# Workshop

## Software Testing Tools

node JS

Jest

# Agenda

- **Node.js**
- **JEST**
- Installing Jest
- Preparing Code Before Testing
- Writing Test Scripts
- Running Test Scripts
- Example 1 + Exercise: Sum Number Program
- Example 2 + Exercise: Triangle Program

# NodeJs

- Node.js is JavaScript runtime.

- This example use program written in javaScript.

- “npm” is a package manager commands used by Node.js

- Download >> https://nodejs.org/en

# JEST

- Jest is a delightful JavaScript Testing Framework with a focus on simplicity.

- It works with projects using: Babel, TypeScript, Node, React, Angular, Vue and more!

- Pros
  - Zero config
  - Snapshot
  - Isolate
  - Great API

- source : [https://jestjs.io/](https://jestjs.io/)

# Installing Jest (1/3)

$ mkdir “ProjectDirectoryName”

$ cd “/yourProjectDirectory”

$ npm init

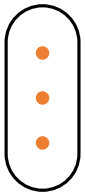
Configure a new Node.js project

```
package name: (jest_workshop)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
```

package.json

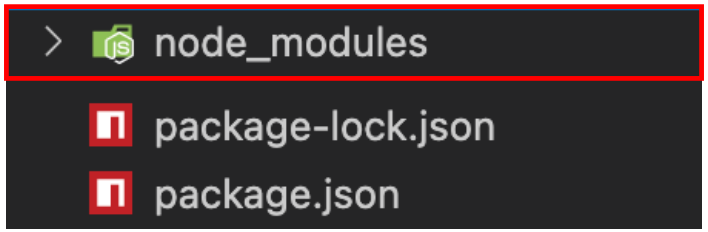
```
{
  "name": "jest_workshop",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "directories": {
    "test": "test"
  },
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}


Is this OK? (yes)
```

# Installing Jest (2/3)



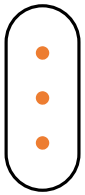
```
$ npm install --save-dev jest
```

Project Structure




--save-dev คือ การติดตั้ง package ของ jest เก็บไว้ใน **devDependencies** ในไฟล์ package.json
(devDependencies = แพ็กเกจที่ใช้เฉพาะตอนพัฒนา/ทดสอบโค้ด)

# Installing Jest (3/3)



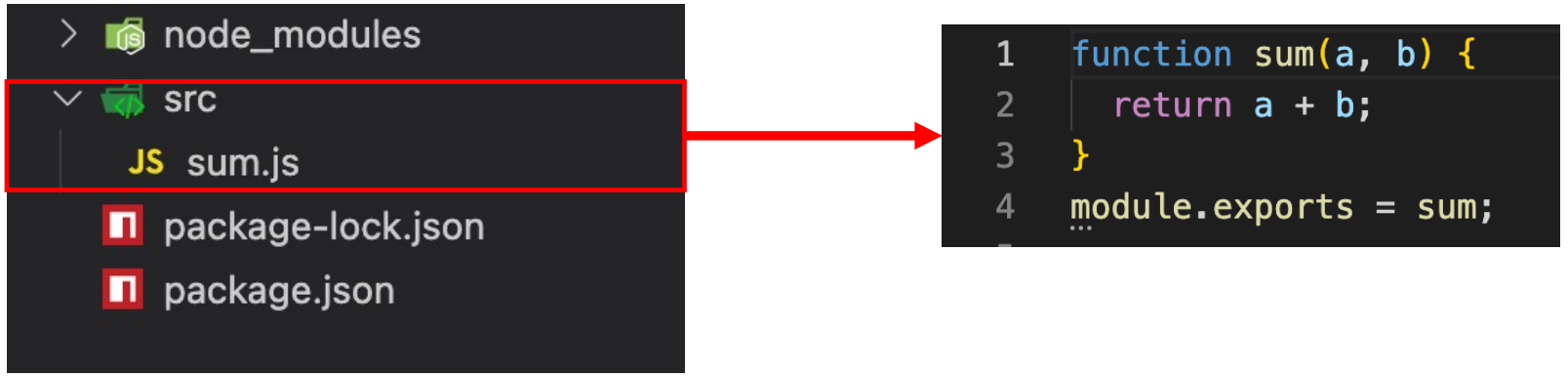
อัพเดท script ภายในไฟล์ package.json

```
“scripts” : {
        “test” : “jest”,
        “test-coverage” : “jest --watchAll --coverage”
}
```

watchAll คือ สำหรับสั่งให้ run test ทุกครั้งที่มีการเปลี่ยนแปลงของ test script/source code
coverage คือ สำหรับสั่งให้ run และแสดงผล code coverage ของการ test

# Preparing Code Before Testing

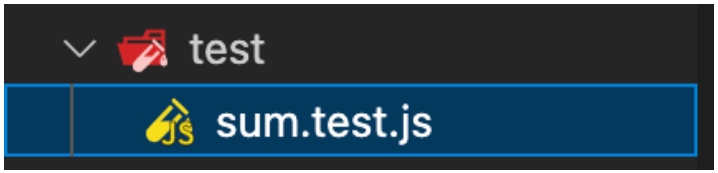
Project Structure



# Writing Test Scripts (1/3)

1.สร้าง folder “test” สำหรับจัดเก็บ test script


2.สร้างไฟล์ test script ดังนี้ “filename”.test.js เช่น sum.test.js




3.Import ชื่อฟังก์ชันที่ต้องการทดสอบในไฟล์ sum.test.js

```
1   const sum = require("../src/sum");
```

sum.test.js

# Writing Test Scripts (2/3)

| คำสั่ง / เมธอด | ใช้ทำอะไร |
|---|---|
| `test(name, fn)` หรือ `it(name, fn)` | ใช้สำหรับเขียน `unit test` เพื่อบอกว่าเรากำลังทดสอบอะไร |
| `expect(received)` | ใช้เตรียมค่าจากที่ได้จากการรันโค้ดเพื่อเตรียมตรวจสอบกับ `matcher` |
| **`Matcher`** เช่น `.toBe(expected)` | ฟังก์ชันเปรียบเทียบค่าจริงกับค่าที่คาดหวัง `.toBe()` |

**สรุป** ขั้นตอนสร้าง test script สำหรับ Jest

1.เขียนคำสั่ง test และกำหนดชื่อ test case

2.เขียนคำสั่ง expect และเรียกใช้ฟังก์ชันที่ต้องการทดสอบจากโค้ด

3.ใช้ matcher ตรวจสอบ.toBe()

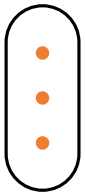
4.รันแล้วจะได้ผลลัพธ์การทดสอบ

- ถ้าเท่ากัน : ผ่าน (PASS)
- ถ้าไม่เท่ากัน : ไม่ผ่าน (FAIL)

sum.test.js

```
1  const sum = require("../src/sum");
2
3  test("TC1: adds 1 + 2 to equal 3", () => {
4    expect(sum(1, 2)).toBe(3);
5  });
```

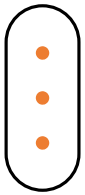
# Writing Test Scripts (3/3)



- describe(message, function) ใช้จัดกลุ่มการทดสอบได้

```
1   const sum = require("../src/sum");
2   describe("Group1", () => {
3     test("TC1: adds 1 + 2 to equal 3", () => {
4       expect(sum(1, 2)).toBe(3);
5     });
6   });
7
8   describe("Group2", () => {
9     test("TC1: adds 20 + 30 to equal 50", () => {
10      expect(sum(20, 30)).toBe(50);
11    });
12    test("TC1: adds 35 + 5 to equal 40", () => {
13      expect(sum(35, 5)).toBe(40);
14    });
15  });
```

# Running Test Scripts



1.เข้าไปที่ directory test

```
$ cd test
```

2.พิมพ์คำสั่งรันเทสต์สคริปต์

```
$ npm run test “filename”.test.js
```

**รันแบบระบุชื่อไฟล์**

```
$ npm run test-coverage “filename”.test.js
```

```
$ npm run test
```

**รันแบบไม่ระบุชื่อไฟล์**

```
$ npm run test-coverage
```

*ถ้าไม่ระบุชื่อไฟล์ที่ต้องการรัน JEST จะเรียกรันทุกเทสต์สคริปต์

*หากรันไม่ได้ให้ตรวจสอบ

1.ส่วนของการ import ของที่อยู่ไฟล์ source code ว่าถูกต้องหรือไม่

2.คำสั่งที่ระบุลงไปในไฟล์ package.json ส่วน scripts ว่าถูกต้องหรือไม่

# **Example 1** : Sum Numbers Program

```
1  function sum(a, b) {
2    return a + b;
3  }
4  module.exports = sum;
```

sum.js

**Exercise 1 :** ให้ทุกคนสร้าง test case สำหรับทดสอบฟังก์ชัน sum(a,b) และรันให้ได้ผลลัพธ์ดังรูปนี้

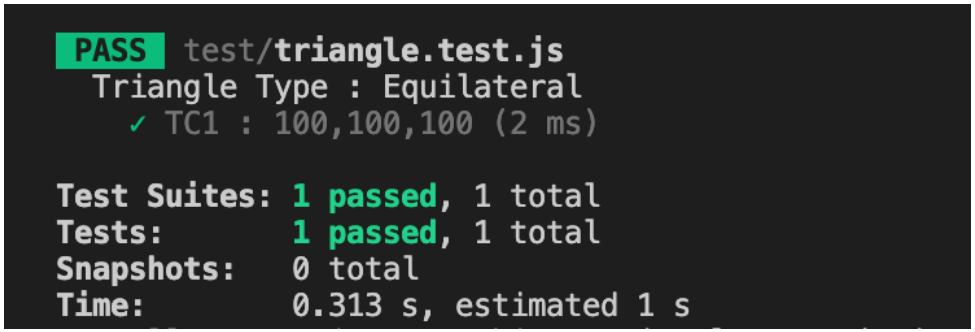
```
✓ TC1: adds 1 + 2 to equal 3
× TC2: adds 2 + 5 to equal 7
```

# Example 2 : Triangle Program (1/2)

Run test

```
$ npm run test triangle.test.js
```

Result



# Example 2 : Triangle Program (2/2)

Run test

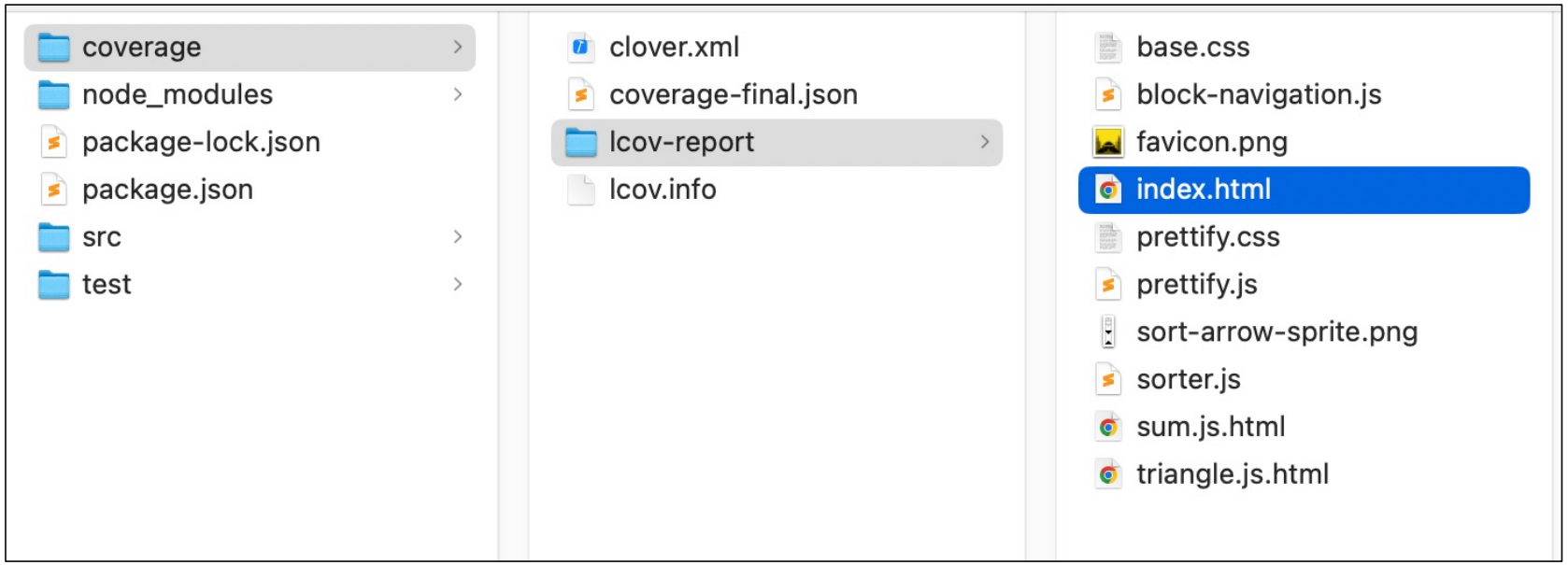
```
$ npm run test-coverage triangle.test.js
```

Result

```
 PASS  test/triangle.test.js
  Triangle Type : Equilateral
    ✓ TC1 : 100,100,100 (5 ms)

-------------|---------|----------|---------|---------|-------------------
File         | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------|---------|----------|---------|---------|-------------------
All files    |    62.5 |    66.66 |     100 |    62.5 |
 triangle.js |    62.5 |    66.66 |     100 |    62.5 | 8,10,13,18-21
-------------|---------|----------|---------|---------|-------------------
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.306 s
Ran all test suites matching /triangle.test.js/i.
```

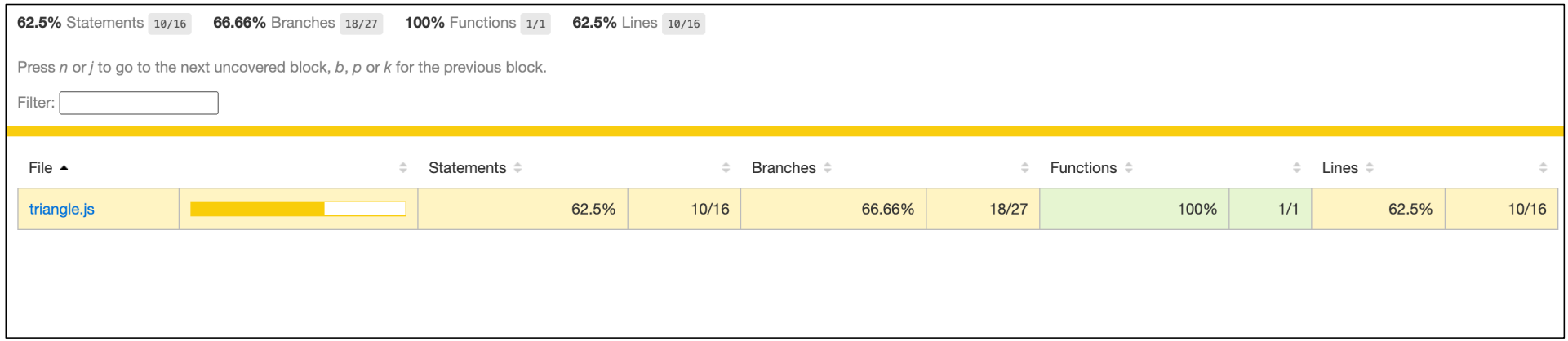
# Report (1/3)

ที่อยู่ของไฟล์ report : Project directory/coverage/lcov-report/index.html



# Report (2/3)

เปิดไฟล์ index.html (ไฟล์report)



# Report (3/3)

สามารถดูรายละเอียดของการ coverage ของ source code ได้

บรรทัดที่มี สีเขียว คือ บรรทัดของ source code ที่ได้ทำงานและตัวเลข 1x หมายถึง source code ได้ทำงานรอบ 1 เมื่อรันเทสต์สคริปต์

บรรทัดที่มี สีแดง คือ source code ไม่ได้ทำงาน เมื่อรันเทสต์สคริปต์

```
1    function TriangleCalculator(a, b, c) {
2  1x  a = parseFloat(a);
3  1x  b = parseFloat(b);
4  1x  c = parseFloat(c);
5      let output;
6
7  1x  I if (!Number.isInteger(a) || !Number.isInteger(b) || !Number.isInteger(c)) {
8        return "Value is not an integer";
9  1x  } else I if (a <= 0 || a > 200 || b <= 0 || b > 200 || c <= 0 || c > 200) {
10       return "Value is out of range";
11     }
12 1x  I if (a + b <= c || a + c <= b || b + c <= a) {
13       output = "Not a triangle";
14     }
15     //Check type
16 1x  else if (a == b && b == c) {
17 1x    output = "Equilateral";
18     } else E if (a == b || a == c || b == c) {
19       output = "Isosceles";
20     } else {
21       output = "Scalene";
22     }
23 1x  return output;
24   }
25
26 1x  module.exports = TriangleCalculator;
27
```

# Exercise 2

จงสร้างเทสต์สคริปต์ของ example 2 : Triangle Program ให้มีผลลัพธ์การรัน statement coverage และ branch coverage มีค่าเป็น 100 % โดยเทสต์สคริปต์ที่สร้างขึ้นต้องเรียกใช้ฟังก์ชัน TriangleCalculator(a, b, c) ซึ่งจะได้ผลลัพธ์เป็นค่าตัวอักษร (String) ที่เป็นประเภทของสามเหลี่ยม โดยมีประเภทของสามเหลี่ยม ได้แก่ Equilateral, Isosceles, Scalene และอินพุต a,b,c มีเงื่อนไขดังนี้

1. ค่า a,b,c คือ ความยาวของสามเหลี่ยมแต่ละด้าน
2. ค่า a,b,c มีค่าตั้งแต่ 1 – 200
3. ค่า a,b,c ต้องเป็นค่าตัวเลขเท่านั้น
4. สามเหลี่ยมมีคุณสมบัติด้านสองด้านรวมกันต้องมากกว่าอีกด้านหนึ่ง (a+b>c หรือ a+c>b หรือ c+b>a)