Jest Cheatsheet

A quick reference guide for Jest, a delightful JavaScript Testing Framework with a focus on simplicity.

Jest Basics

Installation & Setup

Installation:

`npm install --save-dev jest` or `yarn add --dev jest`

Package.json Configuration:

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

```
"scripts": {
  "test": "jest"
}
```

Running Tests:

`npm test` or `yarn test`

Writing Basic Tests

Test Structure:

```
test('description of the test', () => {
  // expectations
  expect(value).toBe(expectedValue);
});
```

`describe` Blocks:

Used to group related tests.

```
describe('My Component', () => {
  test('renders correctly', () => {
    //...
  });
});
```

`expect`:

The `expect` function is used to create an assertion. It takes a value as its argument, which is the value you want to test.

Matchers

`.toBe(value)`	Tests for exact equality using `===`.
`.toEqual(value)`	Tests for deep equality (for objects and arrays).
`.toBeNull()`	Matches only `null`.
`.toBeUndefined()`	Matches only `undefined`.
`.toBeDefined()`	The opposite of `.toBeUndefined()`.
`.toBeTruthy()`	Matches anything that an `if` statement treats as true.
`.toBeFalsy()`	Matches anything that an `if` statement treats as false.
`.toBeGreaterThan(number)`	Tests if value is greater than number.

Asynchronous Testing

Promises

Testing Promises:

Return the promise from your test. Jest will wait for the promise to resolve.

```
test('fetches data', () => {
  return fetchData().then(data => {
    expect(data).toBe('peanut butter');
  });
});
```

`.resolves` and `.rejects` Matchers:

```
test('fetches data', () => {
  return
expect(fetchData()).resolves.toBe('peanut
butter');
});

test('fails with an error', () => {
  return
expect(fetchData()).rejects.toMatch('error');
});
```

Async/Await

Using `async/await`:

```
test('fetches data', async () => {
  const data = await fetchData();
  expect(data).toBe('peanut butter');
});

test('fails with an error', async () => {
  await
expect(fetchData()).rejects.toMatch('error');
});
```

`.rejects` example:

```
test('the fetch fails with an error', async ()
=> {
  await
expect(fetchData()).rejects.toThrow('error');
});
```

Callbacks

Using `.done`:

If you're using callbacks, Jest provides a `done` callback.

```
test('data fetched asynchronously', (done) =>
{
  fetchData((data) => {
    expect(data).toBe('peanut butter');
    done();
  });
});
```

Mocking

Mock Functions

Creating Mock Functions:

```
const myMock = jest.fn();

myMock.mockReturnValue('default');

console.log(myMock()); // -> 'default'
```

Mocking Modules:

```
jest.mock('./moduleToMock');
import myModule from './moduleToMock';

myModule.mockImplementation(() => {
  return 'mocked value';
});
```

Mocking Implementations

`.mockImplementation(fn)`:

Define a custom implementation for the mock function.

```
const myMockFn = jest
  .fn()
  .mockImplementation(scalar => 42 + scalar);

myMockFn(1); // -> 43
```

`.mockReturnValue(value)`:

Define a default return value for the mock.

```
const myMockFn = jest
  .fn()
  .mockReturnValue('result');

myMockFn(); // -> 'result'
```

Mocking Return Values

`.mockResolvedValue(value)`:

Mocks a promise that resolves to the specified value. Helpful for async functions.

```
const asyncMock =
jest.fn().mockResolvedValue('async data');

asyncMock(); // -> Promise that resolves to
'async data'
```

`.mockRejectedValue(value)`:

Mocks a promise that rejects with the specified value.

```
const asyncMock =
jest.fn().mockRejectedValue(new Error('Async
error'));

asyncMock(); // -> Promise that rejects with
an error
```

Snapshot Testing

Basics

What are Snapshots?

Snapshots capture the rendered output of a component at a specific point in time. Jest compares the current output with the stored snapshot to detect unexpected changes.

`.toMatchSnapshot()`:

```
test('renders correctly', () => {
  const tree = render(<MyComponent />);
  expect(tree).toMatchSnapshot();
});
```

Updating Snapshots:

If a snapshot test fails due to an intentional change, you need to update the snapshot.

`jest --updateSnapshot` or `jest -u`

Inline Snapshots

`.toMatchInlineSnapshot()`:

Stores the snapshot directly in the test file.

```
test('renders correctly', () => {
  const tree = render(<MyComponent />);
  expect(tree).toMatchInlineSnapshot(`
    <div>
      My Component Content
    </div>
  `);
});
```

Configuration

Snapshot Directory:

Snapshots are typically stored in a `__snapshots__` directory next to the test file.

Ignoring changes:

It is possible to ignore certain parts of a snapshot.