Mocha Testing Cheatsheet

A quick reference guide to Mocha, a JavaScript test framework, covering setup, assertions, hooks, and advanced features for effective testing.

Mocha Setup and Basics

Installation & Setup

Install Mocha via npm:

```
npm install --global mocha
```

Or as a dev dependency:

```
npm install --save-dev mocha
```

Create a `test` directory and a test file (e.g., `test.js`).

Add a test script to your `package.json`:

```
"scripts": {
  "test": "mocha"
}
```

Basic Test Structure

A basic test case includes `describe` (test suite) and `it` (test case) blocks:

```
describe('My Function', function() {
  it('should return true', function() {
    // Assertion here
  });
});
```

`describe(string, function)`: Defines a suite of tests.

`it(string, function)`: Defines an individual test case.

Running Tests

Run tests from the command line:

```
mocha
```

Or using the npm script:

```
npm test
```

Mocha will look for test files in the `test` directory by default.

To run a specific test file:

```
mocha test/my_test.js
```

Assertions and Hooks

Common Assertions

Mocha is often used with an assertion library like Chai. Here are some common assertions:

`assert.equal(actual, expected, message)`: Checks equality.

`assert.strictEqual(actual, expected, message)`: Checks strict equality.

`assert.notEqual(actual, expected, message)`: Checks inequality.

`assert.isTrue(value, message)`: Checks if a value is true.

`assert.isFalse(value, message)`: Checks if a value is false.

`assert.isNull(value, message)`: Checks if a value is null.

`assert.isNotNull(value, message)`: Checks if a value is not null.

Chai Assertions (BDD Style)

Chai provides `expect` and `should` styles:

`expect(actual).to.equal(expected)`: Checks equality.

`expect(actual).to.be.a('string')`: Checks type.

`expect(actual).to.be.true`: Checks if true.

`actual.should.equal(expected)` (requires `chai.should()`):

```
const chai = require('chai');
chai.should(); // Initialize should

const myVar = 5;
myVar.should.equal(5);
```

Hooks

Hooks are used to set up preconditions and clean up after tests.

`before(function)`: Runs once before all tests in a suite.

`after(function)`: Runs once after all tests in a suite.

`beforeEach(function)`: Runs before each test in a suite.

`afterEach(function)`: Runs after each test in a suite.

Example:

```
describe('My Suite', function() {
  beforeEach(function() {
    // Setup code before each test
  });

  afterEach(function() {
    // Cleanup code after each test
  });

  it('should do something', function() {
    // Test
  });
});
```

Asynchronous Testing

Testing Asynchronous Code

Mocha supports testing asynchronous code using callbacks, Promises, and async/await.

Using callbacks: `it(string, function(done))` - Pass `done` and call it when the async operation completes.

```
it('should call done()', function(done) {
  setTimeout(function() {
    assert.ok(true);
    done();
  }, 50);
});
```

Using Promises: Return a Promise from the `it` block.

```
it('should return a promise', function() {
  return new Promise(function(resolve, reject) {
    setTimeout(function() {
      resolve();
    }, 50);
  });
});
```

Using async/await:

```
it('should use async/await', async function() {
  await new Promise(resolve => setTimeout(resolve, 50));
  assert.ok(true);
});
```

For Promises, use `.then` and `.catch` to handle fulfillment and rejection.

Timeouts

Mocha has a default timeout of 2000ms. You can change it using `this.timeout(ms)` within a test or hook.

```
it('should take a long time', function(done) {
  this.timeout(5000); // Set timeout to 5 seconds
  setTimeout(done, 4000);
});
```

To disable timeouts, use `this.timeout(0)`.

Advanced Mocha Features

Pending Tests

You can define pending tests (tests without a function body) using `it` or `xit`.

```
it('should be implemented later');

xit('should also be implemented later',
function() {
  // This test will be skipped
});
```

Skipping Tests

You can skip tests using `.skip` or `this.skip()`.

```
it.skip('should skip this test', function() {
  // This test will be skipped
});

it('should skip conditionally', function() {
  if (condition) {
    this.skip();
  }
});
```

Only Running Specific Tests

You can run only specific tests using `.only`.

```
it.only('should only run this test',
function() {
  // This test will be the only one run
});

describe.only('My Suite', function() {
  // Only tests in this suite will be run
});
```

Reporters

Mocha supports different reporters to format test results. Specify the reporter using the `-R` flag.

```
mocha -R spec
```

Common reporters include `spec`, `list`, `progress`, `json`, `nyan`, etc.