Spying on your code

- toHaveBeenCalled():

Expect the actual (a Spy) to have been called.

- toHaveBeenCalledWith():

Expect the actual (a Spy) to have been called with particular arguments at least once.

- toHaveBeenCalledTimes():

Expect the actual (a Spy) to have been called the specified number of times.

```
describe('test.js', function() {
   describe('someMethod()', function() {
      it('calls spy', function() {
         spyOn(calc, 'add').and.stub();
         const result = calc.sum('2+3');

         expect(calc.add).toHaveBeenCalled();
         expect(calc.add).toHaveBeenCalledWith('2');
         expect(calc.add).toHaveBeenCalledTimes(2);
      });
   });
});
```

Spying on your code

- callThrough():

Tell the spy to call through to the real implementation when invoked.

```
describe('test.js', function() {
   describe('someMethod()', function() {
      it('calls spy', function() {
         spyOn(calc, 'add').and.callThrough();
         const result = calc.sum('2+3');

         expect(calc.add).toHaveBeenCalled();
         expect(result).toEqual(5);
      });
   });
});
```

Spying on your code

- callFake():

Tell the spy to call a fake implementation when invoked.

```
describe('test.js', function() {
   describe('someMethod()', function() {
      it('calls spy', function() {
         spyOn(calc, 'add').and.callFake(function() {
            return 9;
         });
         const result = calc.sum('2+3');

         expect(calc.add).toHaveBeenCalled();
         expect(result).toEqual(9);
      });
   });
});
```

Spying on your code

- returnValue(Value):

Tell the spy to return the value when invoked.

```
describe('test.js', function() {
   describe('someMethod()', function() {
      it('calls spy', function() {
         spyOn(calc, 'add').and.returnValue(3);
         const result = calc.sum('1+1');

        expect(calc.add).toHaveBeenCalled();
        expect(result).toEqual(3);
      });
   });
});
```

Spying on your code

- returnValues(...Values):

Tell the spy to return one of the specified values (sequentially) each time the spy is invoked.

```
describe('test.js', function() {
   describe('someMethod()', function() {
      it('calls spy', function() {
         spyOn(calc, 'add').and.returnValues(3,4);
         calc.sum('1+1');

         const result = calc.sum('1+1');

        expect(calc.add).toHaveBeenCalled();
        expect(result).toEqual(4);
      });
   });
});
```

Spying on your code

- throwError()

Tell the spy to throw an error when invoked.

```
describe('test.js', function() {
   describe('someMethod()', function() {
      it('calls spy', function() {
         spyOn(calc, 'add').and.throwError(
           'Error'
         );

        expect(function() {
           calc.sum('1+1')
         }).toThrowError('Error');
      });
   });
});
```

Spying on your code

- spyOnProperty()

Install a spy on a property installed with Object.defineProperty onto an existing object.

```
describe('test.js', function() {
   describe('someMethod()', function() {
      it('calls spy', function() {
         const spy = spyOnProperty(calc, total, 'get');

         calc.sum('1+2');

        expect(spy).toHaveBeenCalled();
      });
   });
});
```