Contents

[Quick Labs Environment Set-Up 2](#_Toc143255785)

[Code Editing 2](#_Toc143255786)

[NodeJS 2](#_Toc143255787)

[Quick Labs Alternative Logging Out of Results 2](#_Toc143255788)

[Quick Labs Live Server 2](#_Toc143255789)

[Quick Lab 1a - "Hello World" TypeScript 3](#_Toc143255790)

[Objectives 3](#_Toc143255791)

[Activity 3](#_Toc143255792)

[Quick Lab 1b - TypeScript Dev Environment 4](#_Toc143255793)

[Objectives 4](#_Toc143255794)

[Activity 4](#_Toc143255795)

[Quick Lab 2a - TypeScript Tuples 6](#_Toc143255796)

[Objectives 6](#_Toc143255797)

[Activity 6](#_Toc143255798)

[Quick Lab 2b - Type Assertion and Unknown 7](#_Toc143255799)

[Objectives 7](#_Toc143255800)

[Activity 7](#_Toc143255801)

[Quick Lab 3 - TypeScript Classes 8](#_Toc143255802)

[Objectives 8](#_Toc143255803)

[Activity 8](#_Toc143255804)

[Quick Lab 4 - TypeScript Interfaces 9](#_Toc143255805)

[Objectives 9](#_Toc143255806)

[Activity 9](#_Toc143255807)

[Quick Lab 5 - TypeScript Generics 10](#_Toc143255808)

[Objectives 10](#_Toc143255809)

[Activity 10](#_Toc143255810)

[Quick Lab 6 - Modules 11](#_Toc143255811)

[There is no QuickLab for this module 11](#_Toc143255812)

[Quick Lab 7 - Decorators 12](#_Toc143255813)

[Objectives 12](#_Toc143255814)

[Activity 12](#_Toc143255815)

# Quick Labs Environment Set-Up

## Code Editing

1. Open VSCode (or download and install if not present).
2. Use the desktop shortcut to open the VSCode download page:
3. For Windows users, download the 64-bit System Installer.
4. Check for updates and download and install if necessary:
5. For Windows Users click Help - Check for updates;
6. For MacOS Users, click Code - Check for updates.
7. Using File - Open, navigate to the QuickLabs folder and click Open. This will give you access to all of the QuickLab files and solutions needed to complete the QuickLabs.
8. Icon

   Description automatically generatedUsing CTRL + ' on the keyboard (CTRL + ` on MacOS) or by using click-path View 🡪 Terminal, open VSCode's integrated terminal or click the terminal icon on the bottom bar

## NodeJS

1. Use the desktop shortcut to open the NodeJS download page.
2. Download and install the LTS version for the operating system you’re working in:
3. For Windows users, download the Installer file (.msi);
4. For MacOS users, download the Installer file (.pkg).

## Quick Labs Alternative Logging Out of Results

Webpack has been included in the set-up of the Quick Labs to demonstrate a realistic development environment, so the bundled JavaScript output can be viewed using the browser console.

If you want to manually compile the files you create, you can use the following pattern to see the output:

1. Point the terminal at the TypeScript file location.
2. Run the TypeScript compiler on the file using:

tsc <filename>.ts

1. From the same location, use Node to run the compiled JavaScript file using:

node <filename>.js

The output of any console.logs will appear in the terminal, as will any error messages.

## Quick Labs Live Server

It is also possible to use **Live Server,** which is an extension in VSCode. This spins up a development server, without the need to install and configure Webpack. The tutor will demonstrate how to install, use and set up the file structure to complete this.

# Quick Lab 1a - "Hello World" TypeScript

## Objectives

* To be able to write and compile a simple TypeScript file, and run the outputted JavaScript

## Activity

1. On the command line, using the cd command, navigate to the QuickLabs/01a\_HelloWorld/starter folder.
2. Install TypeScript, globally on your machine using the command:

npm i -g typescript

1. In the starter folder, create a new file called helloWorld.ts.
2. Add the following lines of code into it:

let world = `World`;  
console.log(`Hello ${world}`);

1. Save the file.
2. Back on the command line, compile the TypeScript file to JavaScript using:

tsc helloWorld.ts

Notice that, if the compiler is able to run error free, it appears nothing has happened.

1. Check the starter folder, you should see a new file helloWorld.js - refresh the view if not.
2. Examine the contents of the file helloWorld.js.

Notice how all of the ES2016 syntax has been replaced by older syntax? This is because part of the compilation process deals with the transpilation to ES3!

1. Run the file by typing the following command into the command line:

node helloWorld.js

1. Run the same command but use helloWorld.ts.

The file should run the same - this is because Node has found only JavaScript in the file.

1. Change the declaration of world to the following:

let world: string = `World`;

1. Run the compiler again and check the outputted JavaScript file - you should see that the type declaration has been removed.
2. Run both files with node again - the TypeScript file should now fail as it is not pure JavaScript.

This is the end of Quick Lab 1a

# Quick Lab 1b - TypeScript Dev Environment

## Objectives

* To be able to create a project environment for TypeScript development

## Activity

1. On the command line, using the cd command, navigate to the QuickLabs/01b\_DeveloperEnvironment/starter folder.
2. Initialise an npm project, accepting all defaults by using the command:

npm init -y

1. Install typescript and the webpack plugin ts-loader for the project using:

npm i --save-dev typescript ts-loader

1. Install Webpack, its CLI and the development server using:

npm i --save-dev webpack webpack-cli webpack-dev-server

1. Amend the package.json file scripts section so that the following replace the "test" script:

"build": "webpack --mode production",

"dev": "webpack --mode development",

"start": "webpack serve --open \"Chrome\"",

"check-types": "tsc"

Note on MacOS the start command needs to be:

"start": "webpack serve --open \"Google Chrome\"",

1. Configure webpack by creating the file webpack.config.js and putting the following code inside it:

const path = require('path');

module.exports = {

entry: './src/index.ts',

module: {

rules: [

{

test: /\.tsx?$/,

use: 'ts-loader',

exclude: /node\_modules/,

},

],

},

resolve: {

extensions: ['.tsx', '.ts', '.js'],

},

output: {

filename: 'main.js',

path: path.resolve(\_\_dirname, 'dist'),

},

mode: 'development'

};

The entry part of this file tells Webpack how to get into our application. The resolve object tells Webpack to use both .ts and .js files imported. The module object tells Webpack to use the ts-loader when bundling .ts files. The devtool, set to source map, allows you to set breakpoints within the actual .ts files when debugging.

When the build or dev commands are used, Webpack will create a bundled JS file called main.js (minified for build because of --mode production) and place it in a folder called dist. When using the development server, a virtual file is created and held on it. The check-types script will simply run the compiler and highlight any TypeScript errors.

1. Next, configure TypeScript by creating a tsconfig.json file and putting the following inside it:

{

"compilerOptions": {

"outDir": "./src/",

"noImplicitAny": true,

"module": "es6",

"target": "es5",

"jsx": "react",

"allowJs": true,

"moduleResolution": "node"

}

}

This config enables TypeScript and Webpack to allow you to debug the TypeScript files in the browser (use the Sources tab and then<folder-name>:// 🡪 . 🡪 src and then debug from the .ts file). It also means that the outputted JavaScript is only ES5 compliant. If you need to support ES3 browsers, change this and look into Polyfills!

1. Create the entry file, index.ts in a new folder called src.
2. Re-write or copy the contents from QuickLab 1's TypeScript file in index.ts and save.
3. Create a new file in the starter folder called index.html.
4. Add a skeleton HTML page that as a script tag with a source of main.js.

This file src will need to be modified when going into a production environment. Presently, it enables the development server to run the bundled file.

1. On the command line, run the project using:

npm start

Your browser should spin up now - check the console and also find the file to debug.

1. Experiment with the build and dev commands (insert run after the npm command if it doesn't work out of the box).
2. Run the check-types script and see the results - i.e. the new files in the src folder!

This is the end of Quick Lab 1b

# Quick Lab 2a - TypeScript Tuples

## Objectives

* To experiment with TypeScript's Tuples

## Activity

1. On the command line, navigate to the QuickLabs/02\_TypeScriptTypes/starter folder and initiate the npm install.
2. In VSCode, open the file index.ts from the QuickLabs/02\_TypeScriptTuples/starter/src folder.
3. Define a tuple called person that has 3 types allowed: string, number and boolean.
4. Try to define the person with values in the wrong order - note the errors that are given.
5. Log out the array and verify that the valid JS you’ve written is executed.
6. Define a person correctly and check the logging.
7. Try to add another element using different types to the array and note the errors.
8. Check the output of logging.

This is the end of Quick Lab 2a

# Quick Lab 2b - Type Assertion and Unknown

## Objectives

* To use the unknown and type assertion when working with variables

## Activity

1. In VSCode, open the file index.ts from the QuickLabs/02\_TypeScriptTypes/starter/src folder.
2. Make sure that the development server is still running.
3. It may help you to comment out Part 2a.
4. Define a variable called something with a type of unknown and initially set it to the string 1234.
5. Log out the result of the Boolean expression something == 1234.
6. Log out the result of the Boolean expression something === 1234.
7. Log out the result of the Boolean expression something !== 1234.
8. Log out the result of the Boolean expression something >= 1000.
9. Log out the result of asking for the length of something.

Note the error here - check the output of the compiled JavaScript!

1. Add the as number type assertion to all of the expressions above.

What do you notice about the output of something as number === 1234? Can you explain this?

This is the end of Quick Lab 2b

# Quick Lab 3 - TypeScript Classes

## Objectives

* To experiment with classes and access modifiers

## Activity

1. In VSCode, open the file index.ts from the QuickLabs/03\_TypeScriptClasses/starter/src folder.
2. Run npm install on the command line and start the application running.
3. Declare a class called Vehicle and set it to have private properties of make and model, set as strings and a number private property of speed set initially to 0.
4. Provide get methods for make, model and speed.
5. Provide a set method for speed that:
   * + Takes a parameter delta of type number to represent the change in speed;
     + Checks to see if the new speed (by applying the change speed to the current speed) is greater than 0 - if it is set the new speed to the calculated value, otherwise set speed as 0.
6. Make an instance of a Vehicle and ensure that the methods and modifiers work as expected.
7. Extend the Vehicle with a class called RoadVehicle that has its own private property of wheels (it should be a number).
8. Provide a getter for wheels.
9. Create an instance of a RoadVehicle and check that all of its properties can be accessed in the expected way.
10. Make the Vehicle class and the get and set for speed abstract.

Notice that you are no longer allowed to make an instance of the Vehicle class.

Notice that getters and setters can be abstract - but only as a pair.

Notice that the implementation of the RoadVehicle class is now incorrect.

1. Fix the RoadVehicle class by providing the concrete implementations of the get and set methods for speed. You may notice that \_speed is not accessible - without making it public, what should you do?
2. Check that your new implementation works.

This is the end of Quick Lab 3

# Quick Lab 4 - TypeScript Interfaces

## Objectives

* To use an Interface with multiple classes

## Activity

1. In VSCode, open the file index.ts from the QuickLabs/04\_TypeScriptInterfaces/starter/src folder.
2. Under the provided code, create an interface called HasPassengers, it should specify:
   1. A readonly property of passengerSeats;
   2. A method called makeStop that takes 2 numeric arguments of numberOn and numberOff and specifies that it does not return anything.
3. Create a class called SingleDeckerBus that extends the RoadVehicle class and implements the HasPassengers interface:
   1. The constructor should accept all parameters needed for the RoadVehicle class and the HasPassengers interface, along with a private property passengersOnBoard, initially set to 0.
   2. In the class, implement a getter for passengersOnBoard and the required makeStop method.
      * The makeStop method can be as simple as you like - we implemented it so that the bus never has a negative number of passengers or more passengers than there are seats!
4. Create an instance of SingleDeckerBus and check that the methods work and that the properties are as expected.

If you feel the need…

1. Create a class called Train that extends Vehicle and implements HasPassengers, adding any properties or methods that are needed to make the class function.

This is the end of Quick Lab 4

# Quick Lab 5 - TypeScript Generics

## Objectives

* To experiment with Generic Classes, functions and constraints

## Activity

1. In VSCode, open the file index.ts from the QuickLabs/05\_TypeScriptGenerics/starter/src folder.
2. Examine the supplied code - you should see a Circle class, an abstract Vehicle class, a RoadVehicle interface, a Car class, a Bus class and a Plane class.
3. Under the Plane class, create a Generic Class called Garage:

class Garage<T> {}

1. Give the class a property called garage that is an empty array of type T.
2. Write a park method that takes aThing of type T as an argument and pushes aThing to the garage array.
3. Try to make the method log out the speed of the first item in the garage:

console.log(this.garage[0].speed);

Why is this?

1. Create another Generic Class called VehicleGarage that has a type of T that extends Vehicle.
2. Give the class a property called vehicleGarage that is an empty array of type T and a method called park that takes a vehicle of type T and pushes the vehicle to the vehicleGarage.
3. Try to make the method log out the speed of the first item in the vehicleGarage and then its taxed property.
4. Create a third Generic Class called RoadVehicleGarage that has a type of T that extends RoadVehicle.
5. Give the class a property called roadVehicleGarage that is an empty array of type T and a park method that pushes the roadVehicle supplied to the roadVehicleGarage and then logs out the speed and taxed property value of the first item in the array.
6. Make instances of a Circle, Car, Bus, Plane, Garage, VehicleGarage and RoadVehicleGarage.
7. Try to 'park' each of the objects you have made in each of the garages.

What do you notice?

1. Create a function called logVehicleGarage - it should be Generic, accepting T that extends Vehicle and an argument of anyVehicleGarage that is an array of type T, explicitly returning nothing.

function logVehicleGarage<T extends Vehicle>(

anyVehicleGarage: T[]): void {}

1. Make the body of the function simply log out the garage property of passed arrays.
2. Call the function with the three garages you created earlier.

What do you notice?

This is the end of Quick Lab 5

# Quick Lab 6 - Modules

## There is no QuickLab for this module

This is the end of Quick Lab 6

# Quick Lab 7 - Decorators

## Objectives

* To experiment with a simple set of decorators

## Activity

1. In VSCode, open the file index.ts from the QuickLabs/07\_Decorators/starter/src folder.
2. Write a function called simpleDecorator that takes an argument target of type any.
3. Make the body of the function log out a simple message.
4. Create a class called DecoratedClass and decorate it with the simpleDecorator.

@simpleDecorator

class DecoratedClass {}

1. Save the file and view the result on the console.

The compilation may fail at this point as decorators are still an experimental feature. The message suggests to add experimental decorators option to the tsconfig.json file. Here's how:

1. Open tsconfig.json for the project.
2. At the end of the line for target, add a comma.
3. Add a new key of experimentalDecorators with a value of true.

Note: for compatibility, the **tsconfig.json** is different to the file used in other Labs

1. Save the file and recompile - you may need to restart the process.

This should clear the error.

1. Write a function called factoryDecorator that takes name of type string as an argument.
2. Make the body of the function return a function that takes target of type Function as an argument and logs out name and ' decorator was called'.
3. Create a class called FactoryDecoratedClass, decorating it with the decoratorFactory, passing in the string "factory".
4. Save the file and view the output on the console.
5. Create a function called merge which takes an argument toMerge of type Object.
6. Make the body of the function return a function that:
   1. Takes an argument target of type any
   2. Uses a for in loop to examine each prop in toMerge setting target.prototype[prop] to toMerge[prop].
7. Declare a variable called user with name (string), age (number) and instructor (boolean) key/value pairs.
8. Create a class, decorated with merge and user as a parameter, called AnotherDecoratedClass.
9. Instantiate the class calling it thing.
10. Log out the name property on thing, asserting that thing is of type any.
11. Save the file and examine the output.
12. Write a function called readOnly that has arguments of:
    1. target of type any;
    2. methodName of type string;
    3. descriptor, optional of type PropertyDescriptor
13. Make the body of the function set the writable and enumerable properties of descriptor to be false.
14. Create a class call HoldingClass.
15. In HoldingClass add a function called sayHello that is decorated with readOnly and logs out the message 'Hello'.
16. Instantiate HoldingClass calling it newThing.
17. Call sayHello on newThing.
18. Try to change newThing.sayHello to a value of false.
19. Save the file and observe the output.

This is the end of Quick Lab 7 and of all QuickLabs for TypeScript