



JS Unit testing Techniques

JavaScript unit testing
with



Why Use Jasmine?

- Jasmine does not depend on any other JavaScript framework.
- Jasmine does not require any DOM.
- All the syntax used in Jasmine framework is clean and obvious.
- Jasmine is an open-source framework and easily available in different versions like stand-alone, ruby gem, Node.js, etc.

Suite Block

- **Suite** is the basic building block of Jasmine framework. The collection of similar type test cases written for a specific file or function is known as one suite. It contains two other blocks, one is "**Describe()**" and another one is "**It()**".
- One Suite block can have only two parameters, one "**name of that suite**" and another "**Function declaration**" that actually makes a call to our unit functionality that is to be tested.

Suite Block

```
describe("add", () =>{  
  it('should add', ()=>{  
    expect(add(7,3)).toEqual(10)  
  });  
  it('should return 0', ()=>{  
    expect(add(2,2)-4).not.toEqual(2)  
  })  
})
```

JasmineJS - Matchers

Matchers are the JavaScript function that does a Boolean comparison between an actual output and an expected output

- **toEqual()** is the inbuilt matcher which will compare the result of the **add()** method with the arguments passed to **toEqual()** matchers

```
it('should add', ()=>{  
    expect(add(7,3)).toEqual(10)  
});
```

JasmineJS - Skip Block

Jasmine also allows the developers to skip one or more than one test cases. These techniques can be applied at the **Spec level** or the **Suite level**.

- Skipping Spec

```
it('should add', ()=>{  
    expect(add(7,3)).toEqual(10)  
});  
xit('should return 0', ()=>{  
    expect(add(2,2)-4).not.toEqual(2)  
})
```

- Skipping suit

```
xdescribe("add", () =>{  
    it('should add', ()=>{  
        expect(add(7,3)).toEqual(10)  
    });  
    it('should return 0', ()=>{  
        expect(add(2,2)-4).not.toEqual(2)  
    })  
})
```

JasmineJS - Equality Check

Jasmine provides plenty of methods which help us check the equality of any JavaScript function and file

- **ToEqual()** is the simplest matcher present in the inbuilt library of Jasmine. It just matches whether the result of the operation given as an argument to this method matches with the result of it or not.
- **not.toEqual()** works exactly opposite to toEqual(). **not.toEqual()** is used when we need to check if the value does not match with the output of any function.

- **toBe()** matcher works in a similar way as **toEqual()**, however they are technically different from each other. **toBe()** matcher matches with the type of the object whereas **toEqual()** matches with the equivalency of the result.
- **not.toBe()** As seen earlier, **not** is nothing but a negation of the **toBe()** method. It fails when the expected result matches with the actual output of the function or JavaScript file.

```
describe("Different Methods of Expect Block",function () {  
    it("The Example of not.toBe() method",function () {  
        expect(true).not.toBe(false);  
    });  
});
```

JasmineJS - Boolean Check

Jasmine provides some methods to check Boolean conditions too. Following are the methods that help us check Boolean conditions.

- **ToBeTruthy()** This Boolean matcher is used in Jasmine to check whether the result is equal to true or false
- **toBeFalsy()** also works the same way as toBeTruthy() method. It matches the output to be false whereas toBeTruthy matches the output to be true.

```
window.expectexam = {  
  exampleoftrueFalse: function (num) {  
    if(num < 10)  
      return true;  
    else  
      return false;  
  },  
};
```

```
describe("Different Methods of Expect Block",function () {  
  it("The Example of toBeTruthy() method",function () {  
    expect(expectexam.exampleoftrueFalse(5)).toBeTruthy();  
  });  
});
```

- **toContain()** matchers provide us the facility to check whether any element is a part of the same array or some other sequential objects

```
describe("Different Methods of Expect Block",function () {  
    it("The Example of toContain() method",function () {  
        expect([1,2, 3, 4]).toContain(3);  
    });  
});
```

- **ToMatch()** matcher works on String type variable. It is helpful to find whether a specific String is present in the expected output or not.

```
describe("Different Methods of Expect Block",function () {  
    it("Example of toMatch()", function () {  
        expect("Jasmine tutorial in tutorials.com").toMatch(/com/);  
    });  
});
```

- **ToBeDefined()** This matcher is used to check whether any variable in the code is predefined or not.

```
currentVal = 0;

describe("Different Methods of Expect Block",function () {
  it("Example of toBeDefined", function () {
    expect(currentVal).toBeDefined();
  });
});
```

- **ToBeUndefined()** This matcher helps to check whether any variable is previously undefined or not, basically it works simply opposite to the previous matcher that is toBeDefined.

```
describe("Different Methods of Expect Block",function () {  
    it("Example of toBeUndefined()", function () {  
        var undefineValue;  
        expect(undefineValue).toBeUndefined();  
    });  
});
```

- **toBeNull()** As the name signifies this matcher helps to check null values.

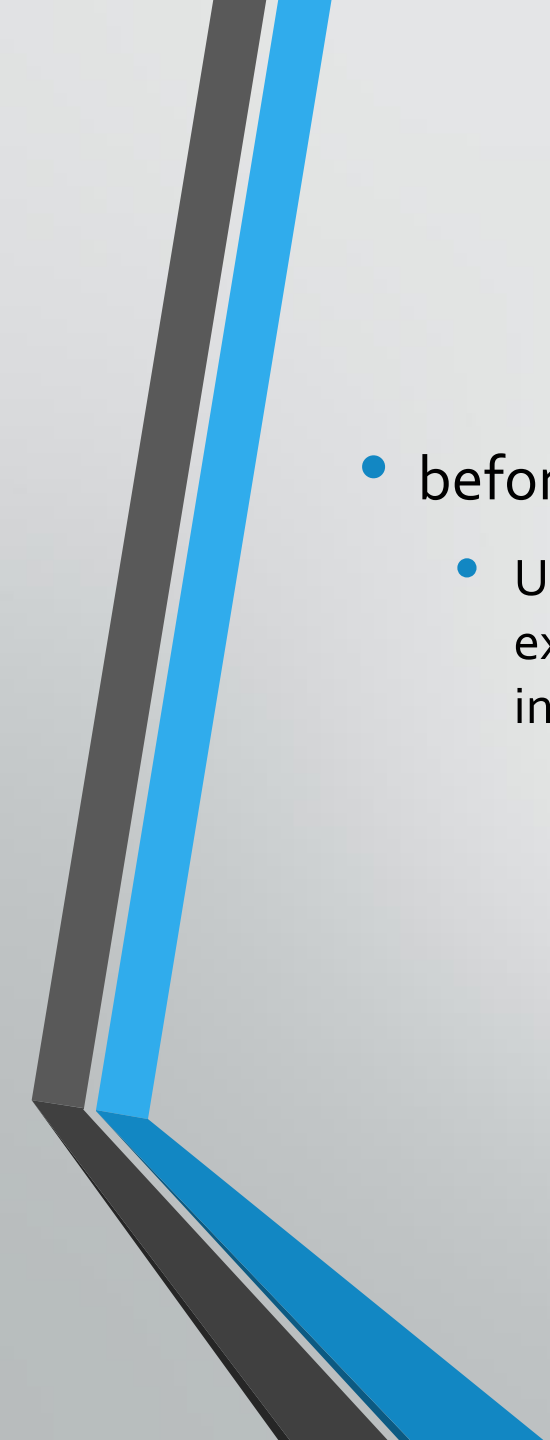
```
describe("Different Methods of Expect Block",function () {  
    var value = null;  
  
    it("Example of toBeNull()", function () {  
        expect(value).toBeNull();  
    });  
});
```


- **ToBeGreaterThan()** As the name suggests this matcher helps to check greater than condition
- **ToBeLessThan()** This matcher helps to check the less than condition of the test scenario. It behaves exactly opposite to that of toBeGreaterThan() matcher

```
describe("Different Methodsof Expect Block",function () {  
    var exp = 4;  
  
    it("Example of toBeLessThan()", function() {  
        expect(exp).toBeLessThan(5);  
    });  
});
```

- toBeNaN().

```
describe("Different Methods of Expect Block",function () {  
  it("Example of toBeNaN()", function () {  
    expect(0 / 0).toBeNaN();  
  });  
});
```

- 
- beforeEach(), afterEach()
 - Using these two functionalities, we can execute some pieces of code before and after execution of each spec. This functionality is very useful for running the common code in the application.

```
var currentVal = 0;
beforeEach(function() {
    currentVal = 5;
});
describe("Different Methods of Expect Block",function() {
    it("after each function ", function() {
        expect(currentVal).toEqual(5);
    });
});
```

```
var currentVal = 0;
afterEach(function() {
    currentVal = 5;
});
describe("Different Methods of Expect Block",function() {
    it("first call ", function() {
        expect(currentVal).toEqual(0);
    });

    it("second call ", function() {
        expect(currentVal).toEqual(5);
    });
});
```

