Created by: Samprada N, References taken from product docs of Jest

JEST PRACTICAL CHEATSHEET

Ref: Using Matchers · Jest (jestjs.io)

In Jest, we can use the following ways of creating test suits and test cases

1. describe('Test Suite Title', ()=>{})
2. To add a test case within the above test suite
   a. It('Test Case Title', ()=>{})
   b. test('Test Case Title', ()=> {})
3. A Test Suite with a Testcase
   a. describe('Test SUITE title', ()=>{

      it('Test CASE title', ()=>{

      …

      })

   })

===

1. A simple test case in JEST
   a.
```
test('two plus two is four', () => {
  expect(2 + 2).toBe(4);
});
```

2.
```
test('object assignment', () => {
  const data = {one: 1};
  data['two'] = 2;
  expect(data).toEqual({one: 1, two: 2});
});
```

3. Using not.toBe

   a.
```
test('adding positive numbers is not zero', () => {
  for (let a = 1; a < 10; a++) {
    for (let b = 1; b < 10; b++) {
      expect(a + b).not.toBe(0);
    }
  }
});
```

4. Working with null, undefined, true, false
   a.
```
test('null', () => {
  const n = null;
  expect(n).toBeNull();
  expect(n).toBeDefined();
  expect(n).not.toBeUndefined();
  expect(n).not.toBeTruthy();
  expect(n).toBeFalsy();
});

test('zero', () => {
  const z = 0;
  expect(z).not.toBeNull();
  expect(z).toBeDefined();
  expect(z).not.toBeUndefined();
  expect(z).not.toBeTruthy();
  expect(z).toBeFalsy();
});
```

5. Number comparisons
    a.
```
test('two plus two', () => {
  const value = 2 + 2;
  expect(value).toBeGreaterThan(3);
  expect(value).toBeGreaterThanOrEqual(3.5);
  expect(value).toBeLessThan(5);
  expect(value).toBeLessThanOrEqual(4.5);

  // toBe and toEqual are equivalent for numbers
  expect(value).toBe(4);
  expect(value).toEqual(4);
});
```
    b.
```
test('adding floating point numbers', () => {
  const value = 0.1 + 0.2;
  //expect(value).toBe(0.3);           This won't work because
of rounding error
  expect(value).toBeCloseTo(0.3); // This works.
});
```

6. Testing Strings
    a.
```
test('there is no I in team', () => {
  expect('team').not.toMatch(/I/);
});

test('but there is a "stop" in Christoph', () => {
  expect('Christoph').toMatch(/stop/);
});
```

7. Working with Arrays
    a.
```
const shoppingList = [
  'diapers',
  'kleenex',
  'trash bags',
  'paper towels',
  'milk',
];
```

```
test('the shopping list has milk on it', () => {
  expect(shoppingList).toContain('milk');
  expect(new Set(shoppingList)).toContain('milk');
});
```

8. Working with Exceptions
    a.
    ```
    function compileAndroidCode() {
      throw new Error('you are using the wrong JDK');
    }

    test('compiling android goes as expected', () => {
      expect(() => compileAndroidCode()).toThrow();
      expect(() => compileAndroidCode()).toThrow(Error);

      // You can also use the exact error message or a regexp
      expect(() => compileAndroidCode()).toThrow('you are using
    the wrong JDK');
      expect(() => compileAndroidCode()).toThrow(/JDK/);
    });
    ```

9. Working with setup & tear-down. beforeEach(), AfterEach() will be called for the initial setup & tear-down respectively
    a.
    ```
    beforeEach(() => {
      initializeCityDatabase();
    });

    afterEach(() => {
      clearCityDatabase();
    });

    test('city database has Vienna', () => {
      expect(isCity('Vienna')).toBeTruthy();
    });

    test('city database has San Juan', () => {
      expect(isCity('San Juan')).toBeTruthy();
    });
    ```

10. BeforeAll(), AfterAll() hooks shall be called only once viz. at the beginning before executing all the test cases and after the execution of all test cases respectively.
    a.
    ```
    beforeAll(() => {
      return initializeCityDatabase();
    });

    afterAll(() => {
      return clearCityDatabase();
    });
    ```

```
test('city database has Vienna', () => {
  expect(isCity('Vienna')).toBeTruthy();
});

test('city database has San Juan', () => {
  expect(isCity('San Juan')).toBeTruthy();
});
```