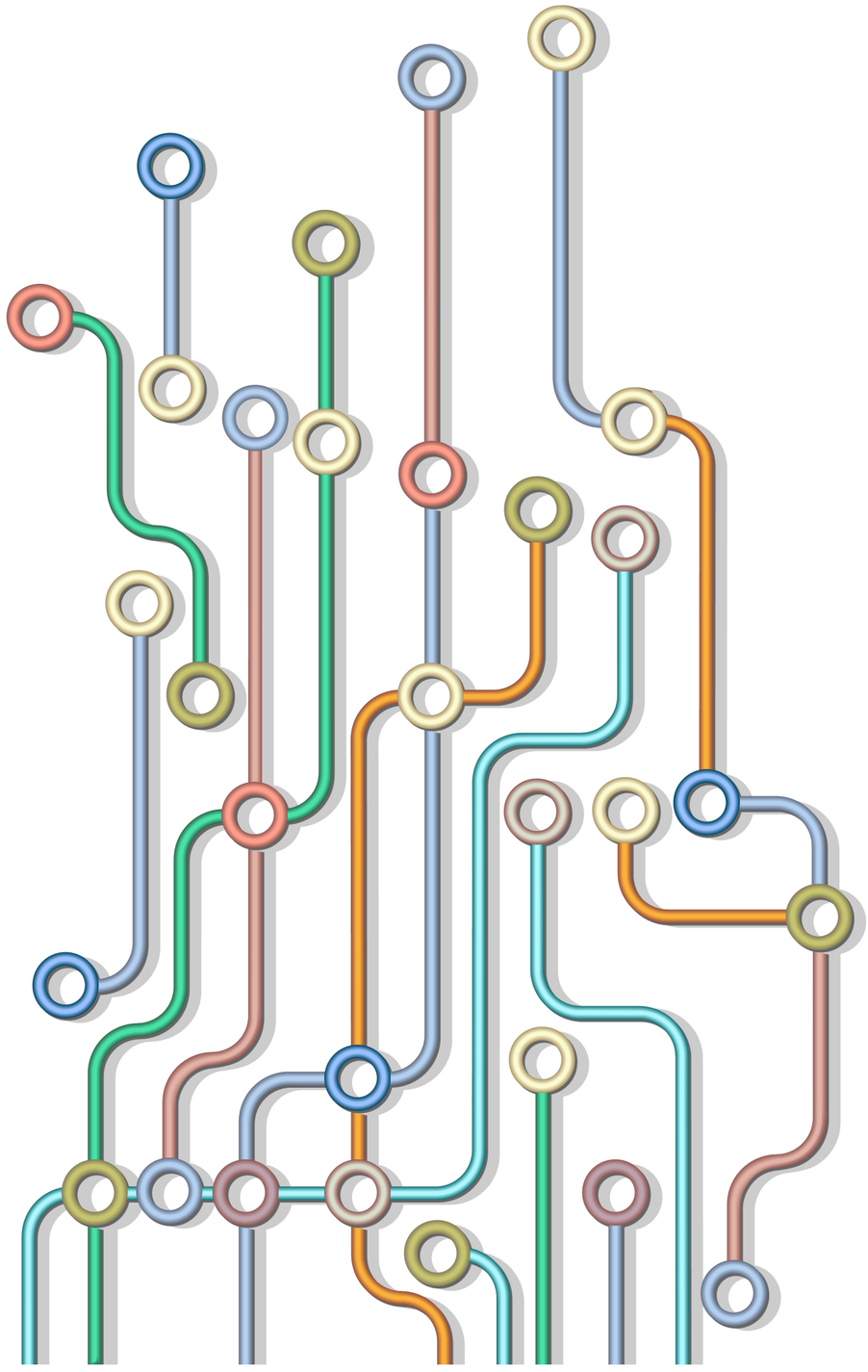
- +, -, *, /, %, ++, -- // just as in C#
- = // assignment
- == // values are equal
- === // values **and types** are equal

```
let a = 10;
a == 10    // true
a === 10 // true
```

```
a == “10”     // true
a === “10”   // false
```

Flow Control

- if, if…else
- ?: (ternary)
- switch
- while, while..do
- for



Flow Control – for..in

```
let myDictonary = { a: "value1", b: "value2", c: "value3" };
for (var key in myDictonary){
      console.log(key + '=' + myDictonary[key]);}
```

```
a = value1
b = value2
c = value3
```

Functions

Optional & Default Parameters

- Optional parameters use ?
- Optional and default parameters must come after required
- Demo

Default Parameters

- Somewhat cleaner than optional parameters
- Assign a default value with the assignment operator (=)
- Demo

Rest Parameters

- Very much like “params”
- Precede with … and follow with [ ]
- Demo

Function Overloading

- Same name, different number of parameters
- Requires creating an *implementation signature*
- Demo

Arrow Functions

- Lambda Expressions
- Short form of functions
- Demo

Object-oriented TypeScript

Object Literals

```
var rectangle = {
   height: 20,
   width: 10,
   area: function () { return this.height * this.width;}
}

console.log("rectangle area = " + rectangle.area());
```

Object Literals (Explicit)

```
var rectangle: Object = {
   height: 20,
   width: 10,
   area: function () { return this.height * this.width;}
}

console.log("rectangle area = " + rectangle.area());
```

Classes

- Public by default
- Refer to properties and fields with *this* keyword
- Constructor parameters can automatically be properties or fields
- Demo

Inheritance

- Uses the keyword *extends*
- Base class is *super*
- Demo

Interfaces

- Uses the keyword *implements*

- Demo

Advanced Topics

- Modules
- Asynchronous programming
- Scale and Performance
- Testing
- Gulp, Grunt, etc.

Resources

- On-line courses
- Books & Articles
- Stack Overflow
- Blogs (http://jesseliberty.com)
- Twitter

Questions?


Thank You
Merci
Gracias
Danke
Grazie