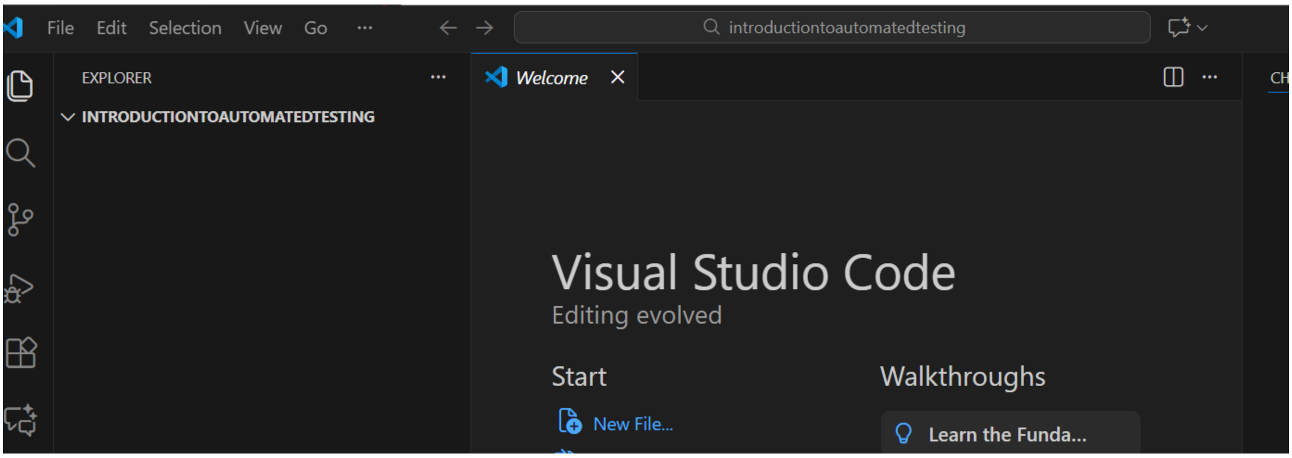
Walkthrough. Introduction to automated testing

Automated testing involves creating code or script, that is testing actual working code of an application. It could be even said that automated testing is “coding tests”. In this exercise a simple Node application calculating two numbers is created. Code will be tested automatically by creating test using Jest library.

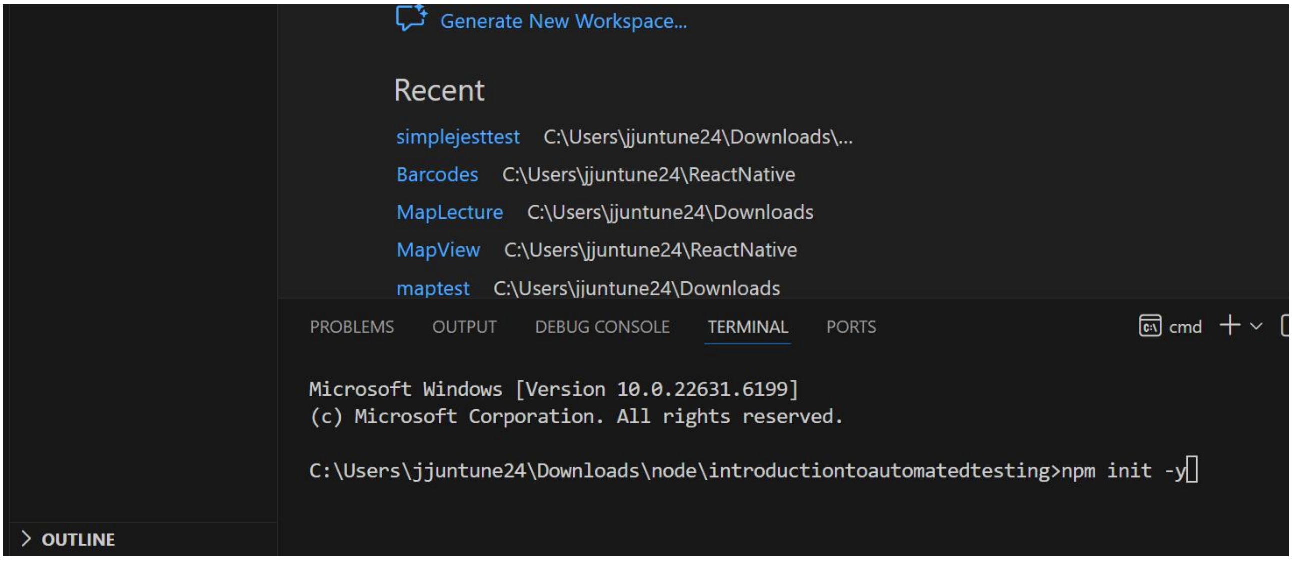
Learning objectives are:

- Creating simple Node project using ESM modules
- Installing and configuring Jest library
- Writing automated tests
- Positive and negative tests
- Running tests

Start by creating a new empty Node project. In this case it will be a simple application that is executed from command line. Create an empty folder for project and open it on Visual Studio. At this point there are no files, just empty folder opened on VS Code.

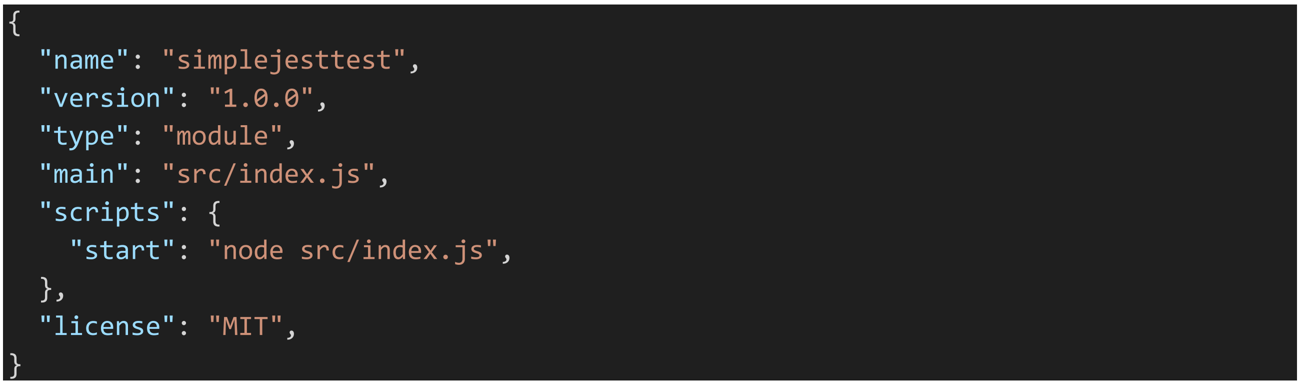


Run npm init -y command on terminal inside VS Code to create an empty Node project. This will create package.json file under project folder.


Generate New Workspace...
Recent
simplejesttest C:\Users\jjuntune24\Downloads\...
Barcodes C:\Users\jjuntune24\ReactNative
MapLecture C:\Users\jjuntune24\Downloads
MapView C:\Users\jjuntune24\ReactNative
maptest C:\Users\jjuntune24\Downloads
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
cmd
OUTLINE

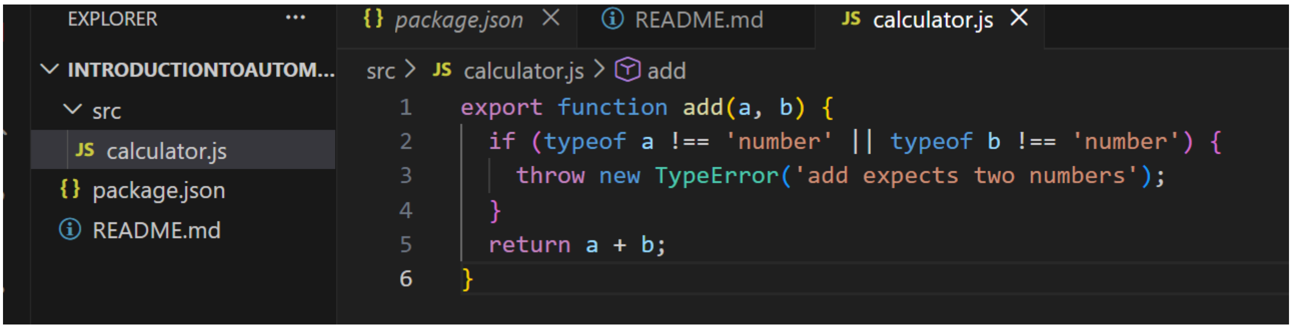
```
Microsoft Windows [Version 10.0.22631.6199]
(c) Microsoft Corporation. All rights reserved.

C:\Users\jjuntune24\Downloads\node\introductiontoautomatedtesting>npm init -y
```

Open package.json and change type from commonjs to module. This means that instead of require statements, import and export statements (ESM modules) can be used in Node. This is more modern way of using modules.



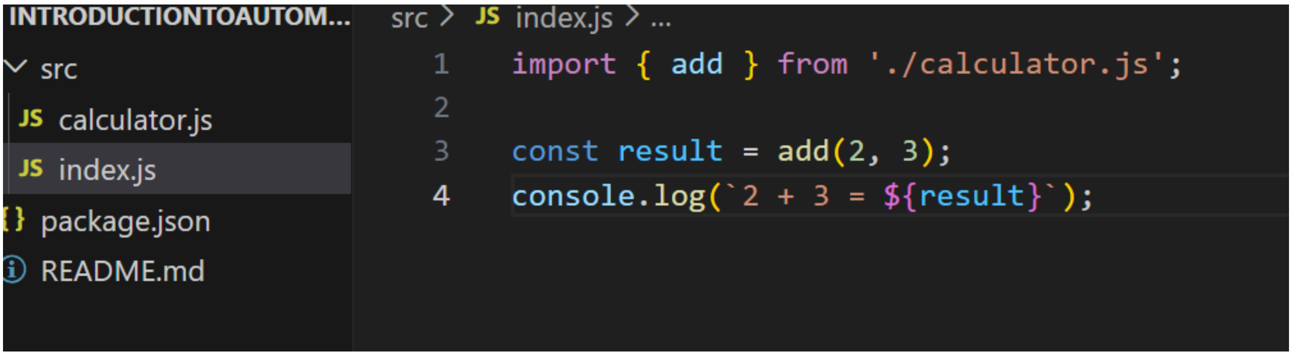
```
{
  "name": "simplejesttest",
  "version": "1.0.0",
  "type": "module",
  "main": "src/index.js",
  "scripts": {
    "start": "node src/index.js",
  },
  "license": "MIT",
}
```

Create src folder and calculator.js file with following content. A simple add function is created that calculates two numbers and returns result. Function is exported, so it can imported from other file/module.



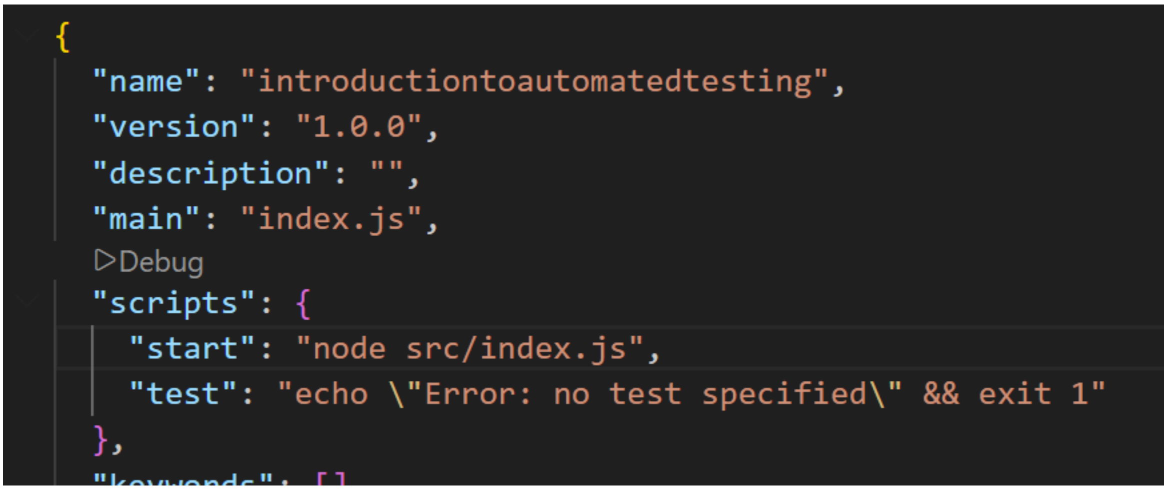
```
export function add(a, b) {
  if (typeof a !== 'number' || typeof b !== 'number') {
    throw new TypeError('add expects two numbers');
  }
  return a + b;
}
```

Create index.js under project's root folder. Code will import add function, call it and print out the result to the console.



```
import { add } from './calculator.js';

const result = add(2, 3);
console.log(`2 + 3 = ${result}`);
```

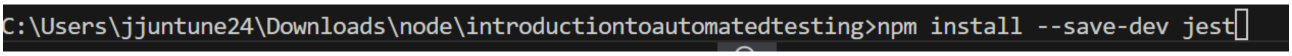
Edit package.json and add start script, which points to index.js under src folder.



```
{
  "name": "introductiontoautomatedtesting",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node src/index.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": []
```

Test out that app works. Open terminal and run application.

Next tests are created. Start by installing jest library. Library is installed just for development (not production), since testing is done in testing environment and not while app is really used in production.



```
C:\Users\jjuntune24\Downloads\node\introductiontoautomatedtesting>npm install --save-dev jest
```

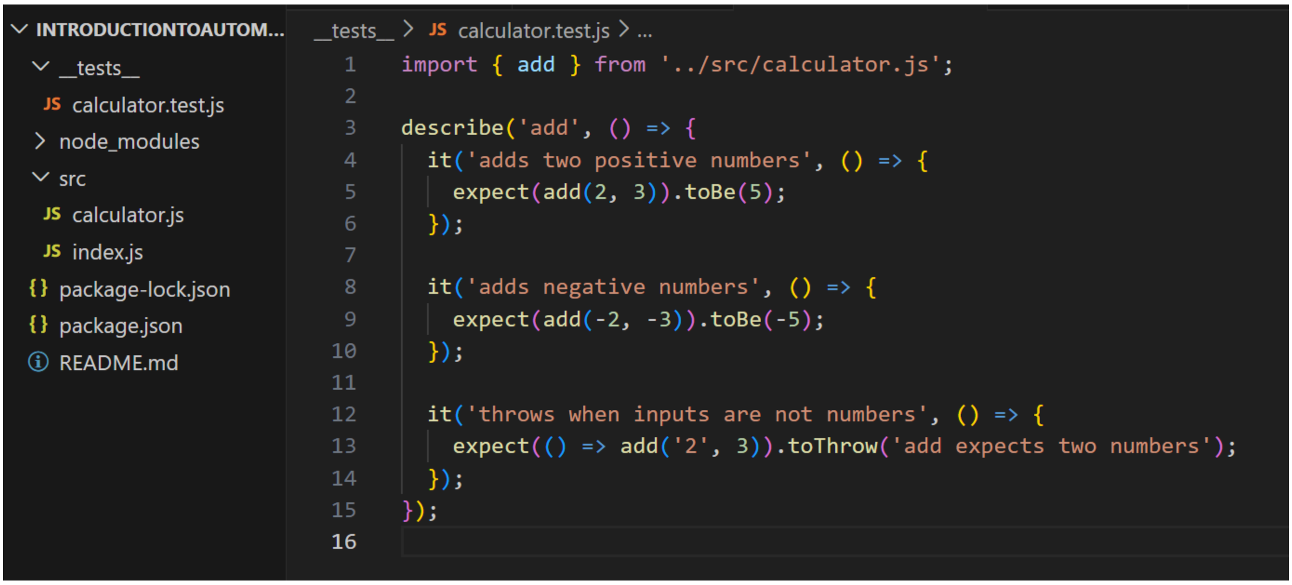
Package.json contains now devDependencies with jest. Edit scripts and test under it. Jest will use experimental library to use ESM modules.

```
{
  "name": "introductiontoautomatedtesting",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "node src/index.js",
    "test": "node --experimental-vm-modules node_modules/jest/bin/jest.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "module",
  "devDependencies": {
    "jest": "^30.2.0"
  }
}
```

Create jest.config.js under project's root folder. Following code will also enable use of ESM modules.

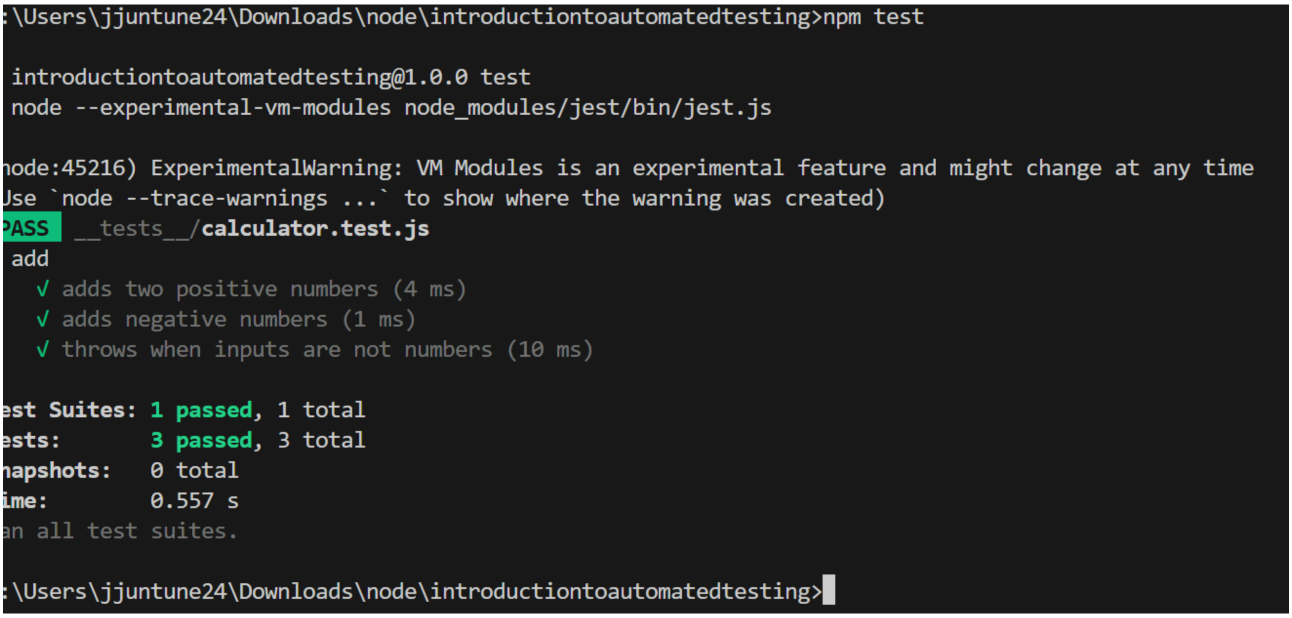
```
export default {
  testEnvironment: 'node',
  // Disable transforms; we are using native ESM
  transform: {}
};
```

Create __tests__ folder with calculator.test.js file (notice two underscores in the beginning and end of folder name). Code will create a test suite under name add. In tes suite there are three tests. First two are so called positive tests, where successful outcome is expected. The last one is called negative test, and it is expected that exception will be thrown. Test case has a name and expect will verify returned result/output.



```
import { add } from '../src/calculator.js';

describe('add', () => {
  it('adds two positive numbers', () => {
    expect(add(2, 3)).toBe(5);
  });

  it('adds negative numbers', () => {
    expect(add(-2, -3)).toBe(-5);
  });

  it('throws when inputs are not numbers', () => {
    expect(() => add('2', 3)).toThrow('add expects two numbers');
  });
});

```

Run tests with npm test command. All tests should pass.



```
:\Users\jjuntune24\Downloads\node\introductiontoautomatedtesting>npm test

 introductiontoautomatedtesting@1.0.0 test
 node --experimental-vm-modules node_modules/jest/bin/jest.js

node:45216) ExperimentalWarning: VM Modules is an experimental feature and might change at any time
Use `node --trace-warnings ...` to show where the warning was created)
PASS  __tests__/calculator.test.js
 add
   √ adds two positive numbers (4 ms)
   √ adds negative numbers (1 ms)
   √ throws when inputs are not numbers (10 ms)

est Suites: 1 passed, 1 total
ests:       3 passed, 3 total
napshots:   0 total
ime:        0.557 s
an all test suites.

:\Users\jjuntune24\Downloads\node\introductiontoautomatedtesting>
```