

# Advanced TypeScript

## Goal

In this lab, you will:

- Bundle Typescript code using Webpack
- Improve the code quality with compiler checks
- Unit-test the code with Mocha

## Your mission

In this lab you will use Webpack to compile and bundle your Typescript code. You will import jQuery to wire up the DOM event handlers. Next you will use the Typescript compiler options to improve the code quality. Finally, you will use Mocha to unit-test your code.

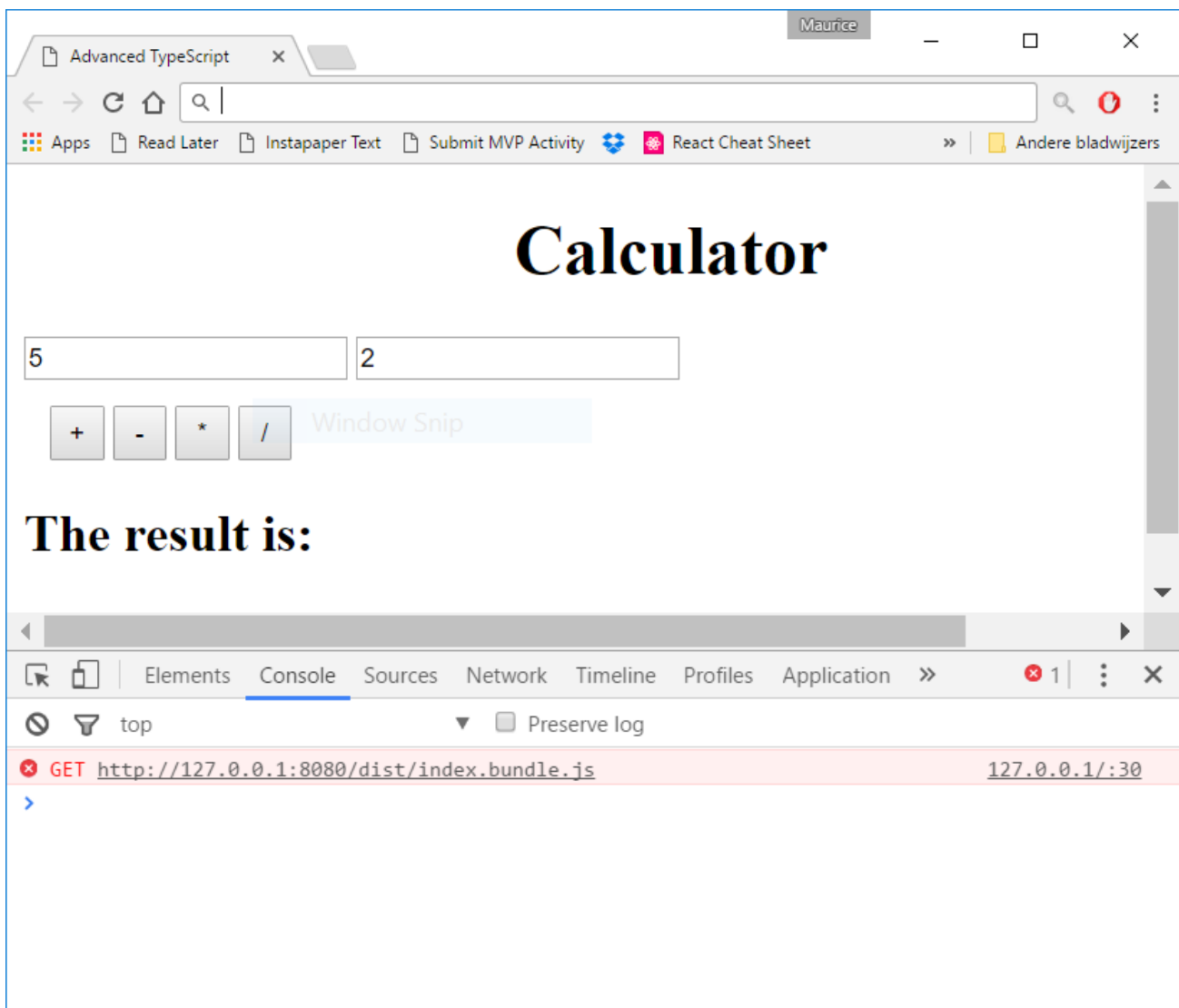
## Type it out by hand?

Typing it drills it into your brain much better than simply copying and pasting it. You're forming new neuron pathways. Those pathways are going to help you in the future. Help them out now.

## Bundle the code

In this section you will compile and bundle the code with Webpack.

1. Open the **begin** folder and observe that there is an **index.html** that loads the **./dist/index.bundle.js**. This file doesn't exist yet but will be created from the files in the **src** folder.
2. Start the server by running **npm start**. Open the **index.html** page in the browser by navigating to <http://127.0.0.1:8080/>.
3. Open the **index.ts** in the editor of your choice. Notice it imports the **Calculator** class and uses it to calculate **2 \* 5** and print the result to the console. Currently the **index.bundle.js** doesn't exist yet and opening Chrome the developer tools shows this error message.

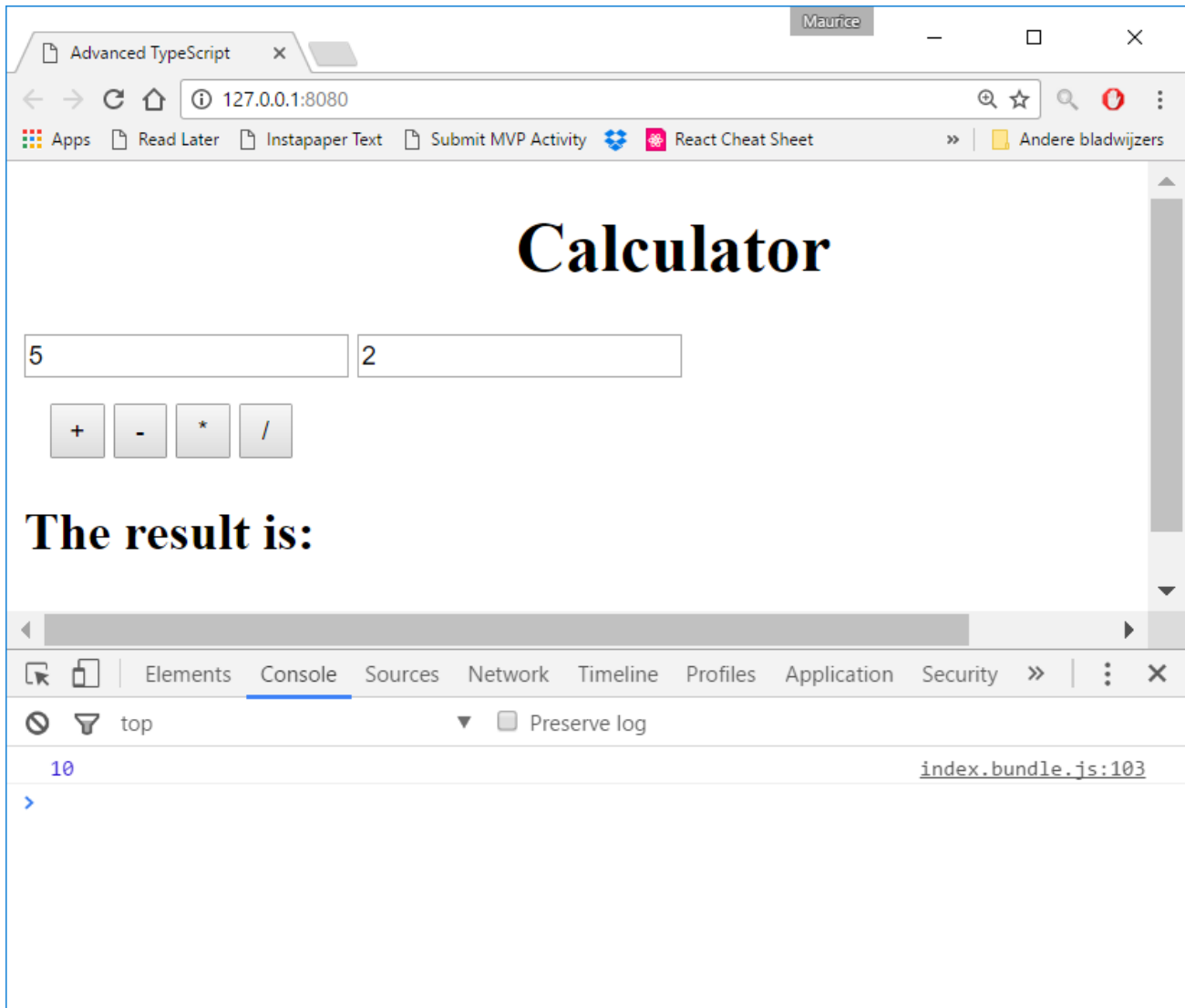


4. Install webpack using `npm install webpack --save-dev`.
5. Open `webpack.config.js` and add `./src/index.ts` as the entry point. Use `./dist/index.bundle.js` as the output and add both `.js` and `.ts` as the resolved file extensions.

```
const path = require('path');

module.exports = {
  entry: './src/index.ts',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'index.bundle.js'
  },
  resolve: {
    extensions: ['.ts', '.js']
  }
};
```

6. Build the bundle using `npm run webpack`. Refresh the browser and you should see the result of multiplying  $2 * 5$  in the console.



## Compile the TypeScript

In this section you will use the **ts-loader** to compile the TypeScript to ECMAScript.

1. Open **calculator.ts** and add type information to the parameters of the multiply function.

```
multiply(x: number, y: number) {  
  return x * y;  
}
```

2. Start Webpack again. Notice it complains about unexpected tokens in **calculator.ts**. This is because Webpack doesn't compile the Typescript yet but treats it as ECMAScript. Without the type annotations the previous code was valid ECMAScript 2015 which Chrome understands.

3. Install **ts-loader** and **typescript** using the command `npm install ts-loader typescript --save-dev`.
4. Open **webpack.config.js** and add the **ts-loader** module.

```
module: {  
  rules: [{  
    test: /\.ts$/,  
    use: 'ts-loader'  
  }]  
},
```

5. Run Webpack again and notice that there are no errors anymore and the bundle is output again.

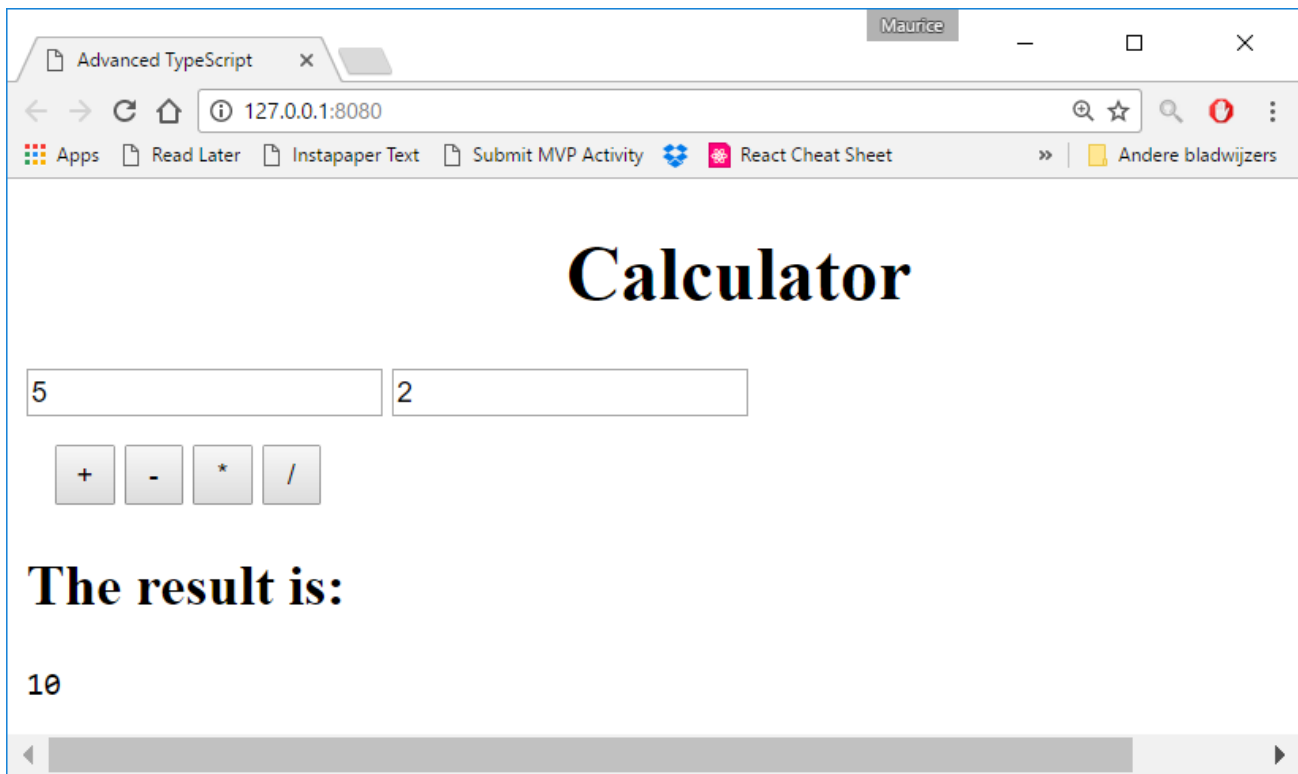
## Include jQuery and wire up the DOM event handlers

In this section you will add jQuery and wire up the DOM event handlers

1. Install **jQuery** using the command `npm install jquery --save`.
2. Install the jQuery type definitions using the command `npm install @types/jquery --save`.
3. Open **wireup.ts** and uncomment all code.
4. Open **index.ts** and uncomment the two lines of code that import and call **wireup**.
5. Run Webpack again. Note that it runs without errors and includes **./src/wireup.ts** now.
6. Refresh the browser and notice the error **\$ is not defined**. This is because jQuery is not include in the bundle yet. Because jQuery is an older library importing it doesn't work and you need to use the Webpack **ProvidePlugin** instead.
7. Open **webpack.config.js** and add the **ProvidePlugin** plugin.

```
plugins: [  
  new webpack.ProvidePlugin({  
    $: 'jquery'  
  })  
]
```

8. Refresh the browser and make sure the buttons and input elements are functional. Note that the add button produces the incorrect result. You will fix this in a later section.



## Improve the code quality with compiler checks

In this section you will add better compile time checking.

1. Open **tsconfig.json** and add the compiler options to prevent implicit any types.

```
{
  "compilerOptions": {
    "noImplicitAny": true
  }
}
```

2. Run Webpack again. Notice there are a number of compilation errors like **error TS7006: Parameter 'x' implicitly has an 'any' type.** from **calculator.ts**.
3. Open **calculator.ts** and fix the errors by adding type information for all functions.
4. Run Webpack again. This time there should be no errors.
5. Note that there are no Typescript errors as the result of passing in the values as strings instead of numbers. This is the result of the jQuery **val()** function being typed as **any**. Fix the error by converting the parameters to numbers.

```
add(x: number, y: number) {
  return (+x) + (+y);
}
```

```
}
```

## Unit-test the code with Mocha

In this section you will add unit tests for the **Calculator** class.

1. Install mocha, chai and ts-node as development dependencies.
2. install type definitions for mocha and chai as development dependencies.
3. Create a new file **calculator-tests.ts** in a new **tests** folder.
4. Import mocha and the expect function from chai.
5. Add the **test** script to execute mocha and use **ts-node/register** as the compiler for files with the **.ts** extension.

```
"test": "mocha --compilers ts:ts-node/register **/*-tests.ts"
```

6. Write tests for the different functions on the Calculator class. Use the values **-1, 1, 2** and **0.1, 0.2** as the test values for all functions. Below is an example for the **add** function.

```
import 'mocha';
import {
  expect
} from 'chai';

import Calculator from '../src/calculator';

describe('The calculator', () => {
  let calculator: Calculator;

  before(() => {
    calculator = new Calculator();
  });

  describe('can add', () => {
    it('-1 + 1 === 0', () => {
      const result = calculator.add(-1, 1);
      expect(result).to.equal(0);
    });

    it('1 + 2 === 3', () => {
      const result = calculator.add(1, 2);
```

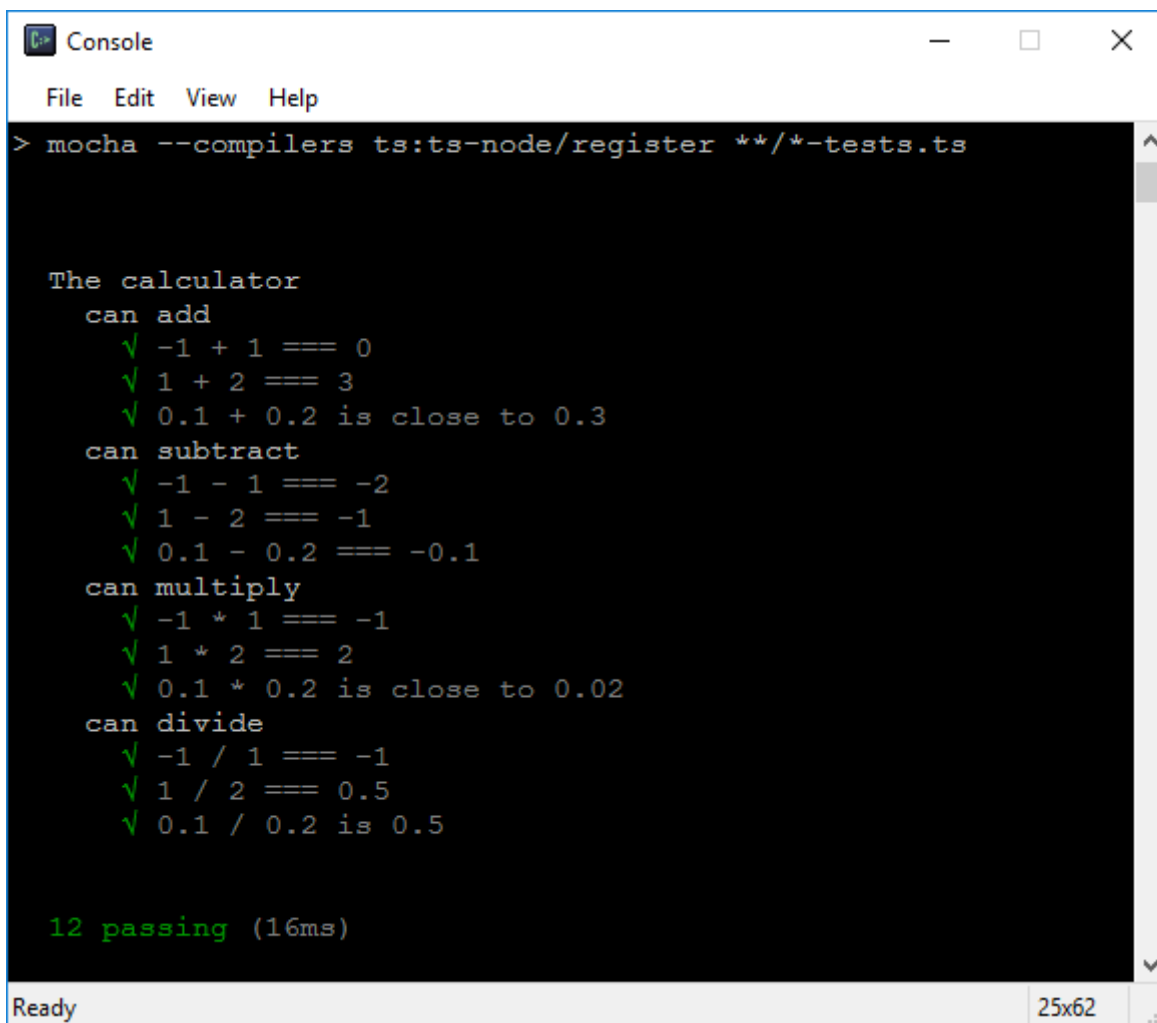
```

        expect(result).to.equal(3);
    });

    it('0.1 + 0.2 is close to 0.3', () => {
        const result = calculator.add(0.1, 0.2);
        expect(result).to.closeTo(0.3, 10);
    });
});
});

```

7. Run the tests using the command `npm test`. The result should look like this:



The screenshot shows a VS Code console window titled 'Console' with a menu bar (File, Edit, View, Help). The command prompt shows the command: `> mocha --compilers ts:ts-node/register **/*-tests.ts`. The output displays the results of 12 passing tests for a calculator, grouped by functionality: 'can add', 'can subtract', 'can multiply', and 'can divide'. Each test is marked with a green checkmark. The total result is '12 passing (16ms)'. The status bar at the bottom indicates 'Ready' and '25x62'.

```

> mocha --compilers ts:ts-node/register **/*-tests.ts

The calculator
  can add
    ✓ -1 + 1 === 0
    ✓ 1 + 2 === 3
    ✓ 0.1 + 0.2 is close to 0.3
  can subtract
    ✓ -1 - 1 === -2
    ✓ 1 - 2 === -1
    ✓ 0.1 - 0.2 === -0.1
  can multiply
    ✓ -1 * 1 === -1
    ✓ 1 * 2 === 2
    ✓ 0.1 * 0.2 is close to 0.02
  can divide
    ✓ -1 / 1 === -1
    ✓ 1 / 2 === 0.5
    ✓ 0.1 / 0.2 is 0.5

12 passing (16ms)

```

## Solution

The Solution can be found in **complete** folder

