

8월 22일 (일)

1. Codecademy - TDD Fundamentals - Write Good Tests with Mocha

Lesson 2. Write Expressive Tests

- Introduction
- `assert.ok`
- `assert.equal`
- `assert.strictEqual`
- `assert.deepEqual` - 1 -
- `assert.deepEqual` - 2 -
- Other assert methods
- Review

- **Introduction**

이번 레슨에서 훨씬 더 표현력있는 테스트를 작성할 수 있는 노드의 `assert` 라이브러리를 어떻게 사용할 것인지에 대해 배울 것이다.

표현력을 가진 테스트는 읽기 쉽고, 설명적이다. 이는 구현 코드를 문서화 형식으로 만들어 유용하게 한다. 테스트를 더 표현력있게 만드는 한 가지 방법은 기대결과를 실제결과와 비교하는 `verify` 단계를 명확히 하는 것이다.

Node.js 는 더 표현력있는 `verification` 코드를 작성하도록 돕는 메서드들을 포함하고 있는 `assert` 라이브러리를 제공한다. 라이브러리 안에 있는 메서드를 통해 더 적은 코드를 작성하게 하고, 사람들이 읽을 수 있는 언어를 사용할 수 있게 된다.

assert 라이브러리는 Mocha 테스트 프레임워크에서 사용될 수 있으며, 이번 레슨에서 사용 방법에 대해 배울 것이다.

```
1  const assert = require('assert');
2
3  describe('+', () => {
4    it('returns the sum of its arguments', () => {
5
6      assert.ok(3 + 4 === 7);
7
8    });
9  });
10
```

```
$ npm test
> learn-mocha-intro-start@1.0.0 test /home/ccuser/workspace/learn-mocha-expressive-tests-expressive-introduction
> mocha test/**/*.test.js

+
  ✓ returns the sum of its arguments

1 passing (5ms)
```

하나의 assert 메서드가 it 블록에 포함되어 있다.

- **assert.ok**

테스트를 작성하기 위해 우리는 Node.js에서 제공하는 assert.ok 메서드를 사용할 수 있다. 프로그래밍에서 테스트는 기대값과 실제값을 비교한다.

```
const a = 1 + 2;
```

다음 값을 테스트해본다면, a는 3이 되도록 기대된다. 바닐라 자바스크립트에서 조건문을 작성하여 결과값과 기대값을 비교하여 에러를 낼 수 있다. 우리가 사용할 `assert.ok` 도 이러한 기능을 갖는데, 이 메서드 또한 값을 비교하고 에러를 던져준다.

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

노드의 하나의 모듈로써 이와 같이 import 해줘야한다.

```
assert.ok(a === 3);
```

이와 같이 사용하면된다. 이러한 경우 `assert.ok` 는 `True` 를 출력하며 에러를 던지지 않는다.
이 메서드가 `false` 를 출력할 땐, `AssertionError` 를 던지며, 모카는 테스트가 실패했다는 문구를 출력하며, 에러 메시지를 콘솔안에 로그한다.

Node.js 공식문서 : <https://nodejs.org/api/documentation.html>

- `assert.equal`

대부분의 입장에서 `assert.ok()` 를 사용해도 좋지만, 때때로 너가 평가해야하는 조건을 결정하는데에 어려움을 겪을 수도 있다.

```
const landAnimals = ['giraffe', 'squirrel'];  
const waterAnimals = ['shark', 'stingray'];  
  
landAnimals.push('frog');  
waterAnimals.push('frog');  
  
assert.ok(landAnimals[2] == waterAnimals[2]);
```

해당 코드는 에러를 던질까?

위 선언문은 동일함을 확인하는 선언문이다. 이 선언문을 이해하기 위해 너는 `ok()` 괄호 안에 있는 표현식을 모두 이해하여 평가해야만 한다. 이렇게 두 값의 동일함을 체크할 때는 `assert.equal` 함수를 사용하자.

```
assert.ok(landAnimals[2] == waterAnimals[2]);
assert.equal(landAnimals[2], waterAnimals[2]);
```

두 가지 방법 모두 같은 값을 도출해내며, 두번째 코드가 훨씬 가독성이 높다.

실습 예제

```
1 // Import assert here
2 const assert = require('assert');
3
4 describe('-', () => {
5   it('returns the difference of two values', () => {
6     const bigNum = 100;
7     const smallNum = 4;
8     const expected = 96;
9
10    const result = bigNum - smallNum;
11
12    // Write assertion here
13    assert.equal(result, expected);
14  });
15 });
16
```

```
$ npm test
> learn-mocha-intro-start@1.0.0 test /home/ccuser/workspace/learn-mocha-expressive-tests-assert-equal
> mocha test/**/*.test.js

-
  ✓ returns the difference of two values

1 passing (6ms)

$
```

- **assert.strictEqual**

```
const a = 3;
const b = '3';
assert.ok(a == b);
assert.ok(a === b);
```

다음 두 결과는 True 를 리턴할까 False 를 리턴할까?

첫번째 equal(==) 은 loose equal 이라고 하며, 두번째 equal 은(===) strict equal 이라고 한다.

즉, 첫번째 결과는 true로 '3' 이 타입전환을 통해 같다고 인식하며, 두번째 결과는 false 를 반환한다.

이 또한 아래와 같이 대체할 수 있는 가독성이 좋은 명령어가 존재한다.

```
const a = 3;
const b = '3';
assert.equal(a, b);
assert.strictEqual(a, b);
```

공식문서에 따르면, 올해부터 equal 말고 항상 strictEqual 을 쓰는 것을 권고한다!

실습 예제

```
1  // Import assert here
2  const assert = require('assert');
3
4  ▼ describe('-', () => {
5  ▼  it('returns the difference of two values', () => {
6      const bigNum = 100;
7      const smallNum = 4;
8      const expected = '96';
9
10     const result = bigNum - smallNum;
11
12     // Write assertion here
13     assert.equal(result, expected);
14   });
15 });
16
```

변수 expected 를 '96' 문자열로 바꿔도 assert.equal 이기 때문에 pass 가 될 것이다.

```
1 // Import assert here
2 const assert = require('assert');
3
4 describe('-', () => {
5   it('returns the difference of two values', () => {
6     const bigNum = 100;
7     const smallNum = 4;
8     const expected = '96';
9
10    const result = bigNum - smallNum;
11
12    // Write assertion here
13    assert.strictEqual(result, expected);
14  });
15 });
16
```

```
$ npm test
> learn-mocha-intro-start@1.0.0 test /home/ccuser/workspace/learn-mocha-expressive-tests-assert-strict-equal
> mocha test/**/*.test.js

-
  1) returns the difference of two values

0 passing (7ms)
1 failing

1) - returns the difference of two values:
     AssertionError: 96 === '96'
       at Context.it (test/index_test.js:13:12)

npm ERR! Test failed.  See above for more details.
$
```

strictEqual()로 바꾸니 test 결과 error 뜬다.

- **assert.deepEqual - 1 - (객체를 비교할 때)**

```
const a = {relation: 'twin', age: '17'};
const b = {relation: 'twin', age: '17'};
assert.equal(a, b);
assert.strictEqual(a, b);
```

이 두 가지 선언은 에러를 던질 것인가?

두 선언은 에러를 던질 것이다. 왜냐하면 자바스크립트에서 별개의 객체들은 loose or strict equal 를 사용할 때 같다고 여기지 않기 때문이다.

만약 두 객체를 비교하기 위해서는 assert.deepEqual()을 사용할 수 있다. 이 메서드는 각각 객체의 값을 loose equal(==) 을 사용하여 비교한다. (ex 각 객체의 프로퍼티가 3, '3' 이라면 같다고 본다.)

```
assert.deepEqual(a, b);
```

해당 코드는 에러를 던지지 않을 것이다.

```
a.relation == b.relation;  
a.age == b.age;
```

그리고 각 객체의 relation 과 age 프로퍼티를 수동적으로 비교하여 확인할 수 있다.

실습 예제

index_test.js

```
1  const assert = require('assert');  
2  
3  ▼ describe('+', () => {  
4    ▼ it('returns the sum of two values', () => {  
5      // Setup  
6      let expected = {a: 3, b: 4, result: 7};  
7      let sum = {a: 3, b: 4};  
8  
9      // Exercise  
10     sum.result = sum.a + sum.b;  
11  
12     // Verify  
13     assert.deepEqual(sum, expected);  
14   });  
15 });  
16
```

- **assert.deepEqual - 2 - (배열을 비교할 때)**

이전 연습에서는 두 객체의 값을 루즈 이퀄리티로 비교했다. Array(배열) 또한 객체이므로, `deepEqual()` 은 배열 또한 루즈하게 비교할 수 있다.

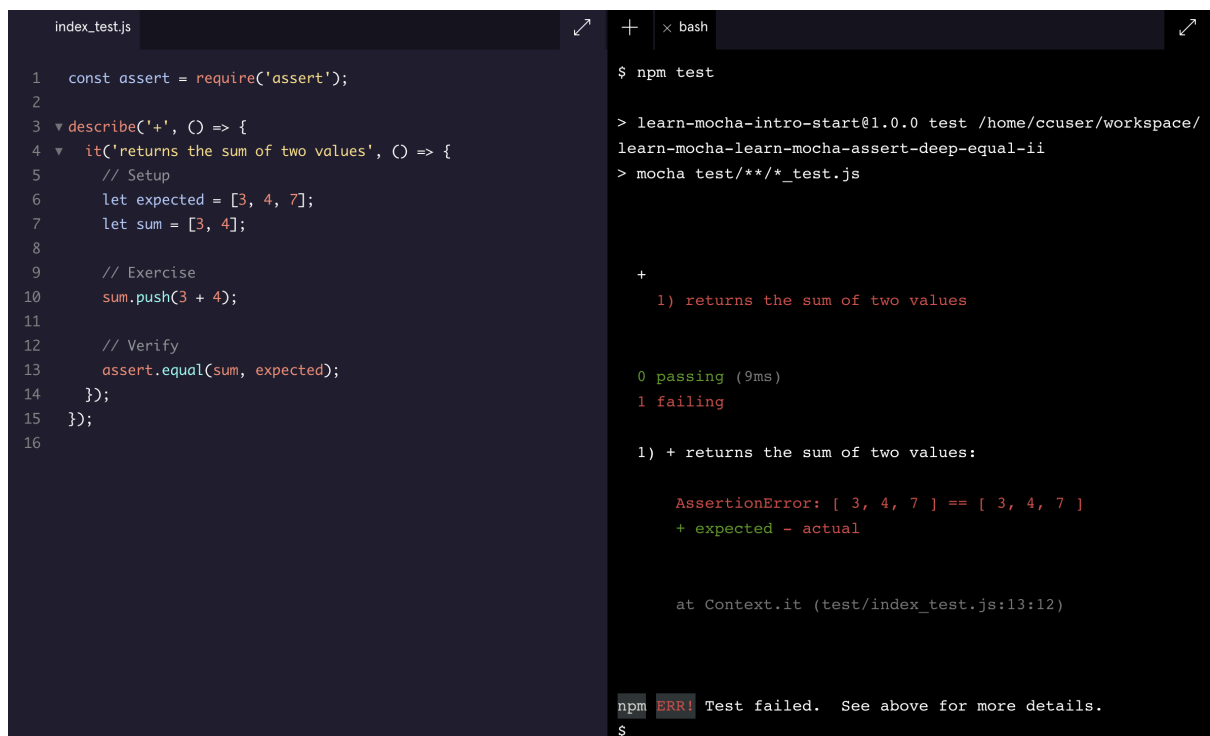
```
const arr1 = [1, 2, 3];
const arr2 = [1, 2, 3];
const arr3 = [1, 2, '3'];

assert.deepEqual(arr1, arr2); // No error
assert.deepEqual(arr1, arr3); // No error
```

`deepEqual()` 에 대한 공식문서 참조:

https://nodejs.org/api/assert.html#assert_assert_deepequal_actual_expected_message

실습 예제



The screenshot shows a code editor with two panes. The left pane displays the content of `index_test.js`, which includes a Mocha test suite. The right pane shows the terminal output of running `npm test`. The test suite in `index_test.js` defines a function `describe` with a test `it` that checks if the sum of two values is stored in an array. The test setup defines `expected` as `[3, 4, 7]` and `sum` as `[3, 4]`. The exercise pushes `3 + 4` to the `sum` array. The verification step uses `assert.equal(sum, expected)`. The terminal output shows that the test failed with an `AssertionError` because the actual array `[3, 4, 7]` does not equal the expected array `[3, 4]`.

```
index_test.js
1  const assert = require('assert');
2
3  describe('+', () => {
4    it('returns the sum of two values', () => {
5      // Setup
6      let expected = [3, 4, 7];
7      let sum = [3, 4];
8
9      // Exercise
10     sum.push(3 + 4);
11
12     // Verify
13     assert.equal(sum, expected);
14   });
15 });
16
```

```
+ x bash
$ npm test
> learn-mocha-intro-start@1.0.0 test /home/ccuser/workspace/learn-mocha-learn-mocha-assert-deep-equal-ii
> mocha test/**/*.test.js

+
  1) returns the sum of two values

0 passing (9ms)
1 failing

1) + returns the sum of two values:
     AssertionError: [ 3, 4, 7 ] == [ 3, 4, 7 ]
       + expected - actual

       at Context.it (test/index_test.js:13:12)

npm ERR! Test failed.  See above for more details.
$
```

`equal` 로 비교하면 에러가 뜬다.


```
index_test.js
1  const assert = require('assert');
2
3  ▼ describe('+', () => {
4  ▼  it('returns the sum of two values', () => {
5      // Setup
6      let expected = [3, 4, 7];
7      let sum = [3, 4];
8
9      // Exercise
10     sum.push(3 + 4);
11
12     // Verify
13     assert.deepEqual(sum, expected);
14   });
15 });
16
```

```
+ x bash
0 passing (9ms)
1 failing

1) + returns the sum of two values:

   AssertionError: [ 3, 4, 7 ] == [ 3, 4, 7 ]
   + expected - actual

   at Context.it (test/index_test.js:13:12)

npm ERR! Test failed.  See above for more details.
$ npm test

> learn-mocha-intro-start@1.0.0 test /home/ccuser/workspace/learn-mocha-learn-mocha-assert-deep-equal-ii
> mocha test/**/*.test.js

+
  ✓ returns the sum of two values

1 passing (5ms)

$
```

deepEqual 로 바꾸면 테스트는 pass 할 것이다.

• Other assert methods

Node 의 assert 라이브러리는 지금까지 배운 4개의 메서드 뿐 만 아니라 더 많은 메서드를 포함하고 있다.

assert.ok(1 == 1) 을 assert.equal(1,1) 로 구현할 수 있는 것처럼 많은 메서드들은 다른 메서드로 대체하여 구현될 수 있다. 이러한 assert.equal, assert.strictEqual, assert.deepEqual 과 같은 메서드들은 훨씬 더 가독성을 증가시킨다. 누군가 테스트를 읽고 구현코드에 대한 기대 결과를 분명히 할 수 있다.

실습 예제

```
index_test.js
1  const assert = require('assert');
2
3  describe('Numbers', () => {
4    it('1 does not equal 2', () => {
5      // Verify
6      assert.ok(1 !== 2);
7    });
8  });
9

bash
$ npm start
npm ERR! Linux 4.14.238-182.422.amzn2.x86_64
npm ERR! argv "/usr/bin/nodejs" "/usr/bin/npm" "start"
npm ERR! node v7.10.1
npm ERR! npm  v4.2.0

npm ERR! missing script: start
npm ERR!
npm ERR! If you need help, you may report this error at:
npm ERR!   <https://github.com/npm/npm/issues>

npm ERR! Please include the following file with any support
request:
npm ERR!   /home/ccuser/.npm/_logs/2021-08-22T14_42_10_036
Z-debug.log
$ npm test

> learn-mocha-intro-start@1.0.0 test /home/ccuser/workspace/
learn-mocha-learn-mocha-other-assert-methods
> mocha test/**/*.test.js

Numbers
  ✓ 1 does not equal 2

1 passing (7ms)

$
```

```
index_test.js
1  const assert = require('assert');
2
3  describe('Numbers', () => {
4    it('1 does not equal 2', () => {
5      // Verify
6      assert.notStrictEqual(1, 2);
7    });
8  });
9

bash
$ npm test

> learn-mocha-intro-start@1.0.0 test /home/ccuser/workspace/learn-mocha-learn-mocha-other-assert-methods
> mocha test/**/*.test.js

Numbers
  ✓ 1 does not equal 2

1 passing (5ms)

$
```

assert.ok(1 !== 2); 는 assert.notStrictEqual() 로 대체가 가능하다.

- assert.notStrictEqual() 메서드는 두 값이 다르면 pass 를 출력한다.

더 많은 assert 메서드 참조

https://nodejs.org/api/assert.html#assert_assert_notequal_actual_expected_message

• Review

이번 레슨을 통해 Node 의 assert 라이브러리를 사용할 수 있게 되었다.

⇒ assert.equal() : loose equality (==)

⇒ assert.strictEqual() : Strict equality (===)

⇒ assert.deepEqual() : 두 객체, 배열을 loose equality 로 비교

⇒ assert 의 다른 메서드를 사용하여 훨씬 더 표현적(가독성) 있도록 만들 수 있다.

앞으로 테스트를 작성할 때, 항상 "좋은 테스트의 특성들(the characteristics of a good test)"의 6가지 기준(fast, complete, reliable, isolated, maintainable, expressive)으로 평가해야하는 것을 기억해라. 만약 이 6가지 기준을 충족한다면, 높은 퀄리티의 테스트 프레임워크를 생성하게 될 것이다.

Quiz. 몰랐던 부분

`describe` blocks can be nested to reflect the structure of the code which they are testing. Which test block best reflects the structure of the following code?

```
const cup = {};  
  
cup.pour = () => { return 'Poured once!'; }  
  
cup.fill = () => { return 'Cup filled!'; }
```

```
describe('cup', () => {  
  describe('.empty', () => {  
  
  });  
  describe('.drink', () => {  
  
  });  
});
```

```
describe('cup', () => {  
  describe('.pour', () => {  
  
  });  
  describe('.fill', () => {  
  
  });  
});
```

What is the outcome of the following code?

```
assert.ok(false);
```

The function returns `false`

`false` is converted to a string

An error is thrown

 Correct!

Nothing


What step must be done before using the `describe` and `it` functions in a test suite?

Use `npm test`

Type `require('mocha')` at the top of the file

Use `mocha.describe()` and `mocha.it()` in your code

Use `npm install mocha -D` to install Mocha

 Correct! Once Mocha is installed with npm, you can call the two functions in your project.

What is the expected terminal output of this code?

```
// Documentation for hooks: https://mochajs.org/#hooks

describe('interrupting cow', () => {
  before(() => {
    console.log('cow enters the conversation...');
  });

  afterEach(() => {
    console.log('MOO!');
  });

  it('has four legs', () => {});
  it('moos above 20 decibels', () => {});
});
```

```
interrupting cow
cow enters the conversation...
  has four legs
MOO!
  moos above 20 decibels
MOO!
```



Correct!

오늘의 단어

- distinct : 뚜렷한, 분명한 ; 다른, 별개의
- reliable : 의지할 수 있는, 확실한, 믿음직한