TypeScript Basics

Todd Motto

Setup instructions

**Tools**

* VSCode
* Google Chrome
* Node.JS

**Project Links**

Access the full source code any time, and get started with the Seed Project.

* Seed Project
* Source Code

**Setup Instructions**

Please make sure that you have the following installed:

* Install the latest version of Node.js (Mac or Windows)
* Install TypeScript globally via any of the following options:
  + npm install -global typescript
  + yarn add global typescript

Readme

TypeScript: Basics Course Seed

Project seed for our comprehensive introduction to TypeScript course.

**Setup and install**

Fork this repo from inside GitHub so you can commit directly to your account, or simply download the .zip bundle with the contents inside.

**Dependency installation**

During the time building this project, you'll need development dependencies of which run on Node.js, follow the steps below for setting everything up (if you have some of these already, skip to the next step where appropriate):

1. Download and install Node.js here for Windows or for Mac.
2. Install TypeScript globally via npm i -g typescript

That's about it for tooling you'll need to run the project, let's move onto the project install.

**Project installation and server**

Now you've pulled down the repo and have everything setup, using the terminal you'll need to cd into the directory that you cloned the repo into and run some quick tasks:

cd <typescript-basics-seed>

yarn install

# OR

npm install

This will then setup all the development and production dependencies we need.

Now simply run this to boot up the server:

yarn start

# OR

npm start

Visit localhost:3000 to start building.

**Tasks**

A quick reminder of all tasks available:

Development server

yarn start

# OR

npm start

**Resources**

There are several resources used inside this course, of which you can read further about to dive deeper or understand in more detail what they are:

* [TypeScript Docs](https://www.typescriptlang.org/)
* [TypeScript Playground](https://www.typescriptlang.org/play)
* [AST Explorer](https://astexplorer.net/)

What and Why TypeScript (7:32)

**What is typescript?**

* Open Source language from Microsoft (Created and Maintained)
* Static type system

This has been lived in java script as dynamic types. In past we generally faced bugs like type has been wrongly assigned, passing a wrong argument through a function.

As we code in typescript, it will tell us as and when we type and build the application.

* Typescript is a superset of JavaScript
* Provides optional static typing
* IDE support is a huge win
* More robust software
* Supports new ECMAScript features

Process where gone through various stages and finally become supported in JavaScript.

Decorators, Interfaces, Types does not exist in JavaScript, this is the whole reason why TypeScript exist.

ES5/ES6 for years is a completely valid JavaScript. ES6/ES7 is used to start with for valid typescript.

\*\* More maintainable software, more robust when we start coding.

**Why use typescript?**

* Typed JavaScript, code quality and code understanding
* Types act as documentation

Function signature (types, arguments, optional) etc., by hover over the function, variables and see what exactly does it do.

* Typescript is JavaScript
* Type inference

Simply assign a number to variable and next line assign a string. The typescript will immediately identify it.

* Structural Types – Semantics – We can describe how something looks and attach for function / variable. This will be read by compiler and give proper information if anything is given wrong.
* Adoption can be incremental
* Brand new EMCAScript features (like Async, Away etc.,) .

TypeScript vs JavaScript (6:08)

Actual typescript documentation says: “Typescript is a typed superset of JavaScript that compiles to plain JavaScript”

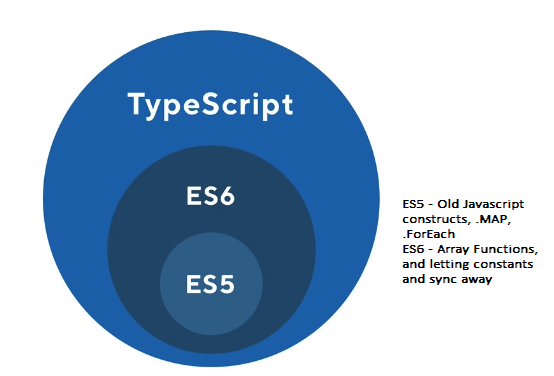
TypeScript vs JavaScript:

* TypeScript is a superset of JavaScript
  + Compiler can compile TypeScript to different versions of JavaScript.
* All ES5/6 features are available in TypeScript.
* New features are commonly added
* Typescript has even more features.
* Support older browser or new browsers.

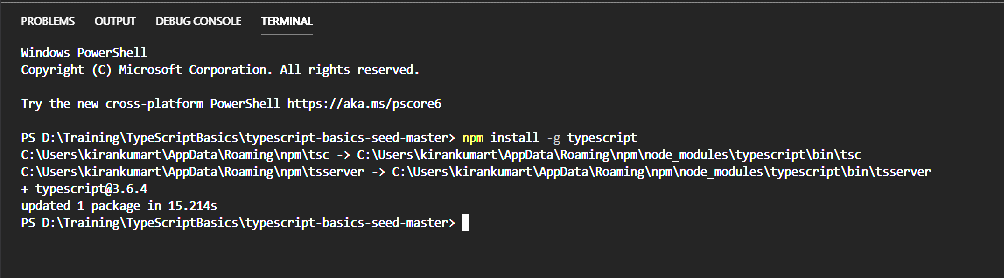
Typescript compiler will convert our typescript code and transform them to JavaScript. This JavaScript code will be served up to the browser or to the node server.

ES6/7 and typescript

* Imports and Exports – not use the module pattern and least concern about order of execution and use functions that is there in other files.
* Arrow functions – this keyword
* Function parameters – optional function parameter
* Spread operators - … typically use with arrays and objects. immutable patterns perhaps for cloning objects or spreading new items into an array.
* Destructuring – way to take properties from objects or ask for values from an array.
* Object literal enhancements
* … and many more



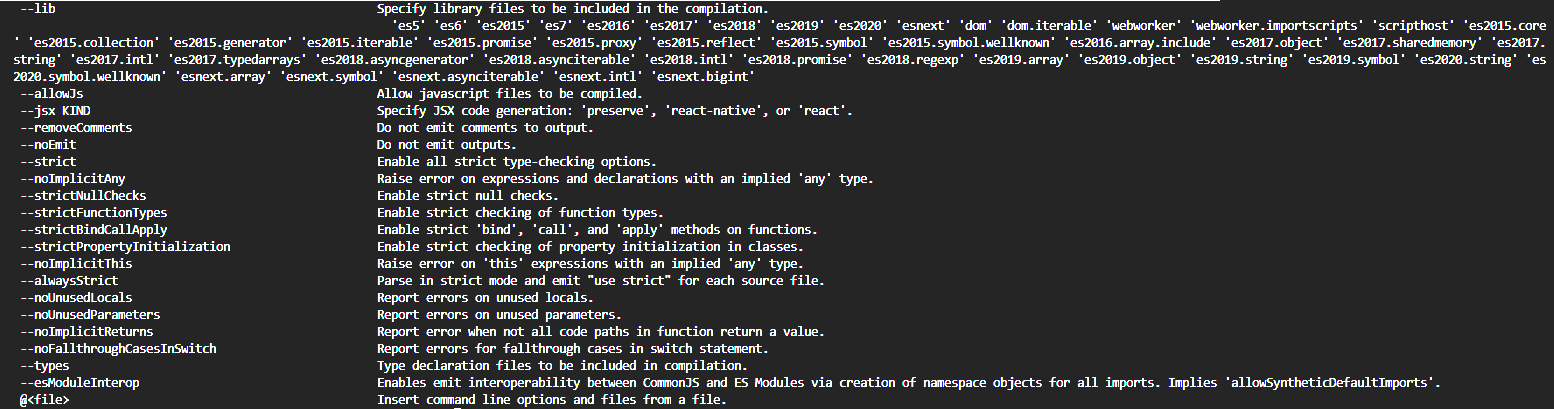
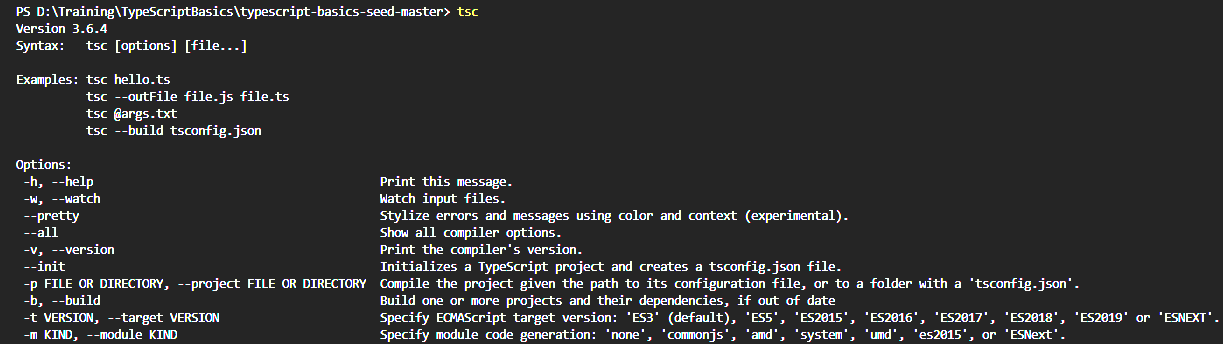
Installing the typescript compiler (05:03)



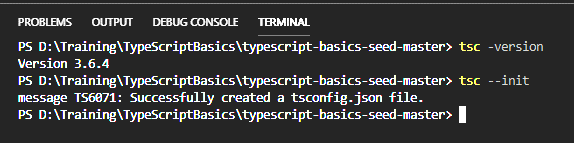
tsc – TypeScript Compiler

tsserver – TypeScript Server

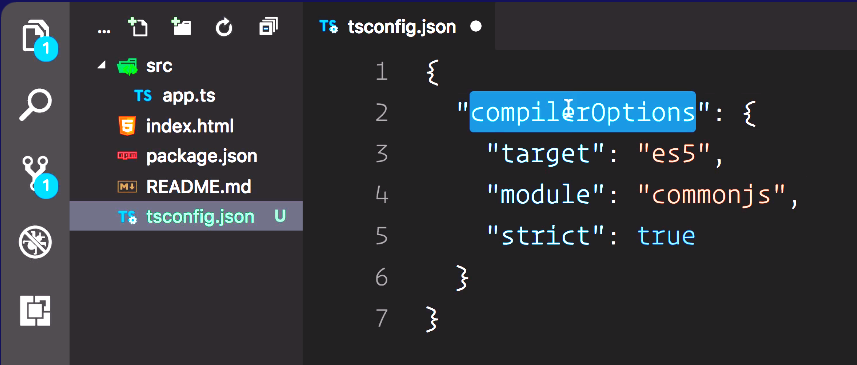
How to setup tsc directly and run app.ts and later we will look at, how we to automatically run with webpack and live reload and automatic recompile of our application.



TypeScript compiler (tsc) and tsconfig (08:11)



Compiler Options in tsconfig.json



Target can contain



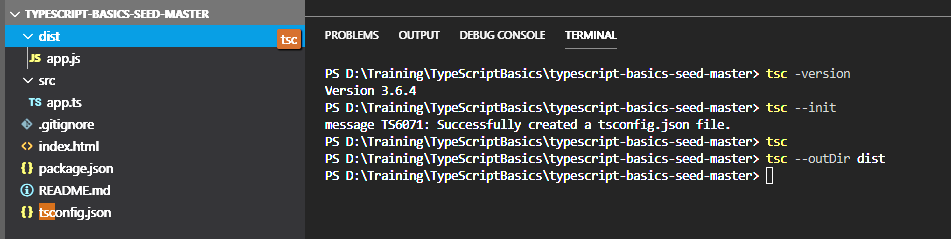
Module can contain



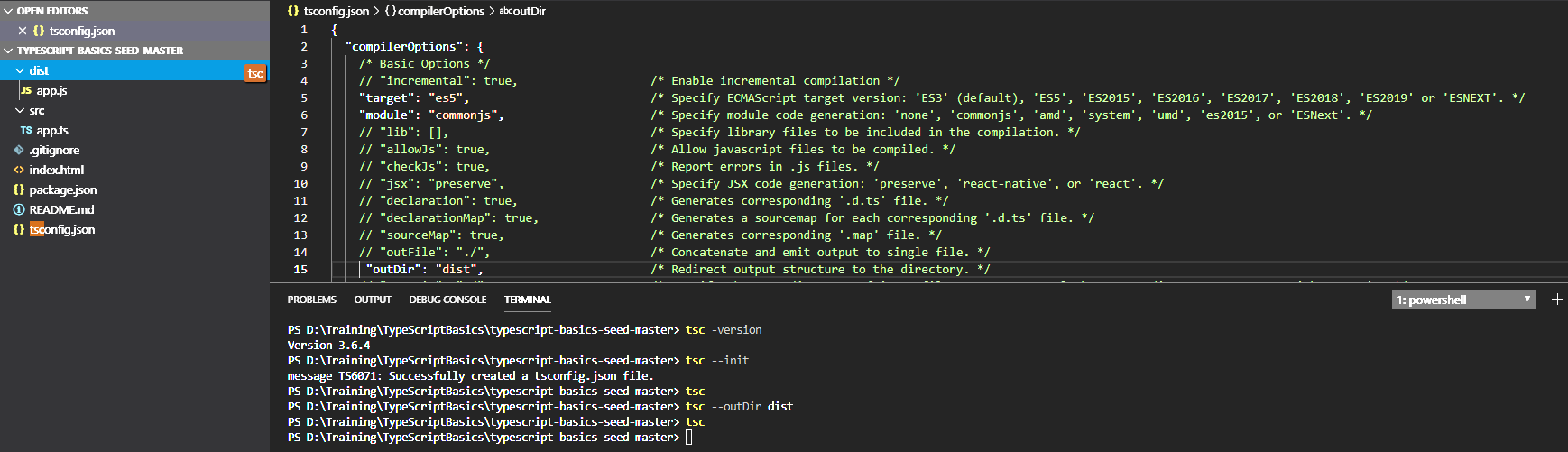
nodeJS – require – commonjs

we also have amd, systemJS, umd (universal module definition)

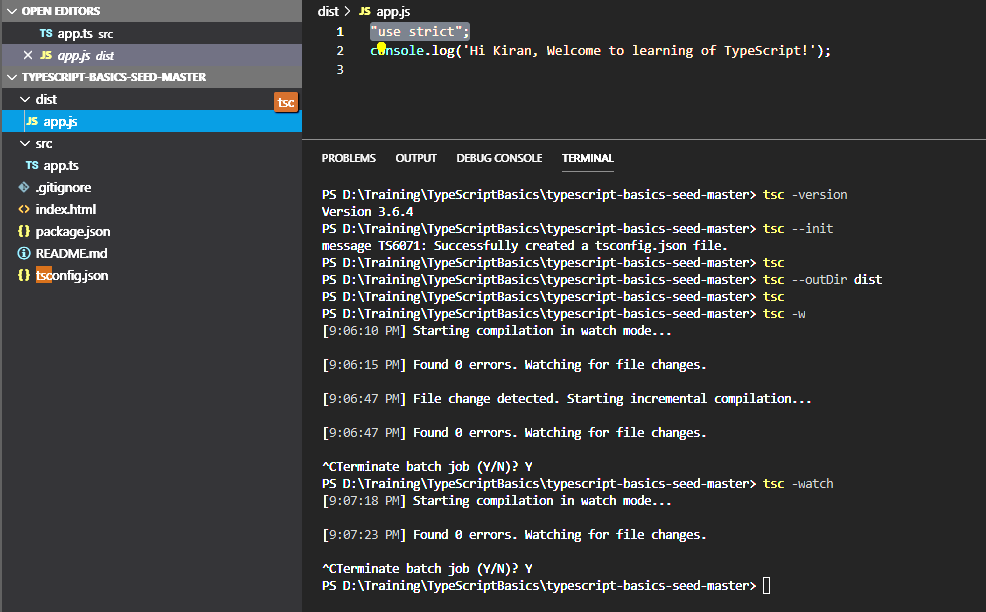




If we have tsconfig.json we can go with running tsc. So, with just typing tsc, we will get app.js created on same folder where app.ts exist. Inside the app.js we will have "use strict";



Talking about outDir, we can specify the output directory structure where JS files will get created and used for running the application.

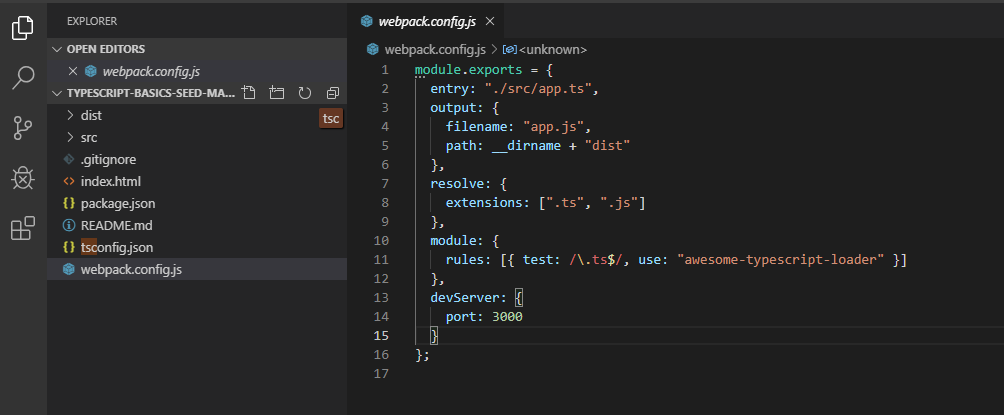


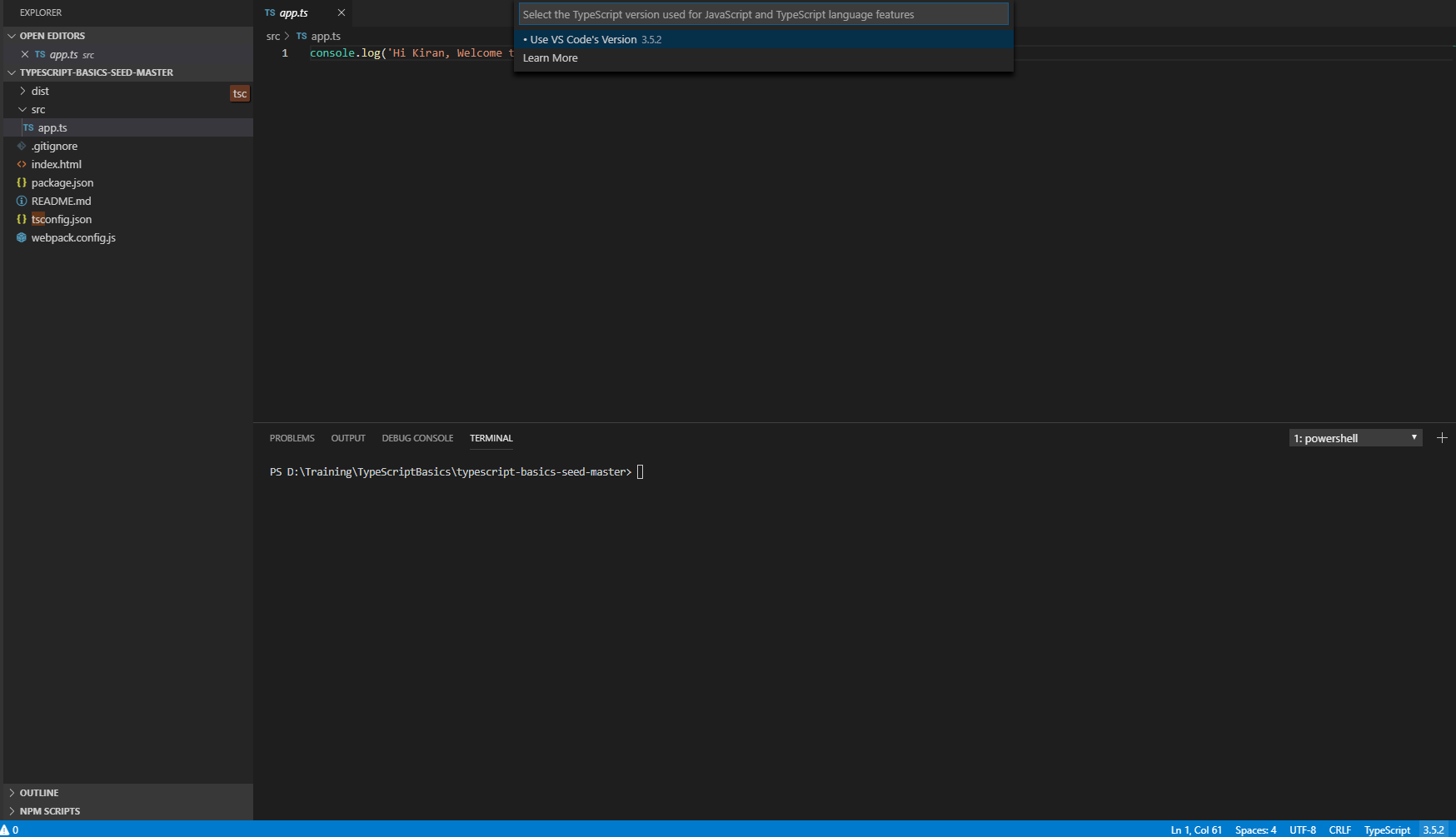
Here we have learned how to ask typescript to automatically update and re-compile on the fly when we do changes to ts file. Local server and live reload application when developing – setup webpack.

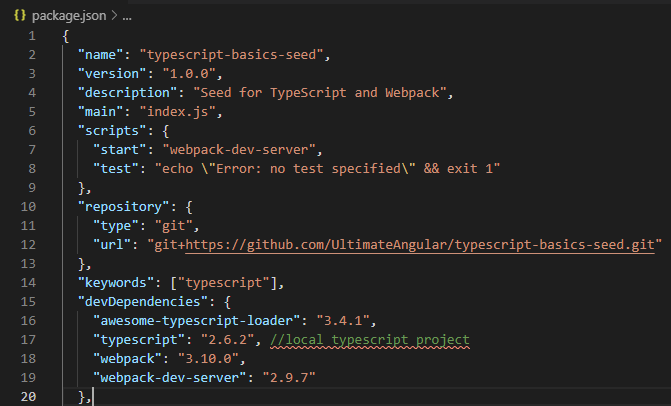
Setting up Webpack for TypeScript (10:40)

Setup basic webpack configuration file, which will just take care of compiling of typescript and the local dev server and keep an eye on the changes.

Create a file webpack.config.js







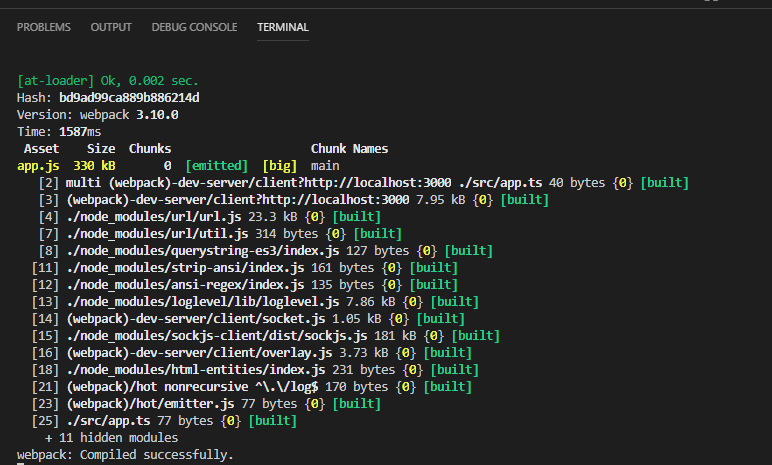
Run below commands in terminal

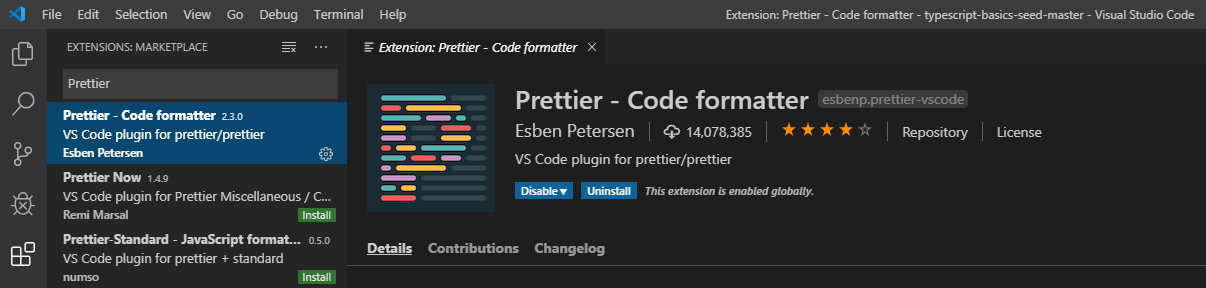
* npm install

In webpack output path is used virtually, so we can remove dist folder and even in index.html we can remove the dist so that bundling will happen properly.

Run below commands in terminal

* npm start





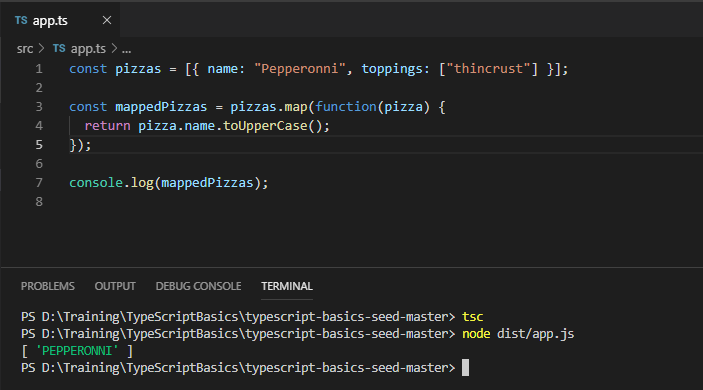
ES6/7 and TypeScript - Arrow functions and implicit returns (08:55)

Let’s discuss about Array function

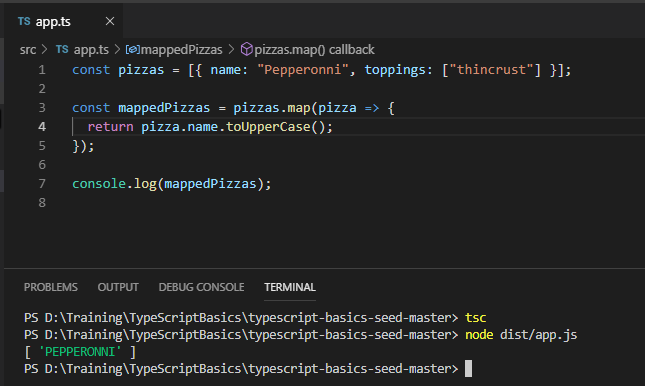
Set up some Data: Data Structure will be name, toppings (array),

Arrow function

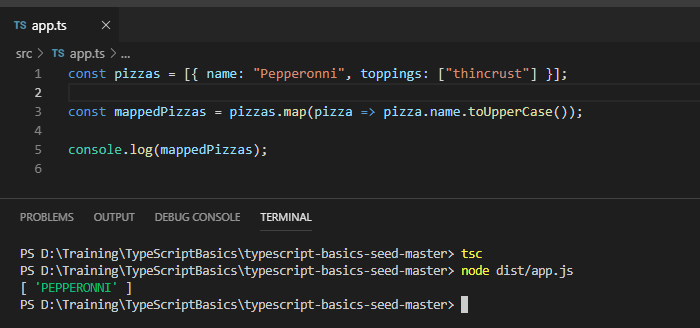
* Arrow function itself, implicit return with array function
* Difference between function and Arrow function

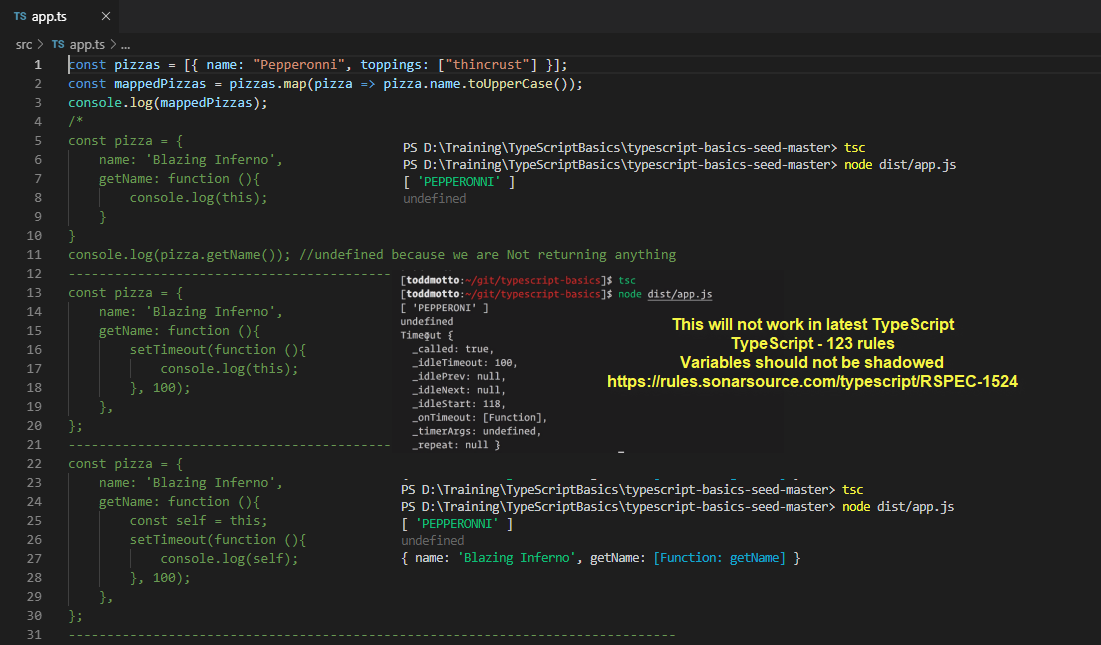


Arrow function (Ignores the scope)

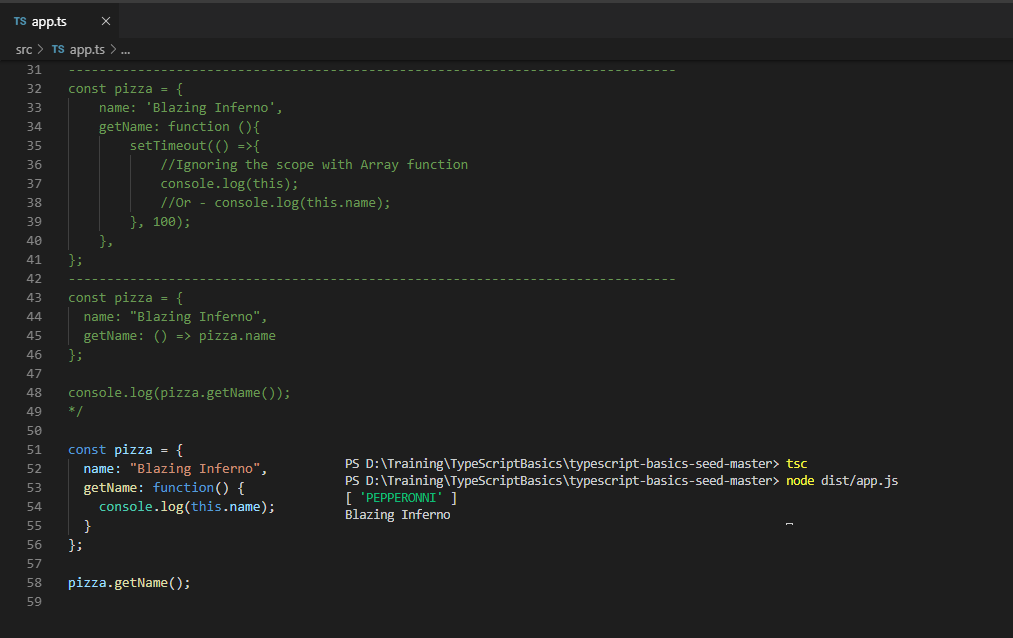


Implicit return

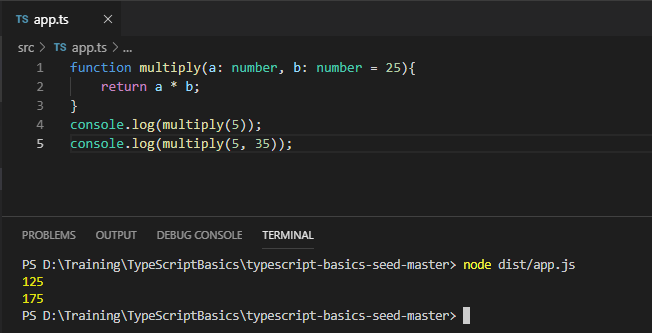




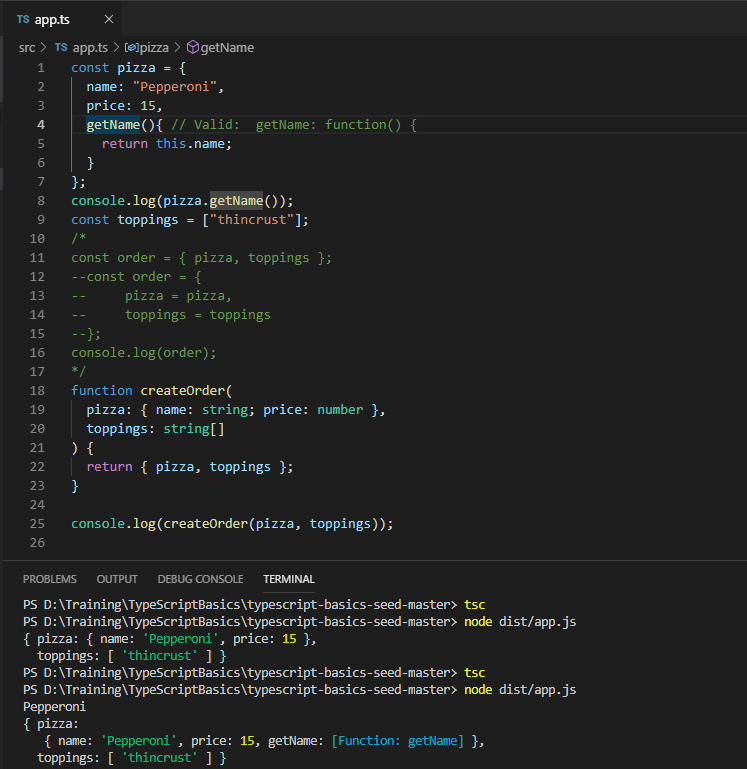
Type script has 136 rules. <https://rules.sonarsource.com/typescript/RSPEC-1524>



Default Function parameters (02:06)

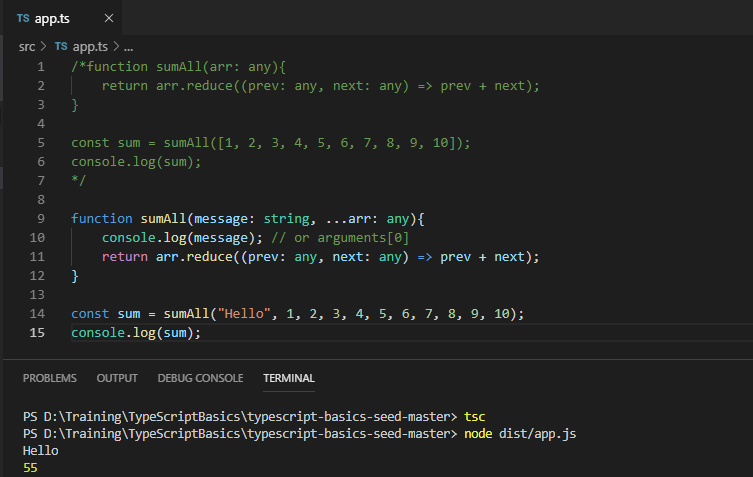


Object literal improvements (04:18)



Rest Parameters (03:17)

Pass any number of arguments into single variable

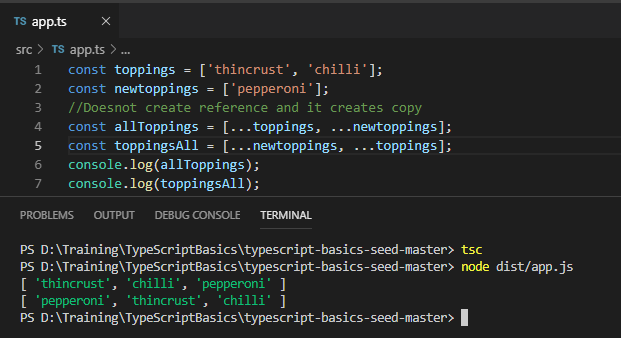


<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce>

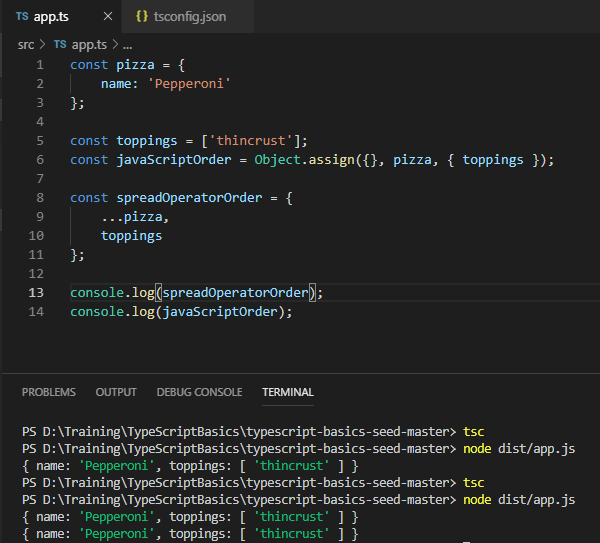
The reduce() method executes the callback once for each assigned value present in the array, taking four arguments:

* accumulator
* currentValue
* currentIndex
* array

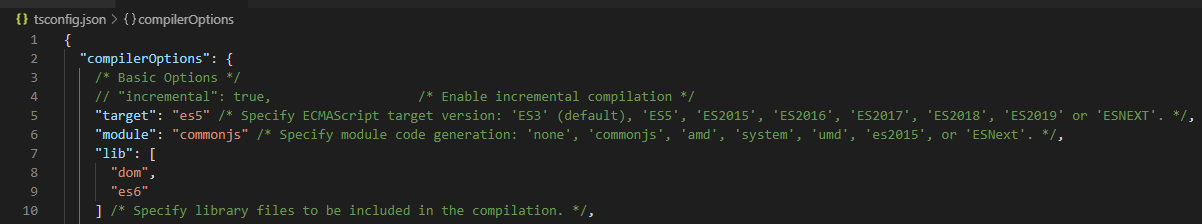
Array spread operator (02:20)



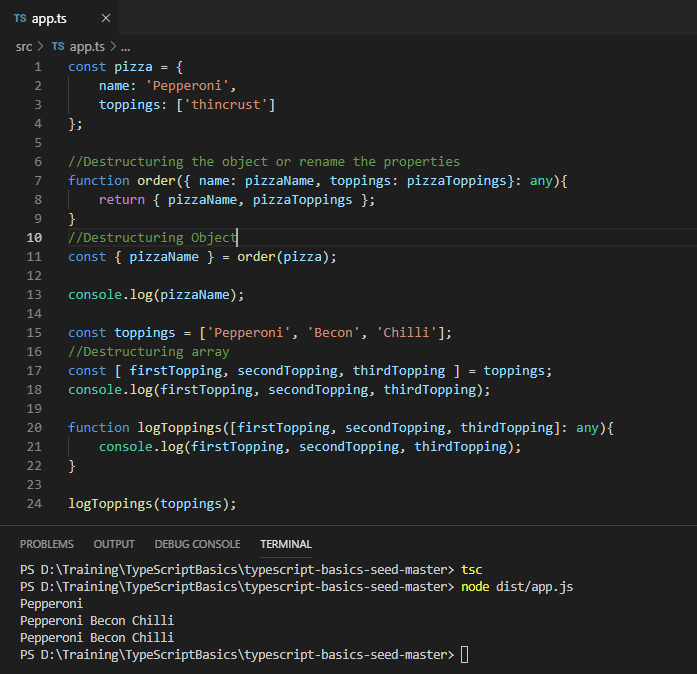
Object spread operator (04:14)



To add Object.assign we have to add lib - compilerOptions entry in tsconfig.json

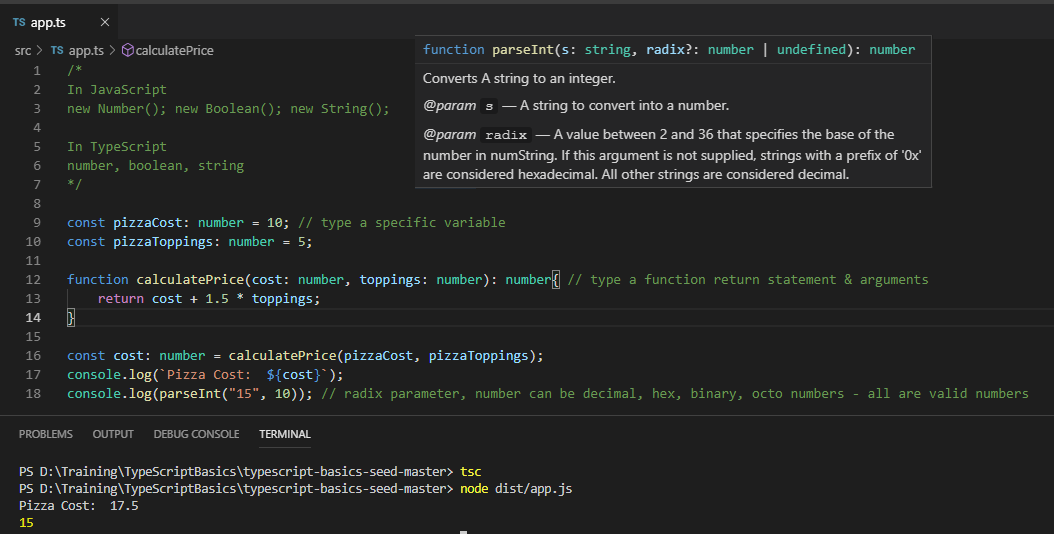


Destructuring Arrays and Objects (06:41)

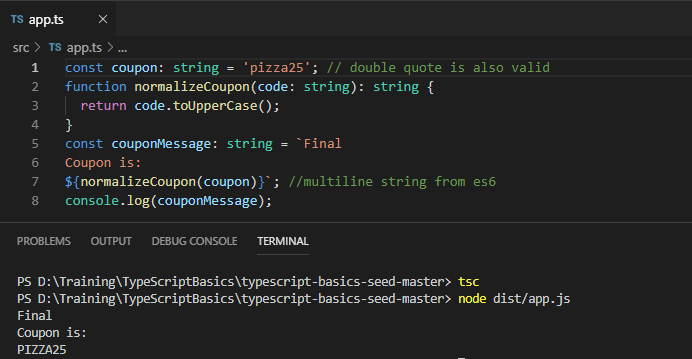


Primitive Types –

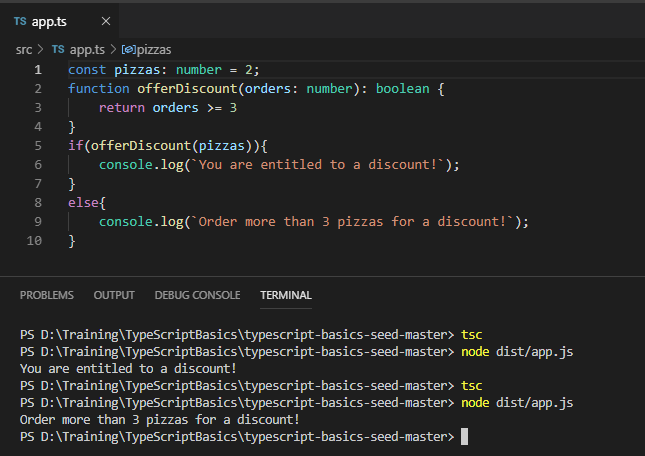
Number Types, arguments and functions (11:21)



String Type, String Literals (05:18)



Boolean Type, Boolean Literals (04:10)



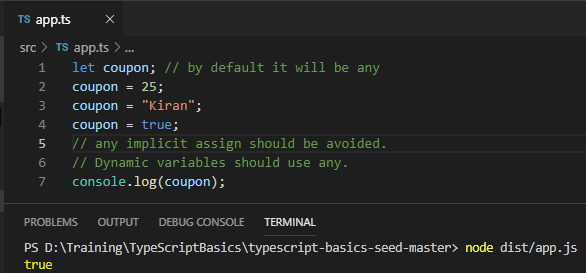
TypeScript Types –

The “ANY” type (03:28)

Essentially, we should try and avoid using ANY type. It should only be used only when what type will be returned.

Cases like using third party library or it could be some kind of unknown response. Typically we know the response type.

let statement allows us to re-assign the value.



Implicit and Explicit types (03:37)

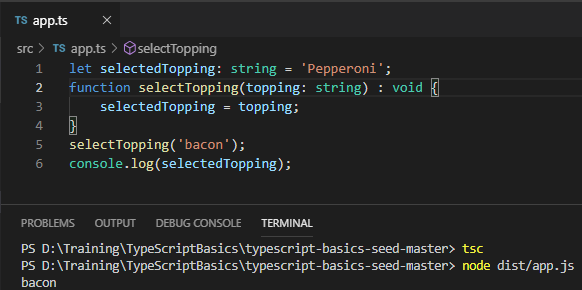
let implicitCoupon = ‘kiran’; // Inferred type

let explicitCoupon: string = ‘kiran’;

Be explicit when Type is implicitly typed to any and Assigning to a parent type

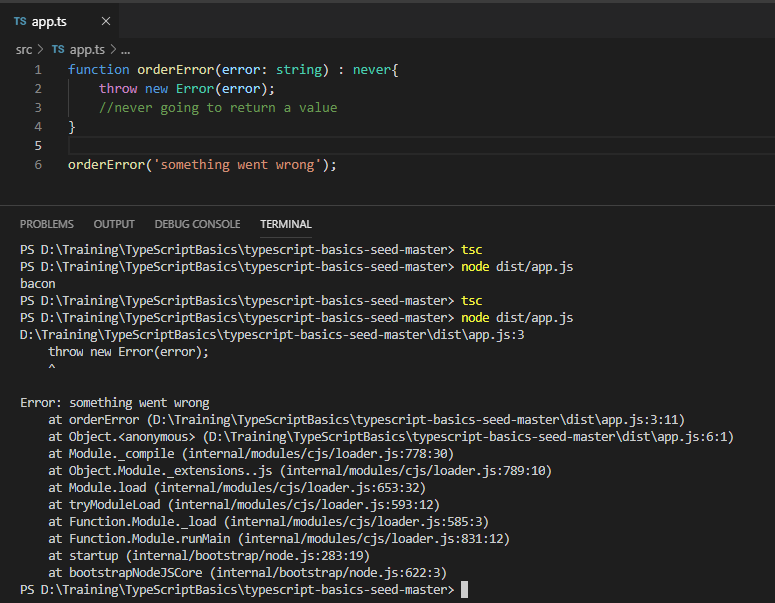
//applies to functions, variables, classes

Void type (03:31)



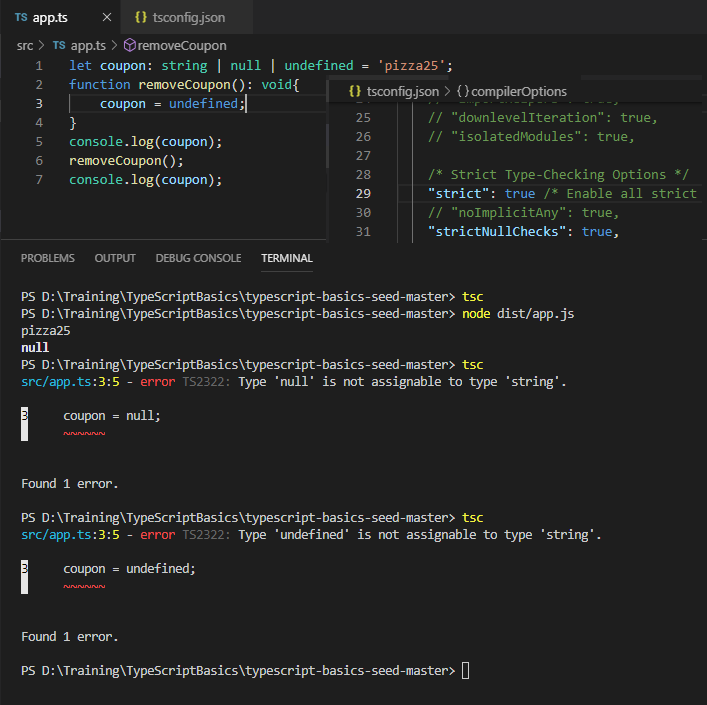
Never type (01:52)

Infrequent types, the value will never occur.



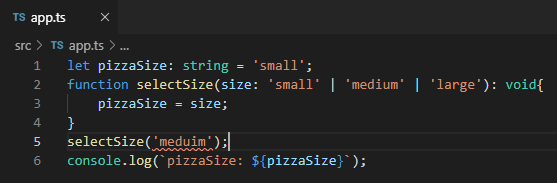
Null, Undefined, Strict Null checks (05:15)

Change strict as false in tsconfig.json

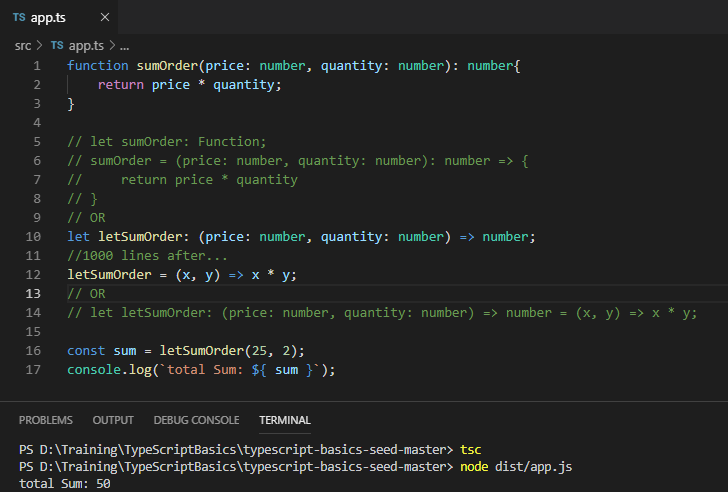


Unions and Literal types (04:05)

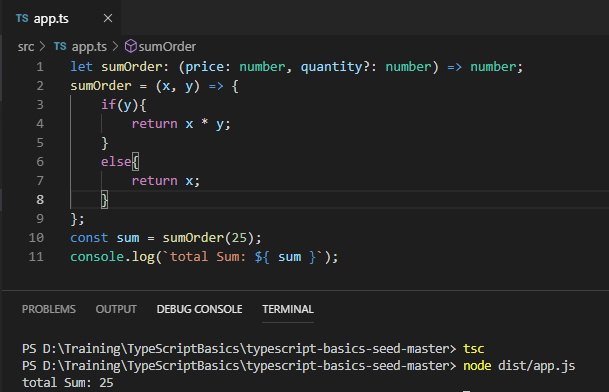
Describing a type to typescript compiler. It will not be found in converted JS file. This will work for string, numbers, boolean and Enum’s as well.



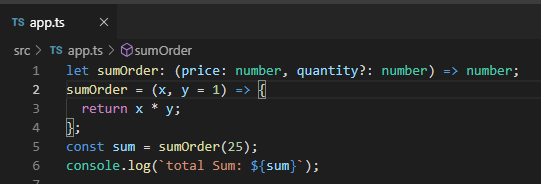
Functional Types (06:19)



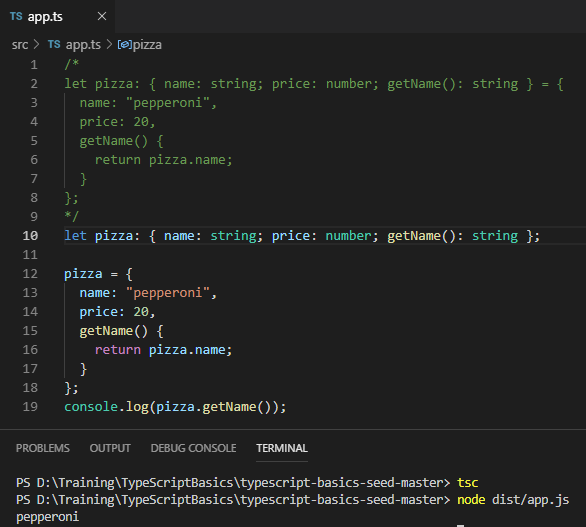
Functions and Optional Arguments (02:28)



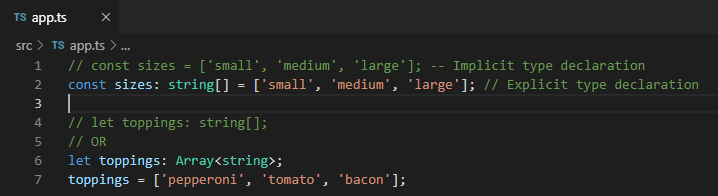
Typed functions and Default Params (02:28)



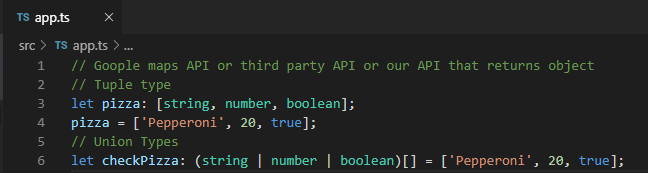
Object Types (03:37)



Array Types and Generics (04:06)

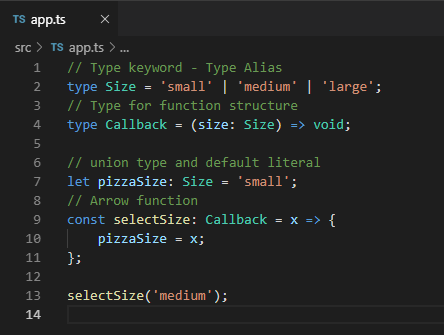


Tuple Types for Arrays (04:28)



Type Aliases and Assertions Type Aliases (05:54)

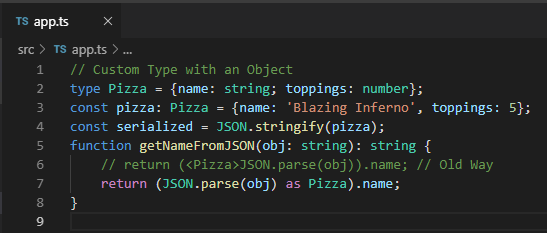
Assign any kind of Type to Type alias.



Type Assertions (07:29)

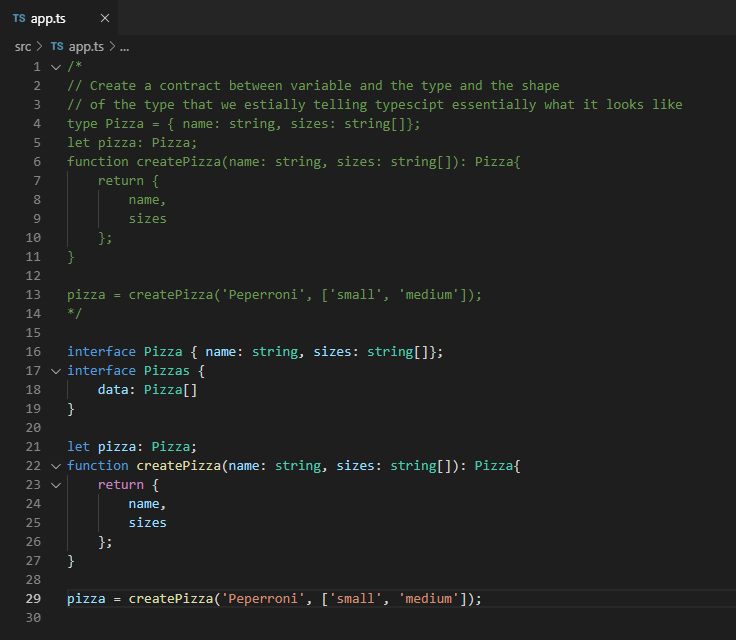
Where we can essential say to typescript that we know more about the types that we are using in our program and typescript will able to pick up on.

We can instruct Typescript what we will be getting back from dataset / callback function.

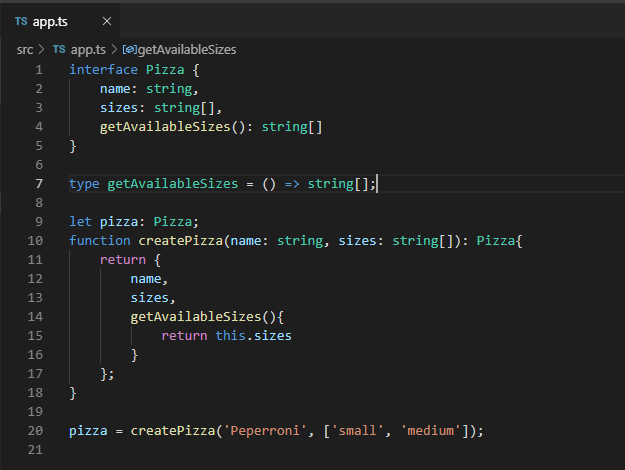


Diving into Interfaces

Creating Interfaces (06:04)

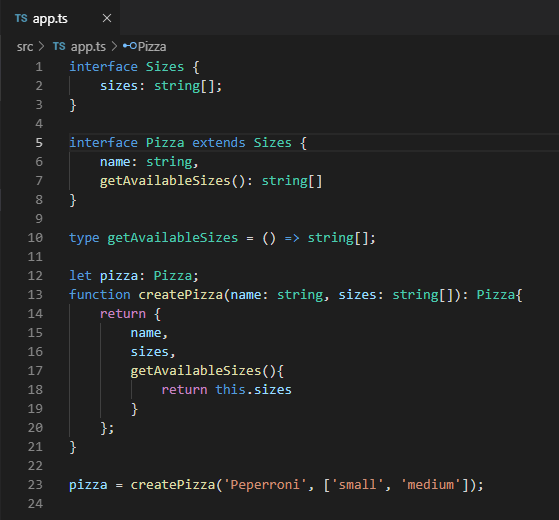


Interfaces with Function Types (03:16)

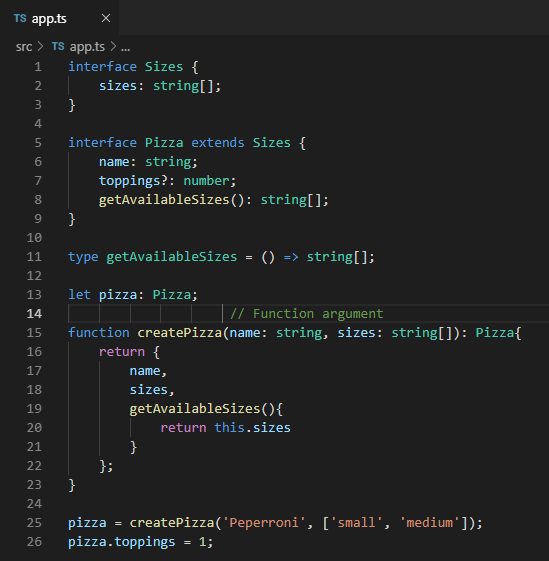


Extending Interfaces (02:23)

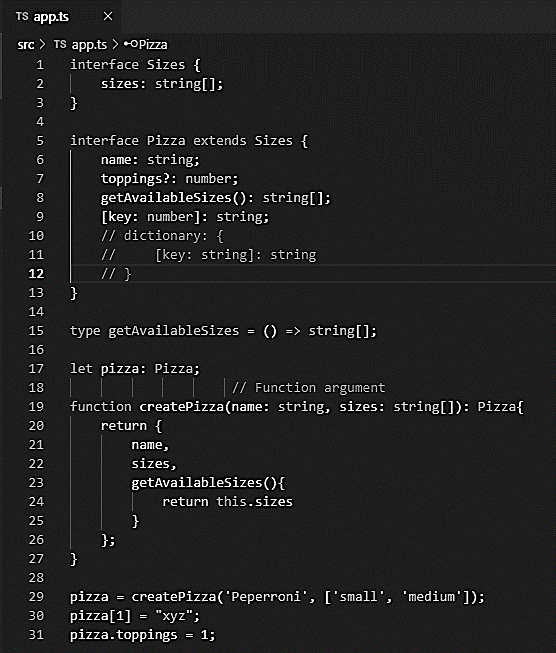
* Extending interfaces
* Implement the interface against the class



Interfaces and Optional Properties (02:00)

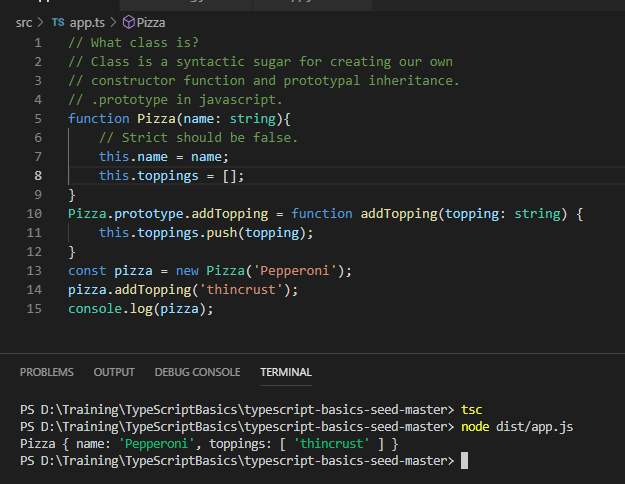


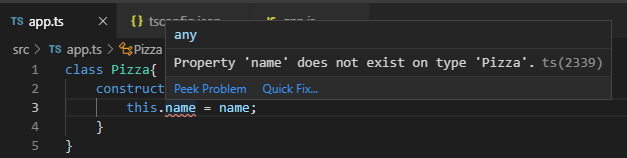
Interfaces with Index Signatures (03:47)

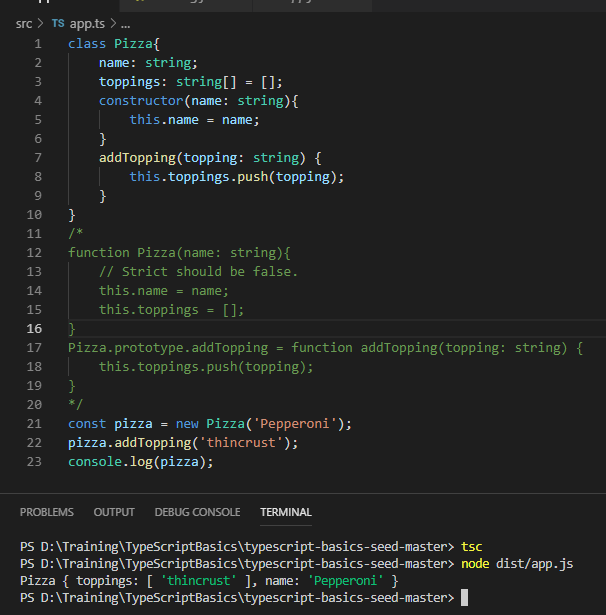


Classes, Properties and Inheritance

Understanding Classes and Constructors (09:16)

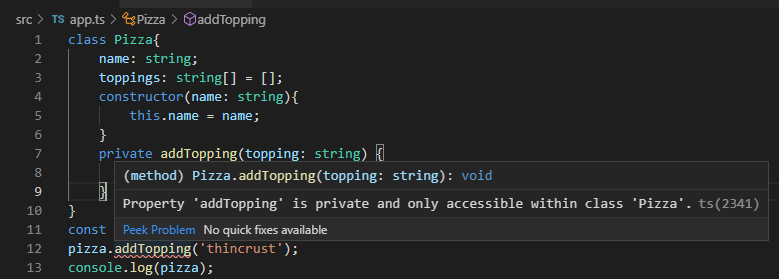


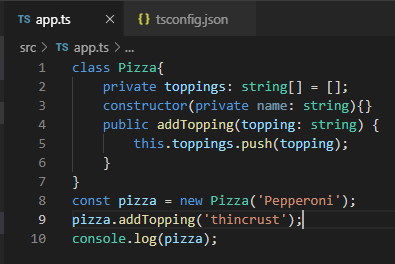




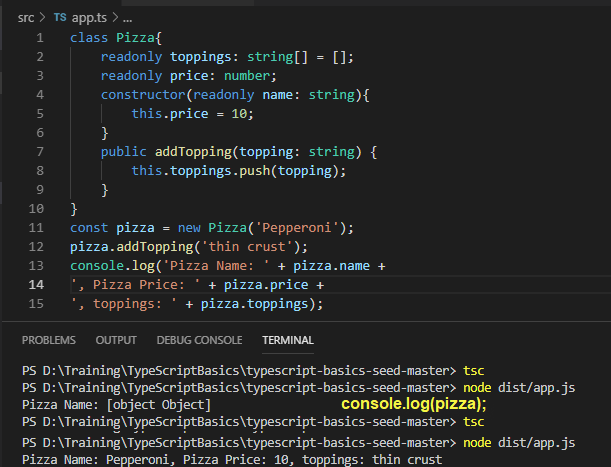
Public and Private Members (04:59)

* Private should be ideally to keep inside internally to a function or a class.
* Public can be access

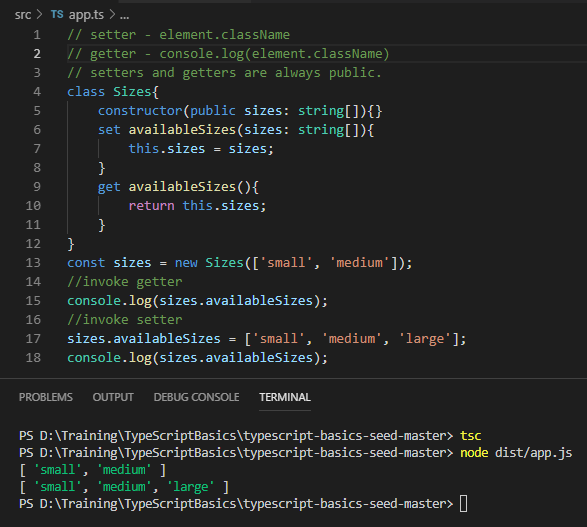


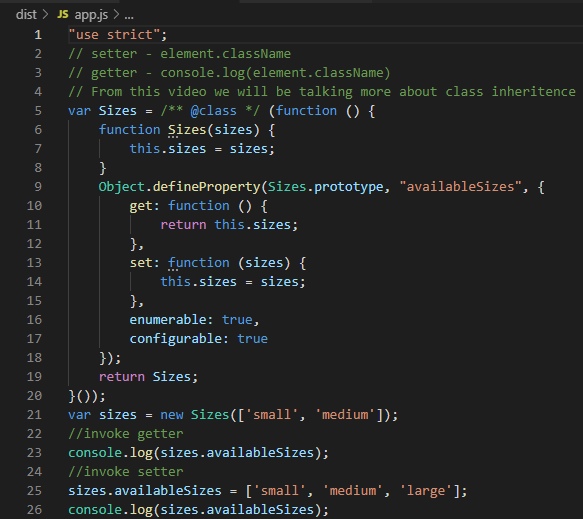


Readonly Members (02:39)

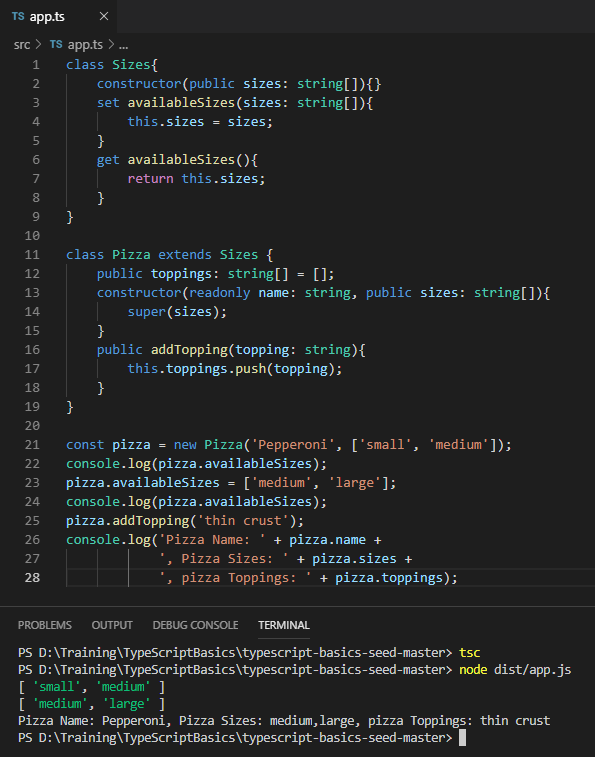


Setters and Getters (Accessors) (06:10)



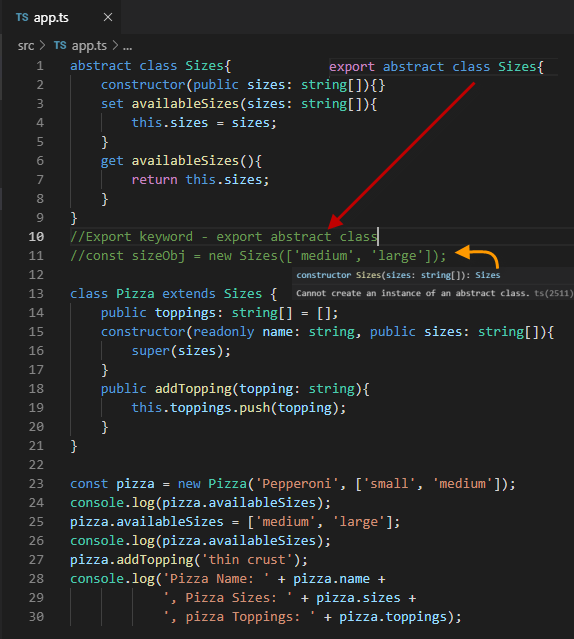


Classes and Inheritance (04:17)



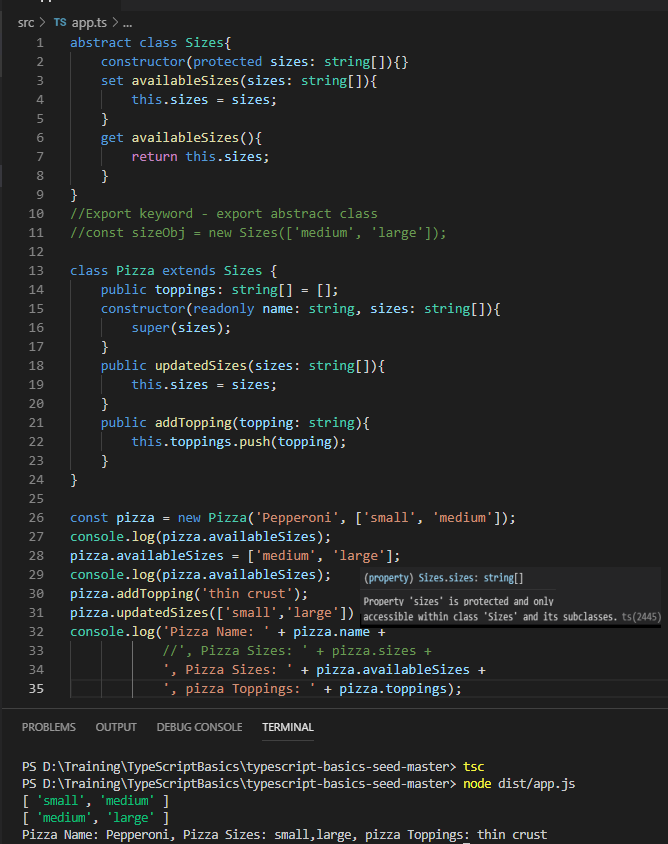


Abstract Classes (01:42)

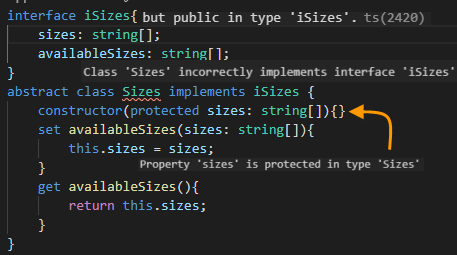


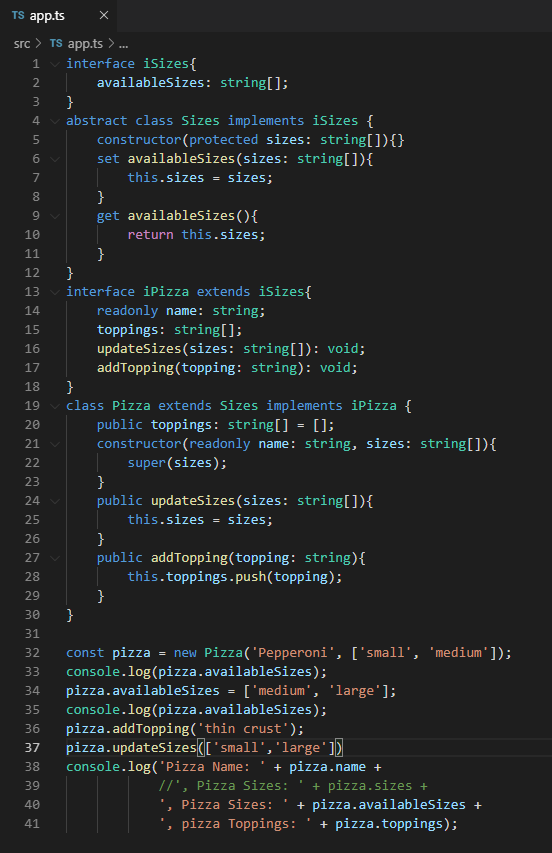
* Cannot create instance of an abstract class
* Abstract class can export, to be used in other classes where it will be imported.

Protected Members and Inheritance (03:38)

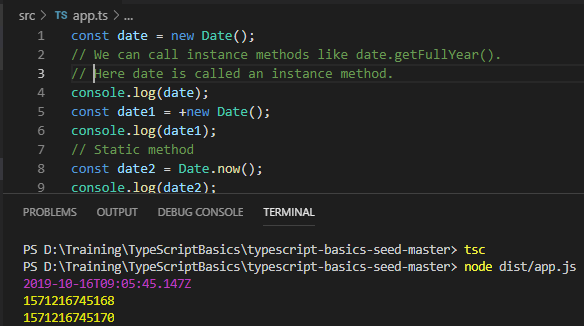


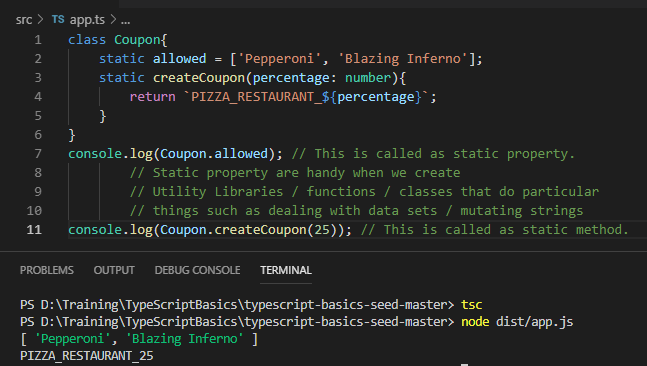
Interface contracts with “implements” (05:39)





Static Properties and Methods (04:57)

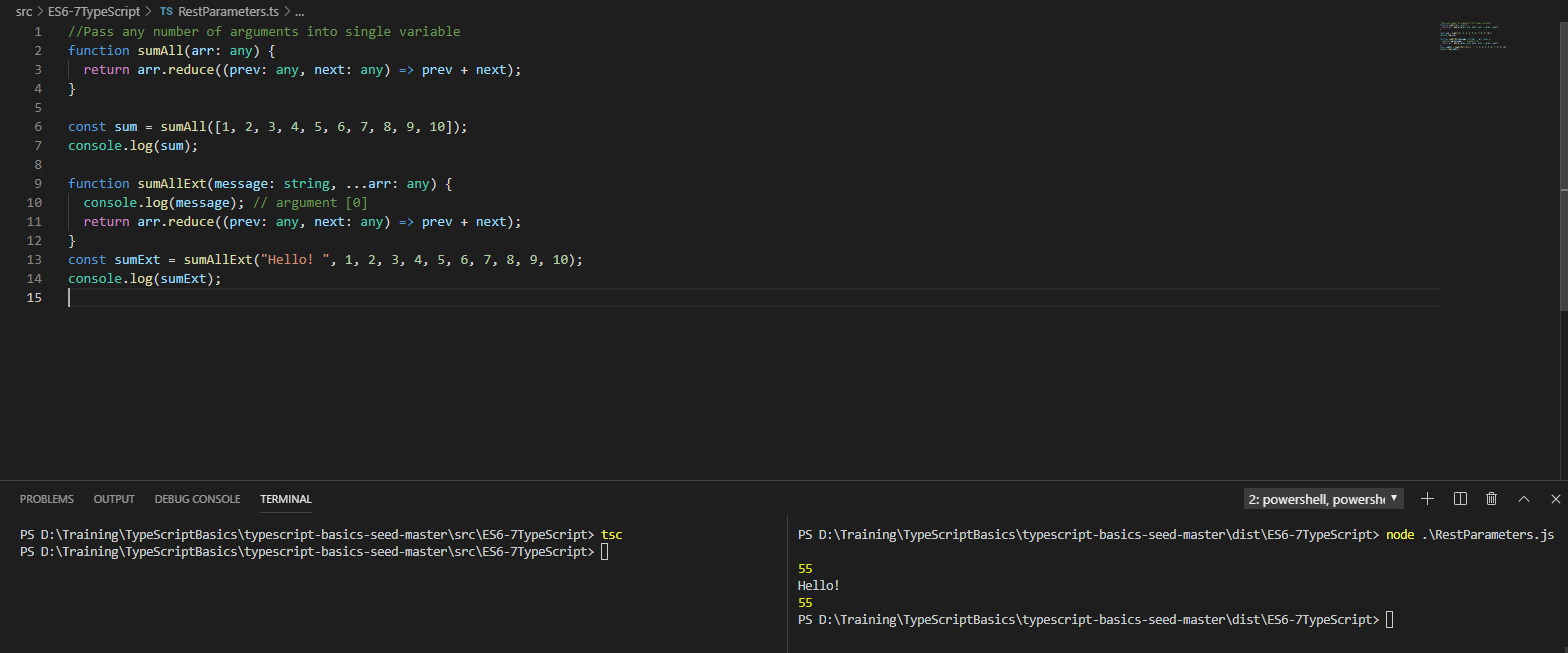




Notes:

1. What is the difference between let, var & const?
2. What are literals?
3. Implicit vs Explicit?
4. Void vs never?
5. Abstract class in TypeScript (analysis)

Sample:



### How numbers in bases work

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Base 10** | **Base 2** | **Base 3** | **Base 4** | **Base 5** | **Base 8** | **Base 16** |
| **1** | 1 | 1 | 1 | 1 | 1 | 1 |
| **2** | 10 | 2 | 2 | 2 | 2 | 2 |
| **3** | 11 | 10 | 3 | 3 | 3 | 3 |
| **4** | 100 | 11 | 10 | 4 | 4 | 4 |
| **5** | 101 | 12 | 11 | 10 | 5 | 5 |
| **6** | 110 | 20 | 12 | 11 | 6 | 6 |
| **7** | 111 | 21 | 13 | 12 | 7 | 7 |
| **8** | 1000 | 22 | 20 | 13 | 10 | 8 |
| **9** | 1001 | 100 | 21 | 14 | 11 | 9 |
| **10** | 1010 | 101 | 22 | 20 | 12 | A |
| **11** | 1011 | 102 | 23 | 21 | 13 | B |
| **12** | 1100 | 110 | 30 | 22 | 14 | C |
| **13** | 1101 | 111 | 31 | 23 | 15 | D |
| **14** | 1110 | 112 | 32 | 24 | 16 | E |
| **15** | 1111 | 120 | 33 | 30 | 17 | F |
| **16** | 10000 | 121 | 100 | 31 | 20 | 10 |
| **17** | 10001 | 122 | 101 | 32 | 21 | 11 |
| **18** | 10010 | 200 | 102 | 33 | 22 | 12 |
| **19** | 10011 | 201 | 103 | 34 | 23 | 13 |
| **20** | 10100 | 202 | 110 | 40 | 24 | 14 |

