Chai Assertion Library Cheatsheet

A comprehensive cheat sheet for the Chai assertion library, covering various assertion styles and methods for effective testing in JavaScript.

Introduction to Chai

Chai Overview

Chai is a BDD / TDD assertion library for node and the browser that can be paired with any JavaScript testing framework.

It provides a clean and readable syntax for writing test assertions.

Chai supports several interfaces: `should`, `expect`, and `assert`. This cheat sheet will primarily focus on `expect` and `assert` styles, which are commonly used.

Installation: `npm install chai --save-dev`

Importing Chai:

```
const chai = require('chai');
const expect = chai.expect; // for 'expect' style
const assert = chai.assert; // for 'assert' style
```

Expect Style Assertions

The `expect` style provides a chainable way to express assertions.

Basic Syntax: `expect(value).to.assertion`

Example:

```
expect(foo).to.equal('bar');
expect(list).to.have.lengthOf(3);
```

Assert Style Assertions

The `assert` style provides a more traditional way to express assertions, similar to standard `assert`.

Basic Syntax: `assert.assertion(value, message)`

Example:

```
assert.equal(foo, 'bar', 'foo should equal bar');
assert.lengthOf(list, 3, 'list should have a length of 3');
```

Common Assertions with Expect

Equality Assertions

`expect(value).to.equal(value2)`	Checks for deep equality (using `===`)
`expect(value).to.eql(value2)`	Checks for deep equality of objects and arrays
`expect(value).to.deep.equal(value2)`	Alias for `eql`
`expect(value).to.be.a(type)`	Checks the type of a value
`expect(value).to.be.an(type)`	Alias for `a`

Existence and Truthiness

`expect(value).to.exist`	Checks if a value is not `null` or `undefined`
`expect(value).to.not.exist`	Checks if a value is `null` or `undefined`
`expect(value).to.be.true`	Checks if a value is strictly `true`
`expect(value).to.be.false`	Checks if a value is strictly `false`
`expect(value).to.be.ok`	Checks if a value is truthy

Number Assertions

`expect(number).to.be.above(number2)`	Checks if `number` is greater than `number2`
`expect(number).to.be.below(number2)`	Checks if `number` is less than `number2`
`expect(number).to.be.at.least(number2)`	Checks if `number` is greater than or equal to `number2`
`expect(number).to.be.at.most(number2)`	Checks if `number` is less than or equal to `number2`
`expect(number).to.be.within(min, max)`	Checks if `number` is within the range `[min, max]`

Common Assertions with Assert

Equality Assertions

`assert.equal(actual, expected, message)`	Tests shallow, coercive equality with the equal comparison operator (==)
`assert.strictEqual(actual, expected, message)`	Tests strict equality (===)
`assert.deepEqual(actual, expected, message)`	Tests for deep equality
`assert.notEqual(actual, expected, message)`	Tests shallow, coercive inequality with the not equal comparison operator (!=)
`assert.notStrictEqual(actual, expected, message)`	Tests strict inequality (!==)
`assert.notDeepEqual(actual, expected, message)`	Tests for deep inequality

Type Assertions

`assert.isOk(value, message)`	Tests if a value is truthy
`assert.isNotOk(value, message)`	Tests if a value is falsy
`assert.isTrue(value, message)`	Tests if a value is strictly true
`assert.isFalse(value, message)`	Tests if a value is strictly false
`assert.isDefined(value, message)`	Tests if a value is not undefined
`assert.isUndefined(value, message)`	Tests if a value is undefined
`assert.isNull(value, message)`	Tests if a value is null
`assert.isNotNull(value, message)`	Tests if a value is not null
`assert.typeOf(value, type, message)`	Tests if the type of value is as expected

Number Assertions

`assert.isAbove(value, above, message)`	Tests if a value is greater than another value
`assert.isBelow(value, below, message)`	Tests if a value is less than another value
`assert.isAtLeast(value, atLeast, message)`	Tests if a value is greater than or equal to another value
`assert.isAtMost(value, atMost, message)`	Tests if a value is less than or equal to another value

Additional Chai Features

String Assertions

`expect(string).to.include(substring)`	Checks if `string` contains `substring`
`expect(string).to.contain(substring)`	Alias for `include`
`expect(string).to.match(regex)`	Checks if `string` matches the regular expression `regex`
`assert.include(haystack, needle, message)`	Asserts that `haystack` contains `needle`.
`assert.match(string, regexp, message)`	Asserts that `string` matches the regular expression `regexp`.

Array Assertions

`expect(array).to.include(value)`	Checks if `array` contains `value`
`expect(array).to.contain(value)`	Alias for `include`
`expect(array).to.have.lengthOf(number)`	Checks if `array` has a length of `number`
`assert.lengthOf(object, length, message)`	Asserts that object has expected length.
`assert.isArray(value, message)`	Asserts that value is an array.

Object Assertions

`expect(object).to.have.property(key)`	Checks if `object` has a property named `key`
`expect(object).to.have.property(key, value)`	Checks if `object` has a property named `key` with value `value`
`expect(object).to.have.nested.property(path)`	Checks if `object` has a nested property specified by `path`
`assert.property(object, property, message)`	Asserts that object has property.
`assert.deepProperty(object, property, message)`	Asserts that object has a deep property.
`assert.propertyVal(object, property, value, message)`	Asserts that object has property with expected value.