

Lesson 04 Demo 08

Creating Test Cases

Objective: To create test cases using the test module and execute them

Tools required: Visual Studio Code and Node Package Manager

Prerequisites: Basic Linux and NPM commands

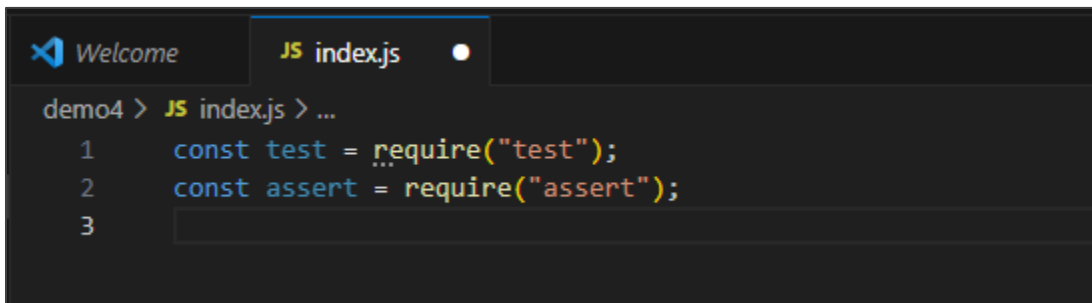
Steps to be followed:

1. Create tests using the test modules
2. Create tests with two subtests
3. Skip the test cases
4. Execute the test cases using the **describe** and **it** functions

Step 1: Create tests using the test module

1.1 Import the test and the assert module:

```
const test = require("test");  
const assert = require("assert");
```



```
demo4 > JS index.js > ...  
1  const test = require("test");  
2  const assert = require("assert");  
3
```

1.2 Create test cases that exhibit various scenarios:

```
test('synchronous passing test', (t) => {  
  // This test passes because it does not throw an exception.  
  assert.strictEqual(1.4, 1.4);  
});
```

```
test('synchronous failing test', (t) => {  
  // This test fails because it throws an exception.  
  assert.strictEqual(4, 6);  
});
```

```
test('asynchronous passing test', async (t) => {  
  // This test passes because the Promise returned by the async  
  // function is not rejected.  
  assert.strictEqual(3, 3);  
});
```

```
test('asynchronous failing test', async (t) => {  
  // This test fails because the Promise returned by the async  
  // function is rejected.  
  assert.strictEqual(7, 8);  
});
```

```
test('failing test using Promises', (t) => {  
  // Promises can be used directly as well.  
  return new Promise((resolve, reject) => {  
    setImmediate(() => {  
      reject(new Error('this will cause the test to fail'));  
    });  
  });  
});
```

```
test('callback passing test', (t, done) => {  
  // done() is the callback function. When the setImmediate() runs, it invokes  
  // done() with no arguments.  
  setImmediate(done);  
});
```

```
test('callback failing test', (t, done) => {
  // When the setImmediate() runs, done() is invoked with an Error object and
  // the test fails.
  setImmediate(() => {
    done(new Error('callback failure'));
  });
});
```

```
demo4 > JS index.js > ...
1  const test = require("test");
2  const assert = require("assert");
3
4  test('synchronous passing test', (t) => {
5    // This test passes because it does not throw an exception.
6    assert.strictEqual(1.4, 1.4);
7  });
8
9  test('synchronous failing test', (t) => {
10   // This test fails because it throws an exception.
11   assert.strictEqual(4, 6);
12 });
13
14 test('asynchronous passing test', async (t) => {
15   // This test passes because the Promise returned by the async
16   // function is not rejected.
17   assert.strictEqual(3, 3);
18 });
19
20 test('asynchronous failing test', async (t) => {
21   // This test fails because the Promise returned by the async
22   // function is rejected.
23   assert.strictEqual(7, 8);
24 });
25
26 test('failing test using Promises', (t) => {
27   // Promises can be used directly as well.
28   return new Promise((resolve, reject) => {
29     setImmediate(() => {
30       reject(new Error('this will cause the test to fail'));
31     });
32   });
33 });
34
35 test('callback passing test', (t, done) => {
36   // done() is the callback function. When the setImmediate() runs, it invokes
37   // done() with no arguments.
38   setImmediate(done);
39 });
40
41 test('callback failing test', (t, done) => {
42   // When the setImmediate() runs, done() is invoked with an Error object and
43   // the test fails.
44   setImmediate(() => {
45     done(new Error('callback failure'));
46   });
47 });
48
```

1.3 Execute the code using the following command:

node index.js

```
demopythonlyopm@ip-172-31-16-204:~/Desktop/nodeProjec/demo4$ node index.js
TAP version 13
# Subtest: synchronous passing test
ok 1 - synchronous passing test
...
duration_ms: 0.002393262
...
# Subtest: synchronous failing test
not ok 2 - synchronous failing test
...
duration_ms: 0.002083568
failureType: 'testCodeFailure'
error: |-
  Expected values to be strictly equal:

    1 !== 2

code: 'ERR_ASSERTION'
stack: |-
  TestContext.<anonymous> (/home/demopythonlyopm/Desktop/nodeProjec/demo4/index.js:11:12)
  Test.runInAsyncScope (node:async_hooks:203:9)
  exports.ReflectApply (/home/demopythonlyopm/Desktop/nodeProjec/demo4/node_modules/test/lib/internal/per_context/primordials.js:32:56)
  Test.run (/home/demopythonlyopm/Desktop/nodeProjec/demo4/node_modules/test/lib/internal/test_runner/test.js:485:25)
  Test.processPendingSubtests (/home/demopythonlyopm/Desktop/nodeProjec/demo4/node_modules/test/lib/internal/test_runner/test.js:251:27)
  Test.postRun (/home/demopythonlyopm/Desktop/nodeProjec/demo4/node_modules/test/lib/internal/test_runner/test.js:555:19)
  Test.run (/home/demopythonlyopm/Desktop/nodeProjec/demo4/node_modules/test/lib/internal/test_runner/test.js:513:10)
  processTicksAndRejections (node:internal/process/task_queues:96:5)
...
# Subtest: asynchronous passing test
ok 3 - asynchronous passing test
...
duration_ms: 0.000191914
...
# Subtest: asynchronous failing test
not ok 4 - asynchronous failing test
...
duration_ms: 0.000507923
failureType: 'testCodeFailure'
error: |-
  Expected values to be strictly equal:

    1 !== 2

code: 'ERR_ASSERTION'
stack: |-
  TestContext.<anonymous> (/home/demopythonlyopm/Desktop/nodeProjec/demo4/index.js:23:12)
  Test.runInAsyncScope (node:async_hooks:203:9)
  exports.ReflectApply (/home/demopythonlyopm/Desktop/nodeProjec/demo4/node_modules/test/lib/internal/per_context/primordials.js:32:56)
  Test.run (/home/demopythonlyopm/Desktop/nodeProjec/demo4/node_modules/test/lib/internal/test_runner/test.js:485:25)
  Test.processPendingSubtests (/home/demopythonlyopm/Desktop/nodeProjec/demo4/node_modules/test/lib/internal/test_runner/test.js:251:27)
  Test.postRun (/home/demopythonlyopm/Desktop/nodeProjec/demo4/node_modules/test/lib/internal/test_runner/test.js:555:19)
  Test.run (/home/demopythonlyopm/Desktop/nodeProjec/demo4/node_modules/test/lib/internal/test_runner/test.js:513:10)
  processTicksAndRejections (node:internal/process/task_queues:96:5)
  async Test.processPendingSubtests (/home/demopythonlyopm/Desktop/nodeProjec/demo4/node_modules/test/lib/internal/test_runner/test.js:251:7)
...
# Subtest: failing test using Promises
not ok 5 - failing test using Promises
...
duration_ms: 0.000647163
failureType: 'testCodeFailure'
error: 'this will cause the test to fail'
code: 'ERR_TEST_FAILURE'
stack: |-
  Immediate.<anonymous> (/home/demopythonlyopm/Desktop/nodeProjec/demo4/index.js:30:20)
  processImmediate (node:internal/timers:466:21)
...
# Subtest: callback passing test
ok 6 - callback passing test
...
duration_ms: 0.000245836
...
# Subtest: callback failing test
not ok 7 - callback failing test
...
duration_ms: 0.000287828
failureType: 'testCodeFailure'
error: 'callback failure'
code: 'ERR_TEST_FAILURE'
stack: |-
  Immediate.<anonymous> (/home/demopythonlyopm/Desktop/nodeProjec/demo4/index.js:45:14)
  processImmediate (node:internal/timers:466:21)
...
1..7
# tests 7
# pass 3
# fail 4
# cancelled 0
# skipped 0
# todo 0
# duration_ms 0.007383536
```

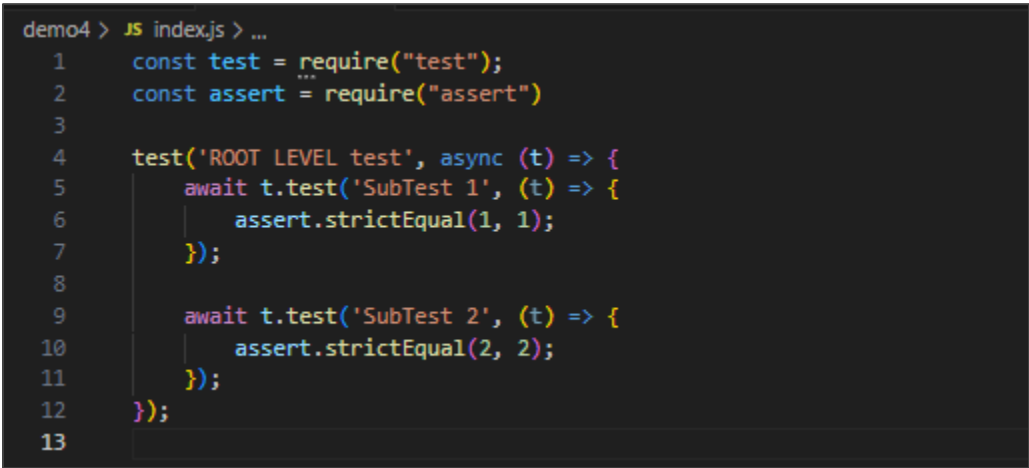
Step 2: Create tests with two subtests

2.1 Use the following code to create a top-level test:

```
const test = require("test");
const assert = require("assert")

test('ROOT LEVEL test', async (t) => {
  await t.test('SubTest 1', (t) => {
    assert.strictEqual(1, 1);
  });

  await t.test('SubTest 2', (t) => {
    assert.strictEqual(2, 2);
  });
});
```



```
demo4 > JS index.js > ...
1  const test = require("test");
2  const assert = require("assert")
3
4  test('ROOT LEVEL test', async (t) => {
5    await t.test('SubTest 1', (t) => {
6      assert.strictEqual(1, 1);
7    });
8
9    await t.test('SubTest 2', (t) => {
10     assert.strictEqual(2, 2);
11   });
12 });
13
```

Here, **await** ensures that both subtests are completed. This is necessary since parent test cases do not wait for their subtests to be completed.

Any subtests still outstanding when their parent finishes are canceled and treated as failures.

2.2 Execute the code to check the output:

```
demopython1yopm@ip-172-31-16-204:~/Desktop/nodeProjec/demo4$ node index.js
TAP version 13
# Subtest: ROOT LEVEL test
  # Subtest: SubTest 1
    ok 1 - SubTest 1
    ---
    duration_ms: 0.001932444
    ...
  # Subtest: SubTest 2
    ok 2 - SubTest 2
    ---
    duration_ms: 0.000159077
    ...
  1..2
ok 1 - ROOT LEVEL test
---
duration_ms: 0.006160685
...
1..1
# tests 1
# pass 1
# fail 0
# cancelled 0
# skipped 0
# todo 0
# duration_ms 0.07990543
```

Step 3: Skip the test cases

3.1 Use the following code to skip a test case using the **skip()** method

```
const test = require("test");
const assert = require("assert");

// The skip option is used, but no message is provided.
test('skip option', { skip: true }, (t) => {
  // This code is never executed.
});

// The skip option is used, and a message is provided.
test('skip option with message', { skip: 'this is skipped' }, (t) => {
  // This code is never executed.
});

test('skip() method', (t) => {
```

```
// Make sure to return here as well if the test contains additional logic.
t.skip();
});
```

```
test('skip() method with message', (t) => {
  // Make sure to return here as well if the test contains additional logic.
  t.skip('this is skipped');
});
```

```
demo4 > JS index.js > ...
1  const test = require("test");
2  const assert = require("assert");
3
4  // The skip option is used, but no message is provided.
5  test('skip option', { skip: true }, (t) => {
6    // This code is never executed.
7  });
8
9  // The skip option is used, and a message is provided.
10 test('skip option with message', { skip: 'this is skipped' }, (t) => {
11   // This code is never executed.
12 });
13
14 test('skip() method', (t) => {
15   // Make sure to return here as well if the test contains additional logic.
16   t.skip();
17 });
18
19 test('skip() method with message', (t) => {
20   // Make sure to return here as well if the test contains additional logic.
21   t.skip('this is skipped');
22 });
23
```

3.2 Run the code using the command **node index.js** in the terminal window

```
...
# Subtest: skip() method with message
ok 4 - skip() method with message # SKIP this is skipped
...
  duration_ms: 0.000095926
...
1..4
# tests 4
# pass 0
# fail 0
# cancelled 0
# skipped 4
# todo 0
# duration_ms 0.078008528
```

Step 4: Execute the test cases using the describe and it functions

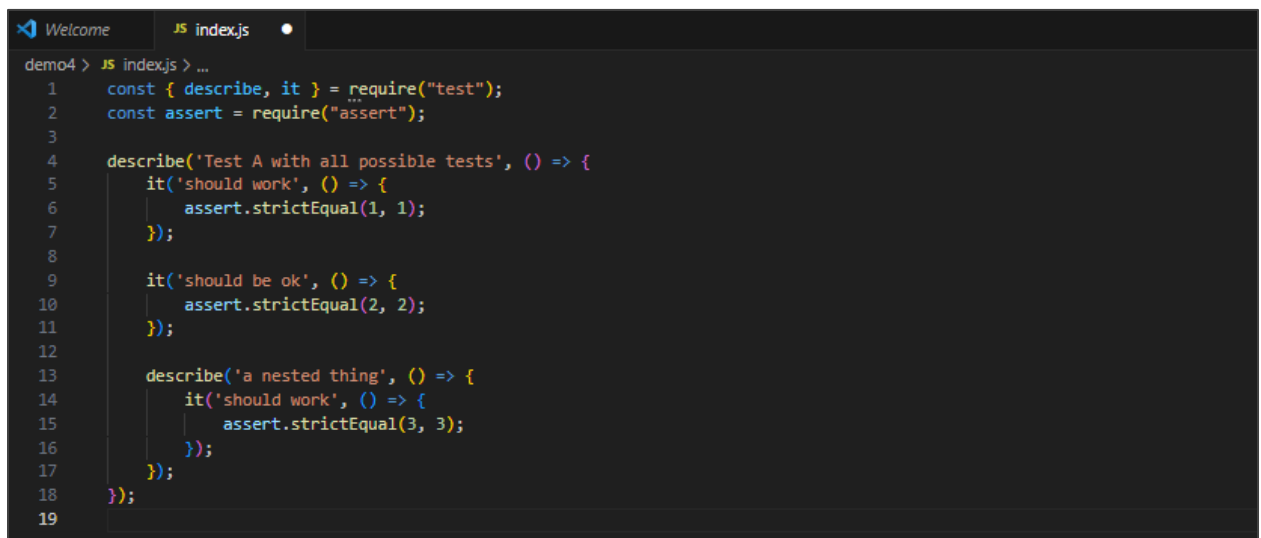
4.1 Use the following code to execute test cases using the **describe** and **it** functions:

```
const { describe, it } = require("test");
const assert = require("assert");

describe('Test A with all possible tests', () => {
  it('should work', () => {
    assert.strictEqual(1, 1);
  });

  it('should be ok', () => {
    assert.strictEqual(2, 2);
  });

  describe('a nested thing', () => {
    it('should work', () => {
      assert.strictEqual(3, 3);
    });
  });
});
```

A screenshot of a code editor with a dark theme. The editor has a tab labeled 'JS index.js'. The code is written in JavaScript and uses Jest's testing functions. It includes imports for 'describe', 'it', and 'assert'. The code structure is as follows: a 'describe' block for 'Test A with all possible tests' containing two 'it' blocks and a nested 'describe' block. The nested 'describe' block is for 'a nested thing' and contains one 'it' block. All 'it' blocks contain an 'assert.strictEqual' call. The code is numbered from 1 to 19.

```
demo4 > JS index.js > ...
1  const { describe, it } = require("test");
2  const assert = require("assert");
3
4  describe('Test A with all possible tests', () => {
5    it('should work', () => {
6      assert.strictEqual(1, 1);
7    });
8
9    it('should be ok', () => {
10     assert.strictEqual(2, 2);
11   });
12
13   describe('a nested thing', () => {
14     it('should work', () => {
15       assert.strictEqual(3, 3);
16     });
17   });
18 });
19
```


4.2 Execute the code using the following command:

node index.js

```
demopython1yopm@ip-172-31-16-204:~/Desktop/nodeProjec/demo4$ node index.js
TAP version 13
# Subtest: Test A with all possible tests
  # Subtest: should work
  ok 1 - should work
  ---
    duration_ms: 0.000354586
  ...
  # Subtest: should be ok
  ok 2 - should be ok
  ---
    duration_ms: 0.000091648
  ...
  # Subtest: a nested thing
    # Subtest: should work
    ok 1 - should work
    ---
      duration_ms: 0.000076374
    ...
    1..1
  ok 3 - a nested thing
  ---
    duration_ms: 0.000402602
  ...
  1..3
ok 1 - Test A with all possible tests
---
  duration_ms: 0.004277171
...
1..1
# tests 1
# pass 1
# fail 0
# cancelled 0
# skipped 0
# todo 0
# duration ms 0.081696604
```

By following these steps, you have successfully created executed test cases using the test module.