MODULE 02

SETTING UP THE DEVELOPMENT ENVIRONMENT

MODULE TOPICS

Code Editors and IDEs
Package Managers
Web Browsers and Servers, node.js and other engines
Transpiling
TypeScript Configuration Files
Task Runners
Linter
Testing and Debugging

CODE EDITORS AND IDES

Editor / IDE	URL
Visual Studio Code	https://code.visualstudio.com/
Visual Studio	https://www.visualstudio.com/
WebStorm	https://www.jetbrains.com/webstorm/

CODE EDITORS AND IDES

Editor / IDE	URL
Sublime Text	https://www.sublimetext.com/
Atom	https://atom.io/
Brackets	http://brackets.io/

WHAT EDITOR / IDE IS EVERYONE USING?

PACKAGE MANAGERS

- npm (Node Package Manager)
  - Comes with node.js, very popular JavaScript package manager
  - Originally designed for server packages, but is used for all types of packages now
  - Uses a package.json file to store configuration and package details about a project
- Bower - deprecated
  - Designed to handle UI / Front End packages as compared to npm which was for server packages
  - It is not being improved, so they suggest using Yarn and WebPack or Parcel

PACKAGE MANAGERS

- Yarn
  - Popular package manager developed by FaceBook which is small, lightweight and fast
  - Uses the same package.json file as npm
- WebPack
  - WebPack does more than just package management
  - WebPack also provides bundling, minification, and other services
- Parcel
  - Like WebPack, it handles bundling as well as package management

WALKTHRU

Viewing the package.json File

```
{
  "name": "introduction-to-typescript",
  "version": "1.1.0",
  "description": "Introduction to TypeScript Course Files",
  "main": "index.js",
  "scripts": {
    "compile": "tsc",
    "dev": "lite-server",
    "tslint": "tslint",
```

```
npm install
yarn install
```

INSTALLING TYPESCRIPT

Installing TypeScript Globally

```
npm install typescript --global
yarn global add typescript
```

Installing TypeScript Locally

```
npm install typescript --save-dev
yarn add typescript --dev
```

TRANSPILING

Transpiling TypeScript Compile a single file

```
tsc <filename>.ts
```

Compile multiple files

```
tsc <filename1>.ts <filename2>.ts <filename3>.ts
```

Compile to ES6

```
tsc <filename>.ts --target ES6
```

TRANSPILING

TypeScript Compiler Options

```
https://www.typescriptlang.org/docs/handbook/compiler-options.html
```

Compiler Options

Compiler Options

Option	Type	Default	Description
`--allowJs`	`boolean`	`false`	Allow JavaScript files to be compiled.
`--allowSyntheticDefaultImports`	`boolean`	`module === "system"` or `--esModuleInterop` is set and `module` is not `es2015` / `esnext`	Allow default imports from modules with no default export. This does not affect code emit, just typechecking.
`--allowUnreachableCode`	`boolean`	`false`	Do not report errors on unreachable

WALKTHRU

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

let greeter: Greeter = new Greeter("TypeScript!");
console.log(greeter.greet());
```

Transpiling a TypeScript File

```
tsc CompileMe.ts
npm run compile CompileMe.ts
node CompileMe.js
```

TYPESCRIPT CONFIGURATION FILES

- tsconfig.json file
  - Presence of a tsconfig.json file in a folder indicates the containing folder is a TypeScript project
  - Invoking tsc with no input files causes it to search for a tsconfig.json file in current folder and then up the folder tree
  - Specify a path to a tsconfig.json file using the --project or -p compiler switches
  - tsc --init will create a tsconfig.json file with a couple of defaults and potential settings commented out

WALKTHRU

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. */
```

TYPESCRIPT CONFIGURATION FILES

tsconfig.json Configuration Options

```
https://www.typescriptlang.org/docs/handbook/tsconfig-json.html
```

tsconfig.json

Overview

The presence of a `tsconfig.json` file in a directory indicates that the directory is the root of a TypeScript project. The `tsconfig.json` file specifies the root files and the compiler options required to compile the project. A project is compiled in one of the following ways:

Using tsconfig.json

WALKTHRU

TypeScript Compiler Watch

```
https://www.typescriptlang.org/docs/handbook/configuring-watch.html
```

npm

```
PS C:\Users\Geek\Documents\GitHub\EsriDevSummit2019\Classes-Esri2019TypeScript\Student Files> npm run watch

> introduction-to-typescript@1.1.0 watch C:\Users\Geek\Documents\GitHub\EsriDevSummit2019\Classes-Esri2019TypeScript\Student Files
> tsc --watch
```

```
tsc --watch
npm run watch
```

TASK RUNNERS

- Task Runners such as Grunt, Gulp, or Brunch automate tasks that happen routinely during application development.
- Tasks such as linting, compiling, minification, bundling, file manipulation can be automated using these tools
- They support executing tasks manually or automatically (for example on file save)

LINTER

- Code linting analyzes the code and provides feedback on style and best practice issues
- They allow for code to be more consistent and efficient
- Linting also provides valuable information about coding practices as they evolve

WALKTHRU

TSLint

```
npm run tslint CompileMe.ts
```

TESTING AND DEBUGGING

- TypeScript was developed with testing in mind
- Mocha / Chai, Jasmine, Jest are examples of testing frameworks that work with TypeScript
- Extra packages such as ts-node for Mocha, jasmine-ts, and ts-jest are needed

ANY QUESTIONS?