Basic test

```
describe('makePoniesPink', () => {
  test('make each pony pink', () => {
    const actual = fn(['Alice', 'Bob', 'Eve'])
    expect(actual).toEqual(['Pink Alice', 'Pink Bob', 'Pink Eve'])
  })
})
```

Matchers

```
expect.assertions(28) // Exactly 28 assertions are called during a test
expect.hasAssertions() // At least one assertion is called during a test

expect(42).toBe(42) // ===
expect(42).not.toBe(3) // !==
expect([1, 2]).toEqual([1, 2]) // Deep equality
expect('').toBeFalsy() // false, 0, '', null, undefined, NaN
expect('foo').toBeTruthy() // Not false, 0, '', null, undefined, NaN
expect(null).toBeNull() // === null
expect(undefined).toBeUndefined() // === undefined
expect(7).toBeDefined() // !== undefined

expect('long string').toMatch('str')
expect('coffee').toMatch(/ff/)
expect('pizza').not.toMatch('coffee')
expect(['pizza', 'coffee']).toEqual([expect.stringContaining('zz'), expect.stringMa

expect(new A()).toBeInstanceOf(A)
expect(() => {}).toEqual(expect.any(Function))
expect('pizza').toEqual(expect.anything())

expect({ a: 1 }).toHaveProperty('a')
expect({ a: 1 }).toHaveProperty('a', 1)
expect({ a: { b: 1 } }).toHaveProperty('a.b')
expect({ a: 1, b: 2 }).toMatchObject({ a: 1 })
expect({ a: 1, b: 2 }).toMatchObject({
  a: expect.any(Number),
  b: expect.any(Number)
})
expect([{ a: 1 }, { b: 2 }]).toEqual([
  expect.objectContaining({ a: expect.any(Number) }),
  expect.anything()
])

expect(2).toBeGreaterThan(1)
expect(1).toBeGreaterThanOrEqual(1)
expect(1).toBeLessThan(2)
expect(1).toBeLessThanOrEqual(1)
expect(0.2 + 0.1).toBeCloseTo(0.3, 5)

expect(['Alice', 'Bob', 'Eve']).toHaveLength(3)
expect(['Alice', 'Bob', 'Eve']).toContain('Alice')
expect([{ a: 1 }, { a: 2 }]).toContainEqual({ a: 1 })
expect(['Alice', 'Bob', 'Eve']).toEqual(expect.arrayContaining(['Alice', 'Bob']))

// const fn = () => { throw new Error('Out of cheese!') }
expect(fn).toThrow()
expect(fn).toThrow('Out of cheese')
expect(fn).toThrowErrorMatchingSnapshot()

expect(node).toMatchSnapshot()

// const fn = jest.fn()
// const fn = jest.fn().mockName('Unicorn') -- named mock, Jest 22+
expect(fn).toBeCalled() // Function was called
expect(fn).not.toBeCalled() // Function was *not* called
expect(fn).toHaveBeenCalledTimes(1) // Function was called only once
expect(fn).toBeCalledWith('first arg', 'second arg') // Any of calls was with these
expect(fn).toHaveBeenLastCalledWith('first arg', 'secon arg') // Last call was with
expect(fn.mock.calls).toEqual([['first', 'call', 'args'], ['second', 'call', 'args'
expect(fn.mock.calls[0][0](1)).toBe(2) // fn.mock.calls[0][0] — the first argument
```

Matchers docs

Aliases

- `toBeCalled` → `toHaveBeenCalled`
- `toBeCalledWith` → `toHaveBeenCalledWith`
- `lastCalledWith` → `toHaveBeenLastCalledWith`
- `toThrowError` → `toThrow`

Promise matchers (Jest 20+)

```
test('resolve to lemon', () => {
  expect.assertions(1)
  // Make sure to add a return statement
  return expect(Promise.resolve('lemon')).resolves.toBe('lemon')
  // return expect(Promise.reject('octopus')).rejects.toBeDefined();
})
```

Or with async/await:

```
test('resolve to lemon', async () => {
  expect.assertions(2)
  await expect(Promise.resolve('lemon')).resolves.toBe('lemon')
  await expect(Promise.resolve('lemon')).resolves.not.toBe('octopus')
})
```

resolves docs

Async tests

See more examples in Jest docs.

async/await

```
test('async test', async () => {
  expect.assertions(1)
  const result = await runAsyncOperation()
  expect(result).toBe(true)
})
```

Promises

Return a Promise from your test:

```
test('async test', () => {
  expect.assertions(1)
  return runAsyncOperation().then(result => {
    expect(result).toBe(true)
  })
})
```

done() callback

Wrap your assertions in try/catch block, otherwise Jest will ignore failures:

```
test('async test', done => {
  expect.assertions(1)
  runAsyncOperation()
  setTimeout(() => {
    try {
      const result = getAsyncOperationResult()
      expect(result).toBe(true)
      done()
    } catch (err) {
      done.fail(err)
    }
  })
})
```

Mocks

Mock functions

```
test('call the callback', () => {
  const callback = jest.fn()
  fn(callback)
  expect(callback).toBeCalled()
  expect(callback.mock.calls[0][1].baz).toBe('pizza') // Second argument of the fir
})
```

You can also use snapshots:

```
test('call the callback', () => {
  const callback = jest.fn().mockName('Unicorn') // mockName is available in Jest 2
  fn(callback)
  expect(callback).toMatchSnapshot()
  // ->
  // [MockFunction Unicorn] {
  //   "calls": Array [
  // ...
})
```

And pass an implementation to `jest.fn` function:

```
const callback = jest.fn(() => true)
```

Mock functions docs

Mock modules using `jest.mock` method

```
jest.mock('lodash/memoize', () => a => a) // The original lodash/memoize should exi
jest.mock('lodash/memoize', () => a => a, { virtual: true }) // The original lodash
```

jest.mock docs

> Note: When using `babel-jest`, calls to `jest.mock` will automatically be hoisted to the top of the code block. Use `jest.doMock` if you want to explicitly avoid this behavior.

Mock modules using a mock file

1. Create a file like `__mocks__/lodash/memoize.js`:

   ```
   module.exports = a => a
   ```

2. Add to your test:

   ```
   jest.mock('lodash/memoize')
   ```

> Note: When using `babel-jest`, calls to `jest.mock` will automatically be hoisted to the top of the code block. Use `jest.doMock` if you want to explicitly avoid this behavior.

Manual mocks docs

Mock object methods

```
const spy = jest.spyOn(ajax, 'request').mockImplementation(() => Promise.resolve({
expect(spy).toHaveBeenCalled()
spy.mockRestore()
```

Mock getters and setters (Jest 22.1.0+)

```
const location = {}
const getTitle = jest.spyOn(location, 'title', 'get').mockImplementation(() => 'piz
const setTitle = jest.spyOn(location, 'title', 'set').mockImplementation(() => {})
```

Mock getters and setters

```
const getTitle = jest.fn(() => 'pizza')
const setTitle = jest.fn()
const location = {}
Object.defineProperty(location, 'title', {
  get: getTitle,
  set: setTitle
})
```

Clearing and restoring mocks

```
fn.mockClear() // Clear number of calls
fn.mockRestore() // Remove the mock and restore the initial implementation
```

> Note: `mockRestore` works only with mocks created by `jest.spyOn`.

Accessing the original module when using mocks

```
jest.mock('fs')
const fs = require('fs') // Mocked module
const fs = require.requireActual('fs') // Original module
```

Testing modules with side effects

Node.js and Jest will cache modules you `require`. To test modules with side effects you'll need to reset the module registry between tests:

```
const modulePath = '../module-to-test'

afterEach(() => {
  jest.resetModules()
})

test('first test', () => {
  // Prepare conditions for the first test
  const result = require(modulePath)
  expect(result).toMatchSnapshot()
})

test('second text', () => {
  // Prepare conditions for the second test
  const fn = () => require(modulePath)
  expect(fn).toThrow()
})
```

Usage with Babel and TypeScript

Add babel-jest or ts-jest. Check their docs for installation instructions.