Angular (Part – 6)

PROF. P. M. JADAV
ASSOCIATE PROFESSOR
COMPUTER ENGINEERING DEPARTMENT
FACULTY OF TECHNOLOGY
DHARMSINH DESAI UNIVERSITY, NADIAD

Content

- Testing

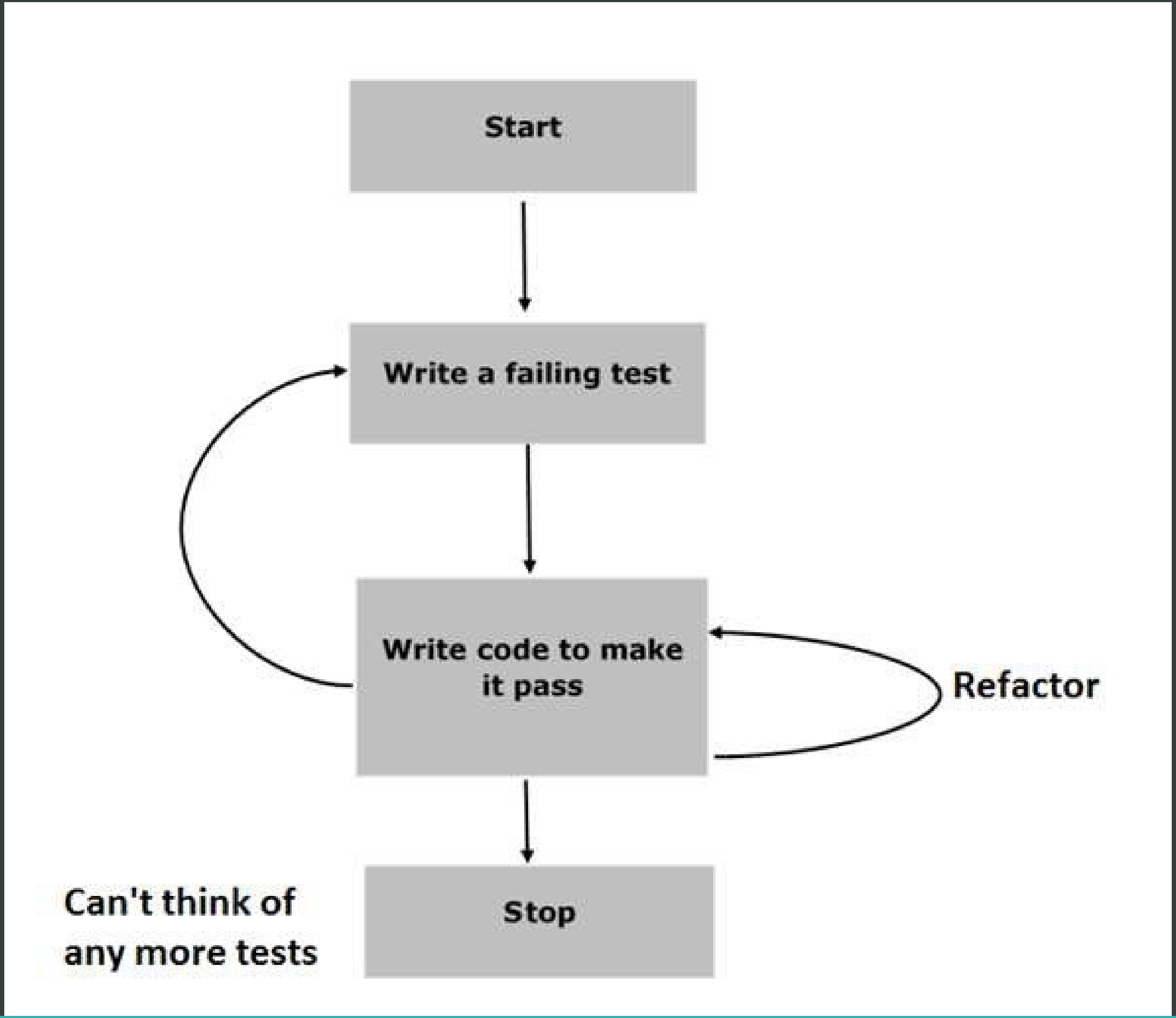
Testing Toolchain

- Jasmine
- Karma

Jasmine

- JavaScript testing framework
- supports Behaviour Driven Development (BDD)
- it is a specific flavour of Test Driven Development (TDD)

Behaviour Driven Development



Behaviour Driven Development

1. **Start**: make our environment ready for Jasmine application
2. **Write a failing test**: write our first ever test case. It is obvious that this test is going to fail because there is no such file or function to be tested.
3. **Write a code to make it pass**: prepare our JavaScript file or function that needs to be tested and make sure that all the test cases we had prepared in the early stage will be successful
4. **Refactor**: we need to prepare as many test cases as we can
5. **Stop**: If everything is going well then your application must be ready and up

Karma

- Karma is a test automation tool for controlling the execution of our tests and what browser to perform them under.
- It also allows us to generate various reports on the results.

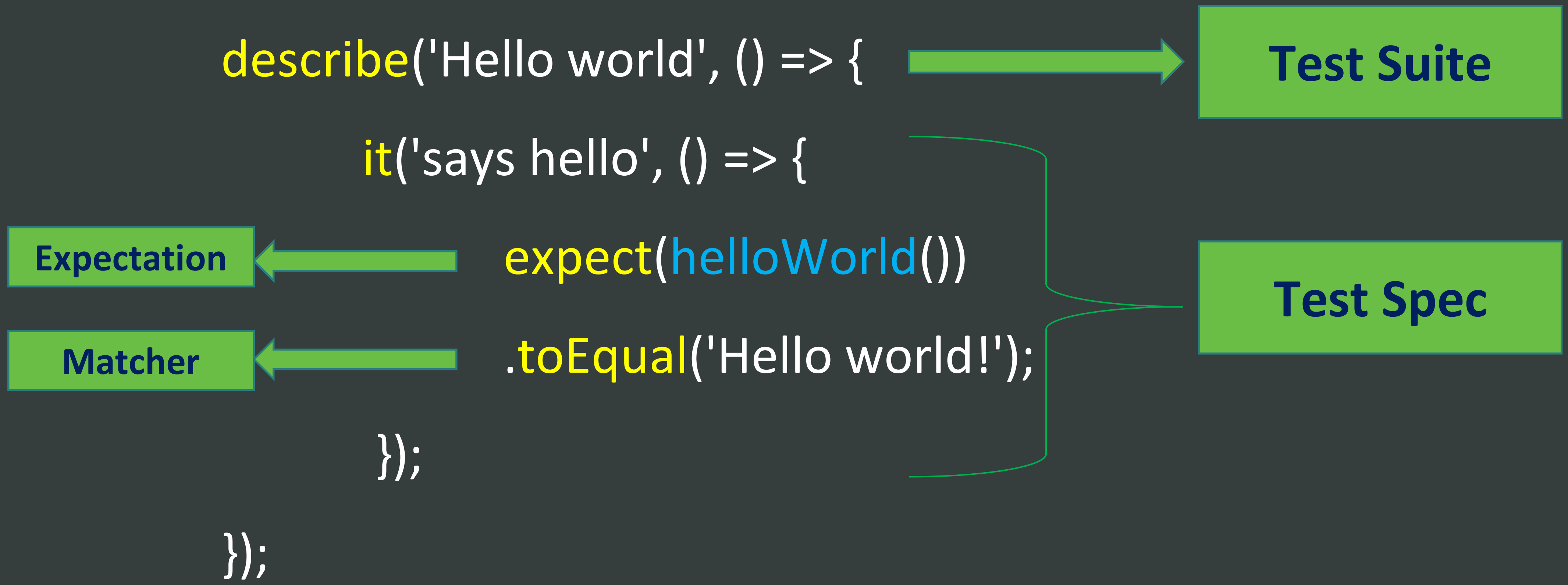
Jasmine

Behavior-Driven JavaScript

Testing in Jasmine

```
function helloWorld() {
        return 'Hello world!';
}
```

Testing in Jasmine



```
describe('Hello world', () => {
    it('says hello', () => {
        expect(helloWorld())
        .toEqual('Hello world!');
    });
});
```

Built-in Matchers

1. expect(array).toContain(member);
2. expect(fn).toThrow(string);
3. expect(fn).toThrowError(string);
4. expect(instance).toBe(instance);
5. expect(mixed).toBeDefined();
6. expect(mixed).toBeFalsy();
7. expect(mixed).toBeNull();
8. expect(mixed).toBeTruthy();

Built-in Matchers

9. expect(mixed).toBeUndefined();
10. expect(mixed).toEqual(mixed);
11. expect(mixed).toMatch(pattern);
12. expect(number).toBeCloseTo(number, decimalPlaces);
13. expect(number).toBeGreaterThan(number);

Built-in Matchers

14. expect(number).toBeLessThan(number);
15. expect(number).toBeNaN();
16. expect(spy).toHaveBeenCalled();
17. expect(spy).toHaveBeenCalledTimes(number);
18. expect(spy).toHaveBeenCalledWith(...arguments);

Testing in Angular

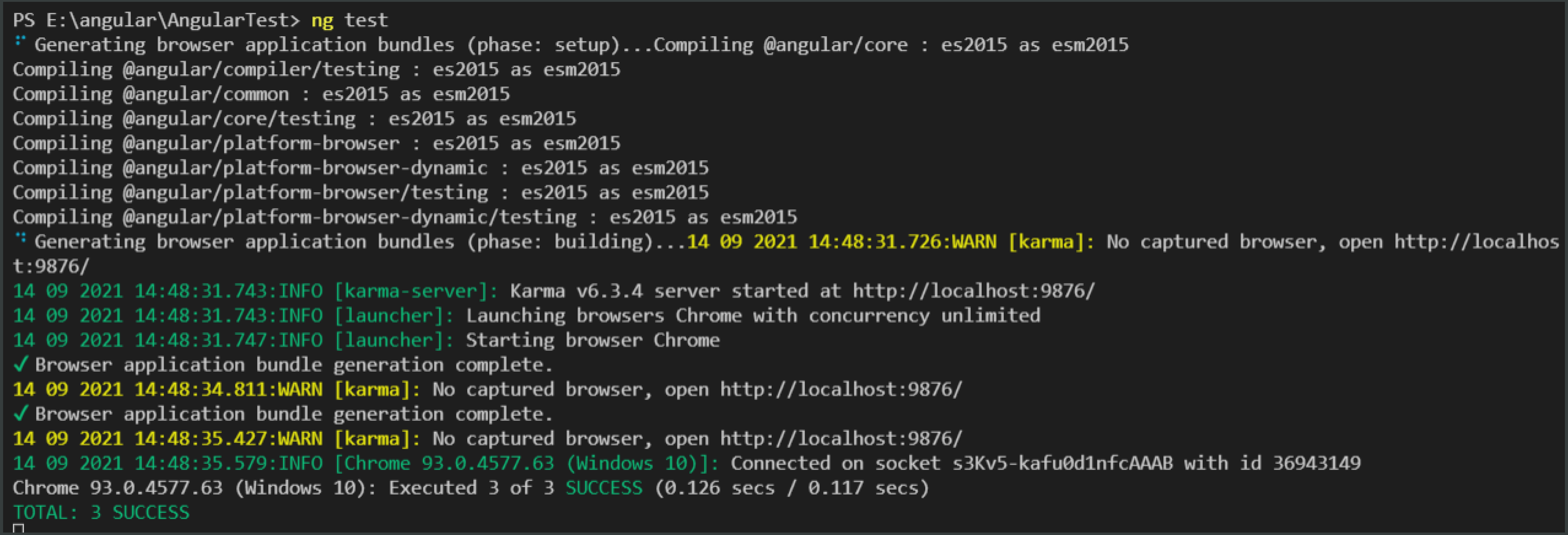
1. Create a new Angular project (without routing features)

```
ng new AngularTest
```

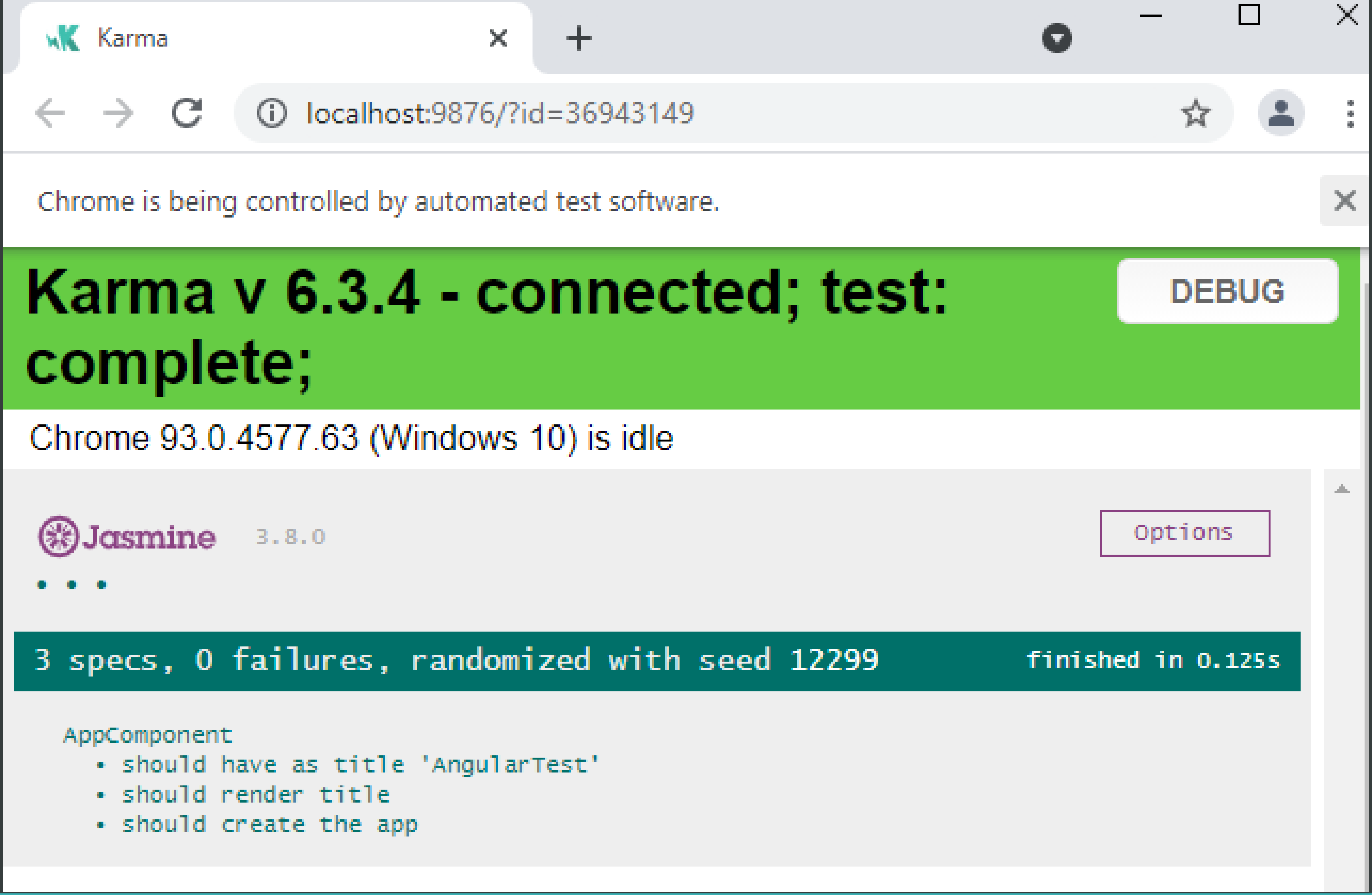
2. Run the following command to test the project files:

```
ng test
```

Output of "ng test" command



```
PS E:\angular\AngularTest> ng test
⠋ Generating browser application bundles (phase: setup)...Compiling @angular/core : es2015 as esm2015
Compiling @angular/compiler/testing : es2015 as esm2015
Compiling @angular/common : es2015 as esm2015
Compiling @angular/core/testing : es2015 as esm2015
Compiling @angular/platform-browser : es2015 as esm2015
Compiling @angular/platform-browser-dynamic : es2015 as esm2015
Compiling @angular/platform-browser/testing : es2015 as esm2015
Compiling @angular/platform-browser-dynamic/testing : es2015 as esm2015
⠙ Generating browser application bundles (phase: building)...14 09 2021 14:48:31.726:WARN [karma]: No captured browser, open http://localhos
t:9876/
14 09 2021 14:48:31.743:INFO [karma-server]: Karma v6.3.4 server started at http://localhost:9876/
14 09 2021 14:48:31.743:INFO [launcher]: Launching browsers Chrome with concurrency unlimited
14 09 2021 14:48:31.747:INFO [launcher]: Starting browser Chrome
✓ Browser application bundle generation complete.
14 09 2021 14:48:34.811:WARN [karma]: No captured browser, open http://localhost:9876/
✓ Browser application bundle generation complete.
14 09 2021 14:48:35.427:WARN [karma]: No captured browser, open http://localhost:9876/
14 09 2021 14:48:35.579:INFO [Chrome 93.0.4577.63 (Windows 10)]: Connected on socket s3Kv5-kafu0d1nfcAAAB with id 36943149
Chrome 93.0.4577.63 (Windows 10): Executed 3 of 3 SUCCESS (0.126 secs / 0.117 secs)
TOTAL: 3 SUCCESS
```


Karma
localhost:9876/?id=36943149
Chrome is being controlled by automated test software.
DEBUG
Jasmine 3.8.0
Options

```
Karma v 6.3.4 - connected; test: complete;
Chrome 93.0.4577.63 (Windows 10) is idle

• • •
3 specs, 0 failures, randomized with seed 12299        finished in 0.125s

AppComponent
  • should have as title 'AngularTest'
  • should render title
  • should create the app
```

Testing a function

Create a file "hello.ts" in /src/app folder.

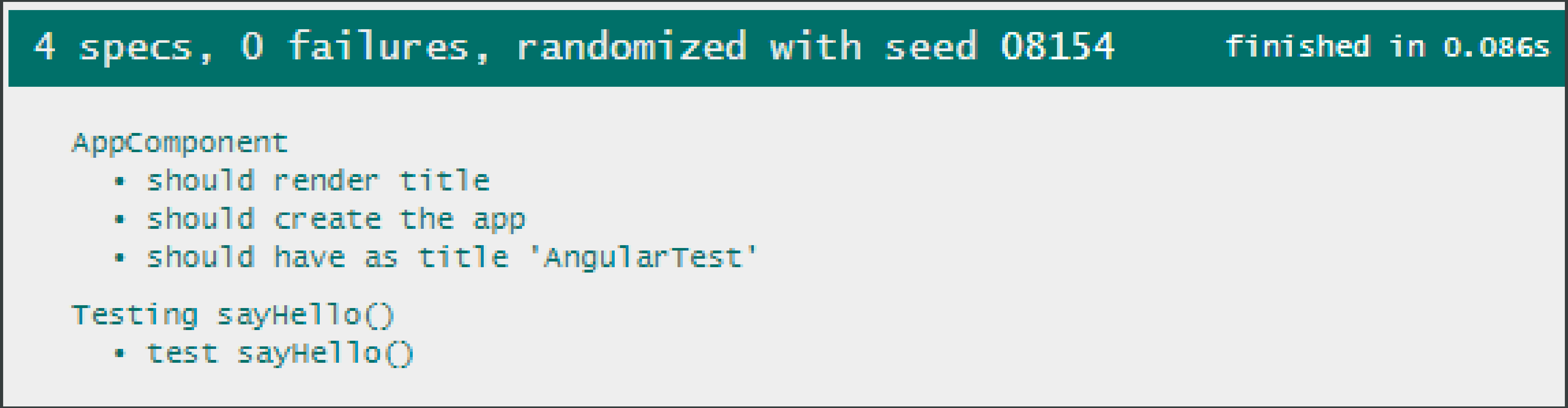
```
export function sayHello()
{
    return "Hello World!"
}
```

Testing a function

Create a unit test file "hello.spec.ts" in /src/app folder.

```
import { sayHello } from "./hello"

describe('Testing sayHello()', () => {
    it('test sayHello()', () => {
        expect(sayHello()).toBe("Hello World!")
    })
})
```

Testing a function (output)



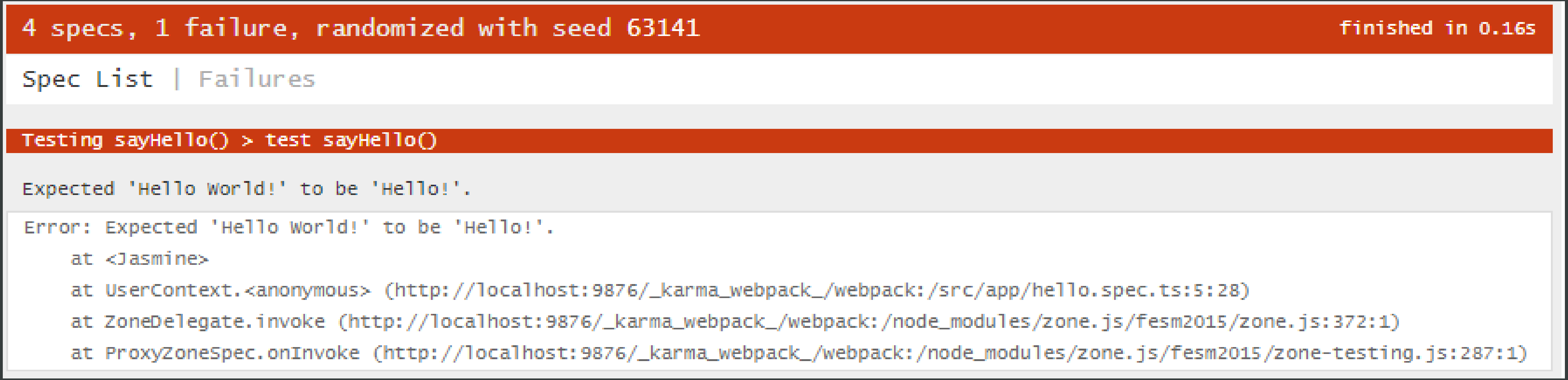
```
4 specs, 0 failures, randomized with seed 08154          finished in 0.086s

  AppComponent
    • should render title
    • should create the app
    • should have as title 'AngularTest'

  Testing sayHello()
    • test sayHello()
```

Testing a function (failed test)

Modify the file "hello.spec.ts".

```
describe('Testing sayHello()', () => {

    it('test sayHello()', () => {
        expect(sayHello()).toBe("Hello!")
    })
})
```

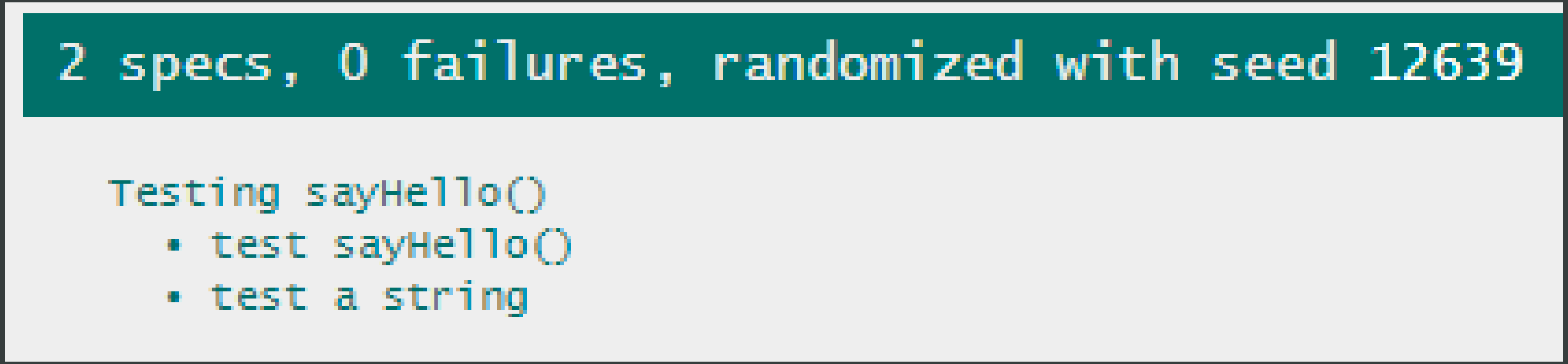
Testing a function (output)



```
4 specs, 1 failure, randomized with seed 63141                                          finished in 0.16s

Spec List | Failures

Testing sayHello() > test sayHello()

Expected 'Hello World!' to be 'Hello!'.

Error: Expected 'Hello World!' to be 'Hello!'.
    at <Jasmine>
    at UserContext.<anonymous> (http://localhost:9876/_karma_webpack_/webpack:/src/app/hello.spec.ts:5:28)
    at ZoneDelegate.invoke (http://localhost:9876/_karma_webpack_/webpack:/node_modules/zone.js/fesm2015/zone.js:372:1)
    at ProxyZoneSpec.onInvoke (http://localhost:9876/_karma_webpack_/webpack:/node_modules/zone.js/fesm2015/zone-testing.js:287:1)
```

Testing a String

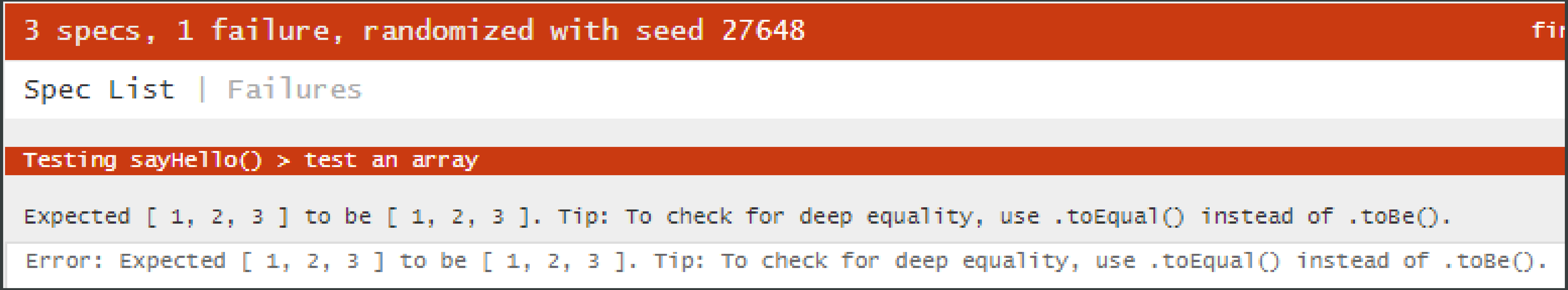
```
describe('Testing sayHello()', () => {
    it('test sayHello()', () => {
        expect(sayHello()).toBe("Hello!")
    })
    it('test a string', () => {
        expect("Hello").toBe("Hello")
    })
})
```



```
2 specs, 0 failures, randomized with seed 12639

  Testing sayHello()
    • test sayHello()
    • test a string
```

Testing an Array using toBe()

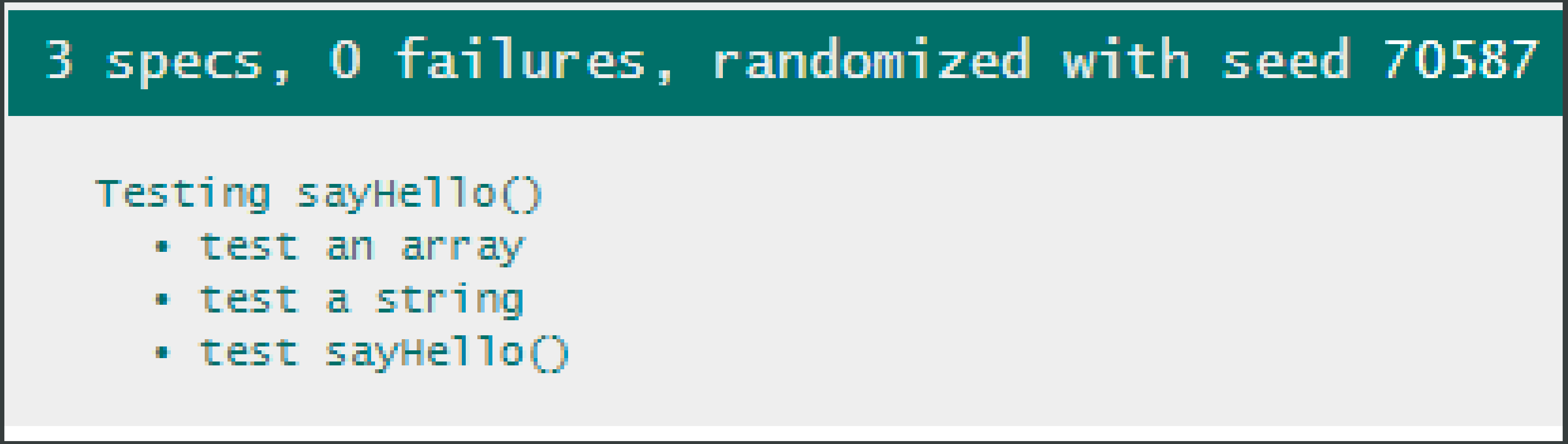
```
it('test an array', () => {
    let arr = [1, 2, 3]
    expect(arr).toBe([1, 2, 3])
})
```



```
3 specs, 1 failure, randomized with seed 27648                                    fi

Spec List | Failures

Testing sayHello() > test an array

Expected [ 1, 2, 3 ] to be [ 1, 2, 3 ]. Tip: To check for deep equality, use .toEqual() instead of .toBe().
Error: Expected [ 1, 2, 3 ] to be [ 1, 2, 3 ]. Tip: To check for deep equality, use .toEqual() instead of .toBe().
```

Testing an Array using toEqual()

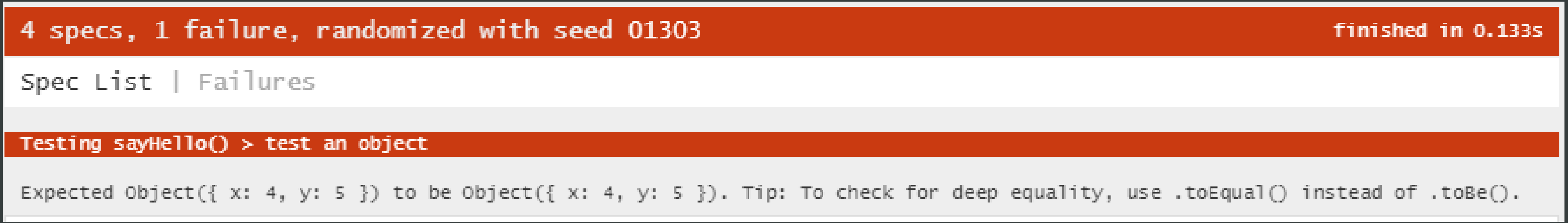
```
it('test an array', () => {
    let arr = [1, 2, 3]
    expect(arr).toEqual([1, 2, 3])
})
```



```
3 specs, 0 failures, randomized with seed 70587

  Testing sayHello()
    • test an array
    • test a string
    • test sayHello()
```

Testing an Object using toBe()

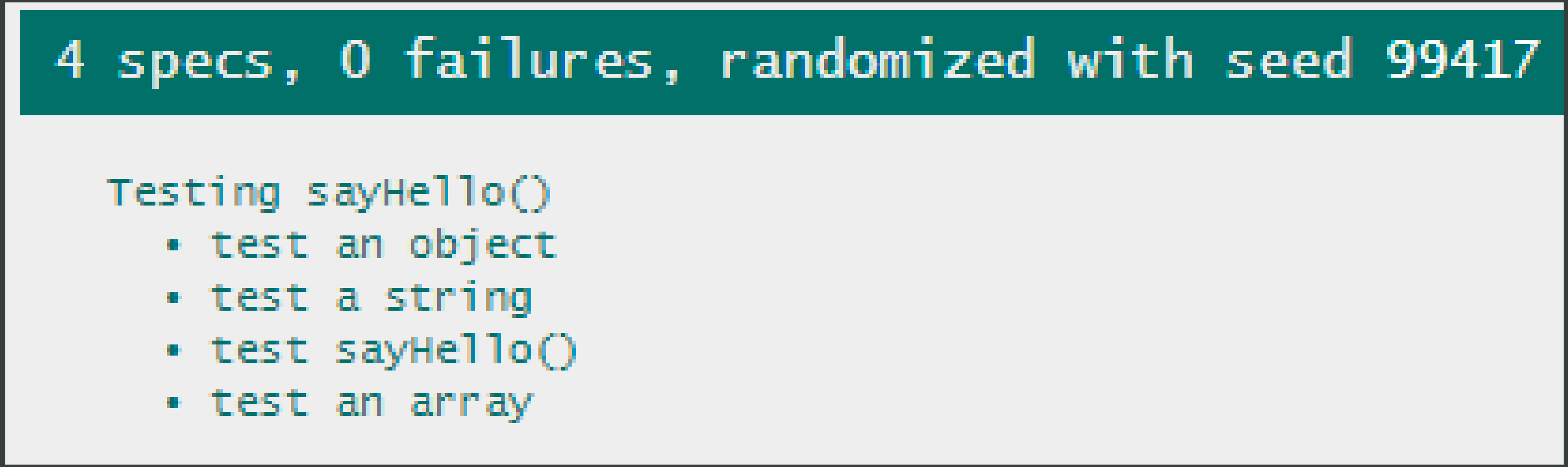
```
it('test an object', () => {
    let obj = { x: 4, y: 5 }
    expect(obj).toBe({ x: 4, y: 5 })
})
```



```
4 specs, 1 failure, randomized with seed 01303                                  finished in 0.133s
Spec List | Failures
Testing sayHello() > test an object
Expected Object({ x: 4, y: 5 }) to be Object({ x: 4, y: 5 }). Tip: To check for deep equality, use .toEqual() instead of .toBe().
```

Testing an Object using toEqual()

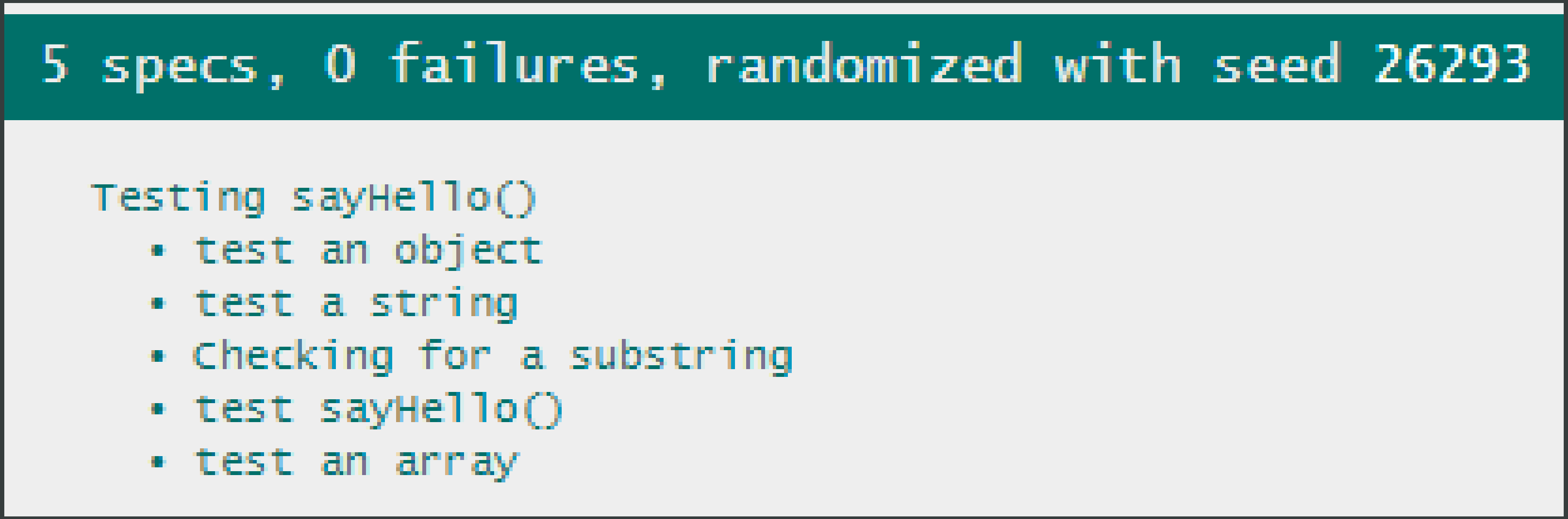
```
it('test an object', () => {
    let obj = { x: 4, y: 5 }
    expect(obj).toEqual({ x: 4, y: 5 })
})
```



```
4 specs, 0 failures, randomized with seed 99417

  Testing sayHello()
    • test an object
    • test a string
    • test sayHello()
    • test an array
```

Use of toContain()

```
it('Checking for a substring', () => {
    expect("Hello World").toContain("World")
})
```



```
5 specs, 0 failures, randomized with seed 26293

  Testing sayHello()
    • test an object
    • test a string
    • Checking for a substring
    • test sayHello()
    • test an array
```

Use of toContain()

```
expect([1, 2, 3, 4]).toContain(3);

expect(["Penguin", "Turtle", "Pig", "Duck"])
     .toContain("Duck");

var dog = { name: "Fido" };
expect([
    { name: "Spike" },
    { name: "Fido" },
    { name: "Spot" }
]).toContain(dog);
```

Yes or No?

To test if something evaluates to true:

```
expect(true).toBeTruthy();
expect(12).toBeTruthy();
expect({}).toBeTruthy();
```

To test if something evaluates to false:

```
expect(false).toBeFalsy();
expect(null).toBeFalsy();
expect("").toBeFalsy();
```

Note: False, 0, "", undefined, null and NaN are falsy in Jasmine (and in JavaScript too)

Negating Matchers

```
expect(foo).not.toEqual(bar);

expect("Hello planet").not.toContain("world");
```

Is It Defined?

```
var somethingUndefined;

expect("Hello!").toBeDefined();             // success
expect(null).toBeDefined();                 // success
expect(somethingUndefined).toBeDefined(); // failure

var somethingElseUndefined;

expect(somethingElseUndefined).toBeUndefined();// succ.
expect(12).toBeUndefined();                     // fail
expect(null).toBeUndefined();                   // fail
```

Nullness

```
let somethingUndefined

expect(null).toBeNull();                      // succ.

expect(false).toBeNull();                     // fail

expect(somethingUndefined).toBeNull(); // fail
```

Is It NaN?

```
expect(5).not.toBeNaN();                // success

expect(0 / 0).toBeNaN();                // success

expect(parseInt("hello")).toBeNaN(); // success
```

Using toMatch()

```
expect("hello world").toMatch(/world/);

expect("jasmine_book.jpg")
    .toMatch(/\w+.(jpg|gif|png|svg)/i);

expect("jasmine@example.com")
    .toMatch(/\w+@\w+\.\w+/);
```

Setup and Teardown

beforeAll: This function is called once, before all the specs in describe test suite are run.

afterAll: This function is called once after all the specs in a test suite are finished.

beforeEach: This function is called before each test spec, it function, has been run.

afterEach: This function is called after each test spec has been run.

Setup and Teardown

```
describe('Hello world', () => {
        let expected = "";
        beforeEach(   () => {    expected = "Hello World";     });
        afterEach(      () => {    expected = "";                             });
        it('says hello', () => {
                expect(helloWorld()).toEqual(expected);
        });
});
```

Testing an Angular Service

```
ng g s service/svcCalc

CREATE src/app/service/svc-calc.service.spec.ts
CREATE src/app/service/svc-calc.service.ts
```

```
ng g s service/svcLogger

CREATE src/app/service/svc-logger.service.spec.ts
CREATE src/app/service/svc-logger.service.ts
```

svc-logger.service.ts

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class SvcLoggerService {

  constructor() { }
  log(msg: string) {
    console.log(msg)
  }
}
```

svc-logger.service.spec.ts

```
import { TestBed } from '@angular/core/testing';
import { SvcLoggerService } from './svc-logger.service';

describe('SvcLoggerService', () => {
  let service: SvcLoggerService;

  beforeEach(() => {
    TestBed.configureTestingModule({});
    service = TestBed.inject(SvcLoggerService);
  });

  it('should be created', () => {
    expect(service).toBeTruthy();
  });
});
```

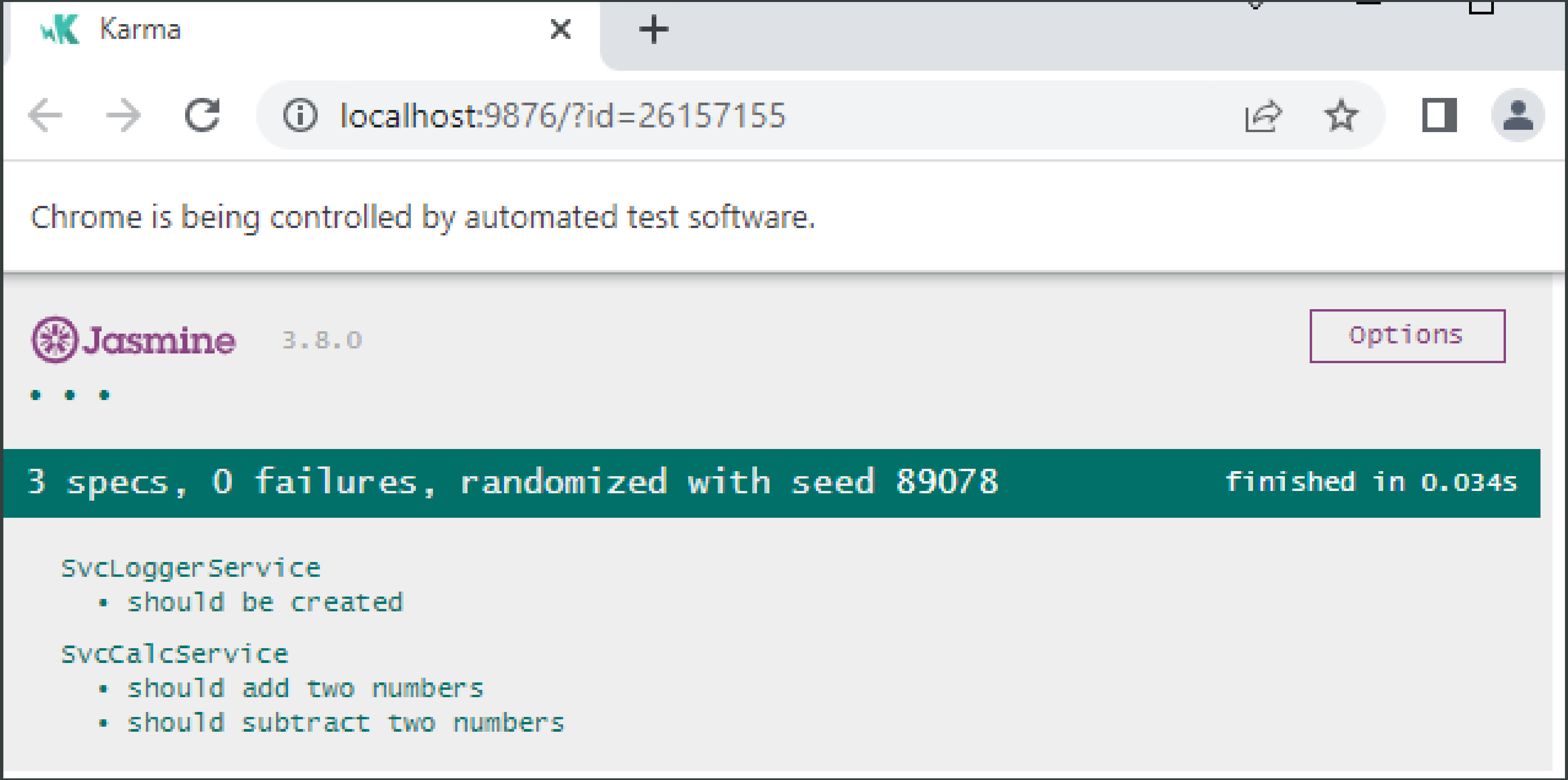
svc-calc.service.ts

```
import { Injectable } from '@angular/core';
import { SvcLoggerService } from './svc-logger.service';
@Injectable({  providedIn: 'root' })
export class SvcCalcService {
  constructor(private logger: SvcLoggerService) { }
  add(n1: number, n2: number): number {
    this.logger.log('svcCalcService: add()')
    return n1 + n2;
  }
  subtract(n1: number, n2: number): number {
    this.logger.log('svcCalcService: subtract()')
    return n1 - n2;
  }
}
```

svc-calc.service.spec.ts

```
import  { SvcCalcService }  from './svc-calc.service’;
import { SvcLoggerService } from './svc-logger.service';
describe('SvcCalcService', () => {
  it('should add two numbers', () => {
    const calc = new SvcCalcService(new SvcLoggerService())
    const result = calc.add(2, 3)
    expect(result).toBe(5)
  })
  it('should subtract two numbers', () => {
    const calc = new SvcCalcService(new SvcLoggerService())
    const result = calc.subtract(2, 3)
    expect(result).toBe(-1)
  })
});
```

Karma Output in Chrome Browser

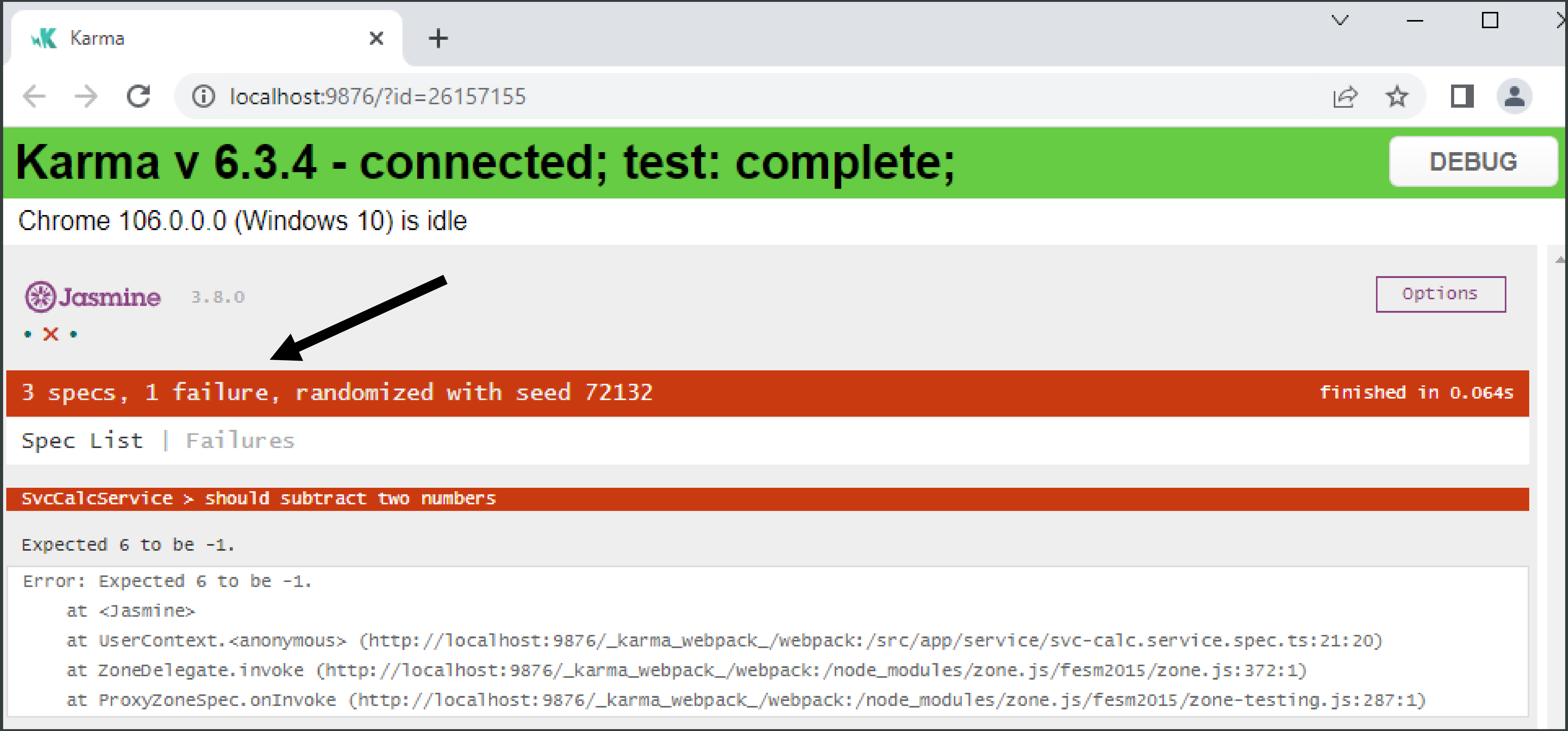


Chrome is being controlled by automated test software.

```
3 specs, 0 failures, randomized with seed 89078          finished in 0.034s

SvcLoggerService
  • should be created
SvcCalcService
  • should add two numbers
  • should subtract two numbers
```

svc-calc.service.ts (Updated)

```
import { Injectable } from '@angular/core';
import { SvcLoggerService } from './svc-logger.service';
@Injectable({  providedIn: 'root' })
export class SvcCalcService {
  constructor(private logger: SvcLoggerService) { }
  add(n1: number, n2: number): number {
    logger.log('svcCalcService: add()')
    return n1 + n2;
  }
  subtract(n1: number, n2: number): number {
    this.logger.log('svcCalcService: subtract()')
    return n1 * n2;
  }
}
```



Karma v 6.3.4 - connected; test: complete;

Chrome 106.0.0.0 (Windows 10) is idle

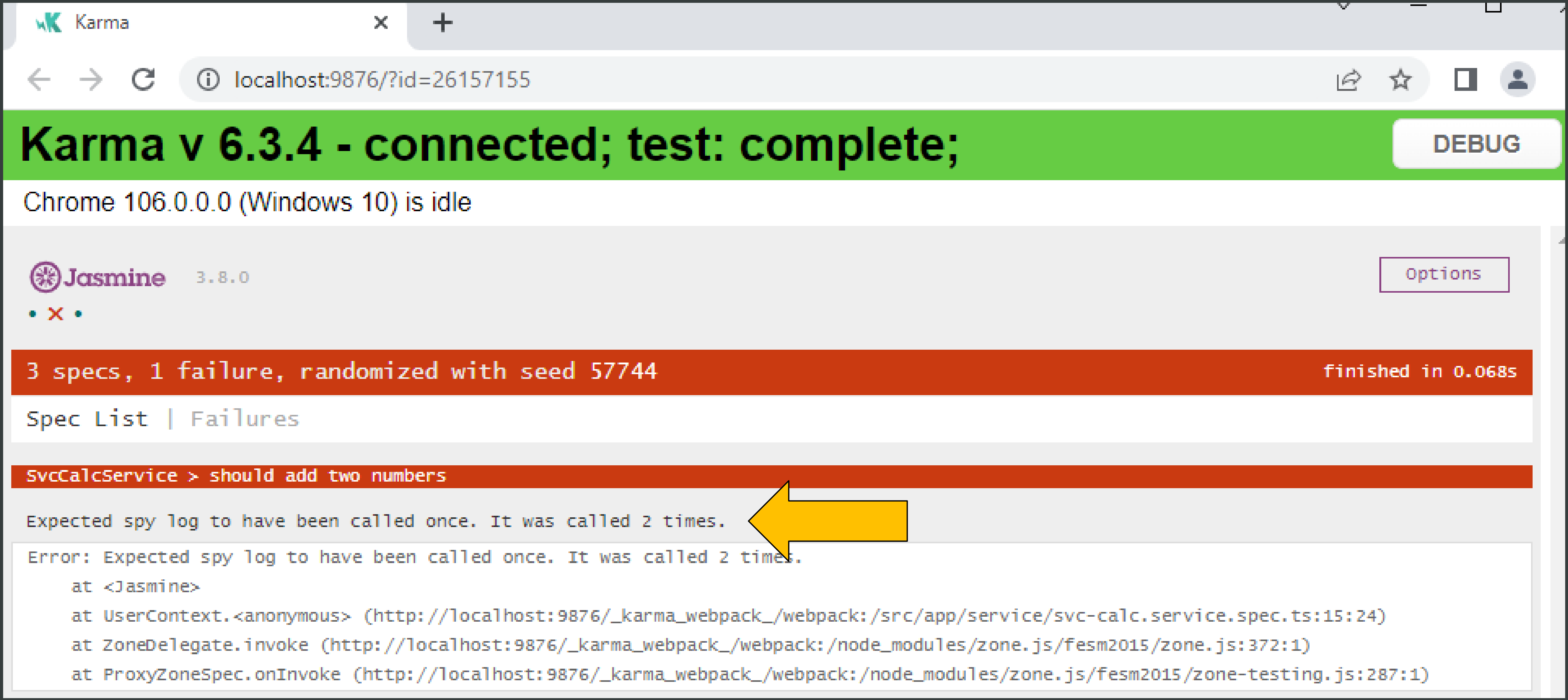
```
3 specs, 1 failure, randomized with seed 72132                finished in 0.064s

SvcCalcService > should subtract two numbers

Expected 6 to be -1.

Error: Expected 6 to be -1.
    at <Jasmine>
    at UserContext.<anonymous> (http://localhost:9876/_karma_webpack_/webpack:/src/app/service/svc-calc.service.spec.ts:21:20)
    at ZoneDelegate.invoke (http://localhost:9876/_karma_webpack_/webpack:/node_modules/zone.js/fesm2015/zone.js:372:1)
    at ProxyZoneSpec.onInvoke (http://localhost:9876/_karma_webpack_/webpack:/node_modules/zone.js/fesm2015/zone-testing.js:287:1)
```

svc-calc.service.spec.ts (Updated)

```
...
describe('SvcCalcService', () => {
  it('should add two numbers', () => {
    const logger = new SvcLoggerService()
    spyOn(logger, "log")
    const calc = new SvcCalcService(logger)
    const result = calc.add(2, 3)
    expect(result).toBe(5)
    expect(logger.log).toHaveBeenCalledTimes(1)
  })
...
```

svc-calc.service.ts (Updated)

```
...
export class SvcCalcService {
  constructor(private logger: SvcLoggerService)
  {
  }
  add(n1: number, n2: number): number {
    logger.log('svcCalcService: add()')
    logger.log('svcCalcService: add()')
    return n1 + n2;
  }
  ...
```



```
Karma v 6.3.4 - connected; test: complete;
Chrome 106.0.0.0 (Windows 10) is idle

3 specs, 1 failure, randomized with seed 57744                finished in 0.068s

SvcCalcService > should add two numbers

Expected spy log to have been called once. It was called 2 times.

Error: Expected spy log to have been called once. It was called 2 times.
    at <Jasmine>
    at UserContext.<anonymous> (http://localhost:9876/_karma_webpack_/webpack:/src/app/service/svc-calc.service.spec.ts:15:24)
    at ZoneDelegate.invoke (http://localhost:9876/_karma_webpack_/webpack:/node_modules/zone.js/fesm2015/zone.js:372:1)
    at ProxyZoneSpec.onInvoke (http://localhost:9876/_karma_webpack_/webpack:/node_modules/zone.js/fesm2015/zone-testing.js:287:1)
```

svc-calc.service.spec.ts (Fake Service)

Fake Service

```
...
describe('SvcCalcService', () => {
  it('should add two numbers', () => {
    const logger =
      jasmine.createSpyObj("SvcLoggerService", ["log"])
    ~~spyOn(logger, "log")~~
    const calc = new SvcCalcService(logger)
    const result = calc.add(2, 3)
    expect(result).toBe(5)
    expect(logger.log).toHaveBeenCalledTimes(1)
  })
  ...
```

svc-calc.service.spec.ts (beforeEach)

```
describe('SvcCalcService', () => {
  let calc: SvcCalcService, logger: any
  beforeEach(() => {
    logger = jasmine.createSpyObj("SvcLoggerService", ["log"])
    calc = new SvcCalcService(logger)
  })
  it('should add two numbers', () => {
    expect(calc.add(2, 3)).toBe(5)
    expect(logger.log).toHaveBeenCalledTimes(1)
  })
  it('should subtract two numbers', () => {
    expect(calc.subtract(2, 3)).toBe(-1)
    expect(logger.log).toHaveBeenCalledTimes(1)
  })
});
```

svc-calc.service.spec.ts (TestBed)

```typescript
import { SvcCalcService }  from './svc-calc.service';
import { SvcLoggerService }from './svc-logger.service';
import { TestBed }         from '@angular/core/testing';
describe('Calculator Service', () => {
  let calc: SvcCalcService, logger: any
  beforeEach(() => {
    logger = jasmine.createSpyObj("SvcLoggerService", ["log"])
    TestBed.configureTestingModule({
      providers: [
        SvcCalcService,
        {provide: SvcLoggerService, useValue: logger}
      ]
    })
    calc = new SvcCalcService(logger)
  })
});
```

svc-calc.service.spec.ts (focus on test suite or spec)

```
...
fdescribe('Calculator Service', () => {
...
  fit('should add two numbers', () => {
    const result = calc.add(2, 3)
    expect(result).toBe(5)
    expect(logger.log).toHaveBeenCalledTimes(1)
  })
  it('should subtract two numbers', () => {
    const result = calc.subtract(2, 3)
    expect(result).toBe(-1)
    expect(logger.log).toHaveBeenCalledTimes(1)
  })
});
```

emp.service.ts

```
import { Injectable } from '@angular/core';
import { IEmployee } from './employee';
@Injectable({  providedIn: 'root' })
export class EmployeeService {
  employees : IEmployee[] = [
    { "id": 1, "name": "Arun", "designation": "Developer" },
    { "id": 2, "name": "Sneha", "designation": "DBA" } ]
  constructor() { }
  getEmployees(): IEmployee[] {
    return this.employees
  }
  getEmployee(id : number) : IEmployee {
    return this.employees[id]
  }
}
```

emp.service.spec.ts

```
import { TestBed } from '@angular/core/testing';
import { IEmployee } from './employee';
import { EmployeeService } from './employee.service';
describe('EmployeeService', () => {
  let service : EmployeeService
  beforeEach(() => {
  TestBed.configureTestingModule({providers:[EmployeeService]});
    service = TestBed.inject(EmployeeService);
  })
  it('should create a service', () => {
    expect(service).toBeTruthy();
  })
})
```

emp.service.spec.ts

```
...
  it('should be called when getEmployee() method invoked', () => {
    let spy = spyOn(service, 'getEmployee').and.callFake( (id) => {
      return { id: 1, name: 'test', designation : 'testing'}
    })

    service.getEmployee(1)

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

emp.service.spec.ts

```
...
  it('should return an emp detail when provided with an id', () => {
    let dummyEmp = { id: 1, name: 'test', designation: 'testing' }

    let spy = spyOn(service, 'getEmployee').and.callFake( (id) => {
      return dummyEmp
    })

    let emp : IEmployee = service.getEmployee(1)

    expect(emp).toEqual(dummyEmp)
  })
...
```

emp.service.spec.ts

```
...
  it('should return all employees detail from the server', () => {
 let dummyEmps = [{ id: 1, name: 'name1', designation: 'desig1' },
                  { id: 2, name: 'name2', designation: 'desig2’ } ]

    let spy = spyOn(service, 'getEmployees’).
                       and.returnValues(dummyEmps)

    let emp : IEmployee[] = service.getEmployees()

    expect(emp).toEqual(dummyEmps)
  })
...
```

Testing an Angular Pipe (title-case.pipe.ts)

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({  name: 'titleCase',  pure: true})
/** Transform to Title Case:
uppercase the first letter of the words in a string. */
export class TitleCasePipe implements PipeTransform {

  transform(input: string): string {
    return input.length === 0 ? '' :
        input.replace(/w\S*/g,
            (text)=> text[0].toUpperCase() +
               text.slice(1).toLowerCase())
  }
}
```

Testing an Angular Pipe (title-case.pipe.spec.ts)

```
import { TitleCasePipe } from './title-case.pipe';
describe('TitleCasePipe', () => {
  const pipe = new TitleCasePipe();

  it('transforms "abc" to "Abc"', () => {
    expect(pipe.transform('abc')).toBe('Abc');
  });

  it('transforms "abc def" to "Abc Def"', () => {
    expect(pipe.transform('abc def')).toBe('Abc Def');
  });
});
```

Manually failing a spec with fail

```
describe("A spec using the fail function", function() {
        var test = function(x, callBack) {
                if (x) {
                        callBack();
                }
        };
        it("should call the callBack", function() {
                test(true, function() {
                        fail("Callback has been called");
                });
        });
});
```

Nesting Suites

- As the code size gets increases, we can organize our suites into groups, subgroups sub-subgroups, and so on.

- Jasmine makes it very easy for you to do that by simply nesting the specs.

Nesting describe Blocks

```
describe("A spec", function() {
        var outer
        beforeEach( function() {  outer = 1  } )
        it("just a fun", function() {    expect(outer).toEqual(1)       })
        describe("nested inside a describe", function() {
                var inner
                beforeEach(function() {      inner = 1;    })
                it("can reference both scopes as needed", function() {
                        expect(outer).toEqual(inner)
                })
         })
})
```

Disabling Suites

```
xdescribe("A suite", function() {
        it("contains spec with an expectation", function()
{
                expect(true).toBe(true);
        })
})
```

Pending Specs

```
describe("Pending specs", function() {
        xit("can be declared 'xit'", function() {
                expect(true).toBe(false);
        });

        it("can be declared without a function body");

        it("calling 'pending' in the spec body", function() {
                expect(true).toBe(false);
                pending('this is why it is pending');
        });
});
```

Spies

- Jasmine spy allows to spy on application functions calls.
- A spy can stub any function and tracks calls to it and all arguments.
- A spy only exists in the describe or it block in which it is defined, and will be removed after each spec.
- There are special matchers for interacting with spies.

Spies (spyOn)

```
describe("A spy", function() {
        var obj, num = null;
        beforeEach(function() {
                obj = { someMethod: function(value) { num = value; } }
                spyOn(obj, 'someMethod')
                obj.someMethod(123)
                obj.someMethod(456, ' param2 ')
        })
        it("tracks that the spy was called", function() {
                expect(obj.someMethod).toHaveBeenCalled()
        })
})
```

Spies (spyOn)

```
describe("A spy", function() {
        var obj, num = null;
        beforeEach(function() {
                obj = { someMethod: function(value) { num = value; } }
                spyOn(obj, 'someMethod')
                obj.someMethod(123)
                obj.someMethod(456, ' param2 ')
        })
        it("tracks that the spy was called x times", function() {
                expect(obj.someMethod).toHaveBeenCalledTimes(2)
        })
})
```

Spies (spyOn)

```
describe("A spy", function() {
        var obj, num = null;
        beforeEach(function() {
                obj = { someMethod: function(value) { num = value; }  }
                spyOn(obj, 'someMethod')
                obj.someMethod(123)
                obj.someMethod(456, 'param2')
        })
        it("tracks all the arguments of its calls", function() {
                expect(obj.someMethod).toHaveBeenCalledWith(123)
                expect(obj.someMethod).toHaveBeenCalledWith(456, 'param2')
        })
})
```

Spies (spyOn)

```
describe('Testing sayHello()', () => {
    var obj: any, num: any = null;

    beforeEach(function () {
        obj = {
            someMethod: (value: any, second?: any) => num = value
        }
        spyOn(obj, 'someMethod')
        obj.someMethod(123)
        obj.someMethod(456, 'param2')
    })
    it("tracks the arguments of each call", function () {
        expect(obj.someMethod.calls.argsFor(0)).toEqual([123])
        expect(obj.someMethod.calls.argsFor(1)).toEqual([456, 'param2'])
    })
})
```

Spies (spyOn)

```
describe("A spy", function() {
	var obj;
	beforeEach(function() {
		obj = { someMethod: function(value) { console.log('fn'); } }
		spyOn(obj, 'someMethod')
		obj.someMethod(123)
		obj.someMethod(456, 'param2')
	})
	it("tracks if it was called at all", function() {
		expect(obj.someMethod.calls.any()).toEqual(true)
	})
})
```

Spies (spyOn)

```
describe("A spy", function() {
        var obj;
        beforeEach(function() {
                obj = { someMethod: function(value) { console.log('fn'); } }
                spyOn(obj, 'someMethod')
                obj.someMethod(123)
                obj.someMethod(456, 'param2')
        })
        it("tracks if it was called at all", function() {
                expect(obj.someMethod.calls.count()).toEqual(2)
        })
})
```

Spies (spyOn)

```
describe("A spy", function() {
        var foo, bar = null;
        beforeEach(function() {
                foo = {  setBar: function(value)   {   bar = value   }
                spyOn(foo, 'setBar')
        };
        it("tracks the arguments of each call", function() {
                foo.setBar(123);
                foo.setBar(456, 'test');
                expect(foo.setBar.calls.argsFor(0)).toEqual([123]);
                expect(foo.setBar.calls.argsFor(1)).toEqual([456, 'test']);
        })
})
```

Spies (spyOn)

```
describe("A spy", function() {
        var foo, bar = null;
        beforeEach(function() {
                foo = { setBar: function(value) {           bar = value   }
                spyOn(foo, 'setBar')
        };
        it("tracks the arguments of all call", function() {
                foo.setBar(123);
                foo.setBar(456, 'test');
                expect(foo.setBar.calls.allArgs()).toEqual([[123], [456, 'test']]);
        })
})
```

Spies (spyOn)

```
describe("A spy", function() {
        var foo, bar = null;
        beforeEach(function() {
                foo = { setBar: function(value) {          bar = value   }
                spyOn(foo, 'setBar')
        };
        it("can be reset", function() {
                foo.setBar(123);
                expect(foo.setBar.calls.any()).toBe(true);
                foo.setBar.calls.reset();
                expect(foo.setBar.calls.any()).toBe(false);
        })
})
```

Spies (spyOn, and.callThrough)

```
describe("A spy", function() {
  var foo, bar, fetch;

  beforeEach(function() {
    foo = {
      setBar: function(value) {       bar = value;      },
      getBar: function() {                return bar      }
    };
    spyOn(foo, 'setBar').and.callThrough();
    foo.setBar(123)
    fetch = foo.getBar()
  });
  it("should not affect other functions", function() {   expect(bar).toEqual(123);  });
  it("returns the requested value", function() {           expect(fetch).toEqual(123);  });
});
```

Spies (spyOn, and.returnValues)

```
describe("A spy", function() {
 var foo, bar, fetch;
 beforeEach(function() {
   foo = {
     setBar: function(value) {       bar = value;      },
     getBar: function() {                return bar      }
   };
   spyOn(foo, 'getBar').and.returnValues('first', 'second');
   foo.setBar(123)
});
 it("when called multiple times returns the requested values in order", function() {
           expect(foo.getBar()).toEqual("first");
            expect(foo.getBar()).toEqual("second");
           expect(foo.getBar()).toBeUndefined();
 });
});
```

Spies (spyOn, and.callFake)

```
describe("A spy", function() {
   var foo, bar, fetch;

   beforeEach(function() {
   foo = {
          setBar: function(value) {       bar = value;      },
          getBar: function() {         return bar      }
   };
   spyOn(foo, "getBar").and.callFake(function(args, can, be) {  return 1000;    })
          foo.setBar(123)
          fetch = foo.getBar();
   });
   it("returns the requested value", function() {   expect(bar).toEqual(123);       })
   it("returns the requested value", function() {   expect(fetch).toEqual(1000);  })
});
```

Creating a new Spy Function

- Sometime it is useful to create a spy for a function that doesn't yet exist.
- jasmine.createSpy can create a "bare" spy
- This spy acts as any other spy:
  - tracking calls
  - arguments, etc.
- There is no implementation behind it

Spies (createSpy)

```
it("is having a spy function", function () {
    var person = { getName() { } };
    person.getName = jasmine.createSpy("Spy function");
    person.getName();
    expect(person.getName).toHaveBeenCalled();
})
```

Spies (createSpyObj)

- In order to create a mock with multiple spies, use jasmine.createSpyObj and pass an array of strings
- It returns an object that has a property for each string that is a spy

Spies (createSpyObj)

```
describe("Multiple spies, when created manually", function() {
        var tape;
        beforeEach(function() {
                tape = jasmine.createSpyObj('tape', ['play', 'pause', 'stop', 'rewind']);
                tape.play();            tape.pause();           tape.rewind(0);
        })
        it("creates spies for each requested function", function() {
                expect(tape.play).toBeDefined();
                expect(tape.pause).toBeDefined();
                expect(tape.stop).toBeDefined();
                expect(tape.rewind).toBeDefined();
        });
});
```

Spies (createSpyObj)

```
describe("Multiple spies, when created manually", function() {
        var tape;
        beforeEach(function() {
        tape = jasmine.createSpyObj('tape', ['play', 'pause', 'stop', 'rewind']);
                tape.play();  tape.pause();                    tape.rewind(0);
        })
        it("tracks that the spies were called", function() {
                expect(tape.play).toHaveBeenCalled();
                expect(tape.pause).toHaveBeenCalled();
                expect(tape.rewind).toHaveBeenCalled();
                expect(tape.stop).not.toHaveBeenCalled();
        })
})
```

Spies (createSpyObj)

```
describe("Multiple spies, when created manually", function() {
        var tape;
        beforeEach(function() {
   tape = jasmine.createSpyObj('tape', ['play', 'pause', 'stop', 'rewind']);
                tape.play();        tape.pause();               tape.rewind(0);
        })
        it("tracks all the arguments of its calls", function() {
                expect(tape.rewind).toHaveBeenCalledWith(0);
        })
})
```

Matching anything with jasmine.any

```
describe("jasmine.any", function() {
	it("matches any value", function() {
		expect({  }).toEqual(jasmine.any(Object));
		expect(12).toEqual(jasmine.any(Number));
	});
});
```

Matching anything with jasmine.any

```
describe("when used with a spy", function() {
        it("is useful for comparing arguments", function() {
                var fn = jasmine.createSpy(‘spy function');

                fn(12, function() {
                        return true;
                })

                expect(fn).toHaveBeenCalledWith(jasmine.any(Number),
                                                jasmine.any(Function) )
        });
});
```

Matching existance with jasmine.anything

```
// jasmine.anything returns true if the actual value is not null or undefined.

describe("jasmine.anything", function() {

        it("matches anything", function() {
                expect(123).toEqual(jasmine.anything());
        })
})
```

Matching existance with jasmine.anything

```
describe("when used with a spy", function() {
	it("is useful when the argument can be ignored", function() {
		var fn = jasmine.createSpy(‘spy function');
		fn(12, function() {
			return false;
		})

		expect(fn).toHaveBeenCalledWith(
						12,jasmine.anything());
	})
})
```

Partial matching with jasmine.objectContaining

```
describe("jasmine.objectContaining", function() {
    var obj;

    beforeEach(function() {
        obj = {     num: 1,     msg: "hello"    };
    });

    it("matches objects with the expect key/value pairs", function() {
        expect(obj).toEqual(jasmine.objectContaining({  msg: "hello" }))
        expect(obj).not.toEqual(jasmine.objectContaining({ contact: 123  }))
    })
})
```

Partial Array Matching with jasmine.arrayContaining

```
describe("jasmine.arrayContaining", function() {
        var numArray

        beforeEach(function() {
                numArray = [1, 2, 3, 4]
        })

        it("matches arrays with some of the values", function() {
                expect(numArray).toEqual(jasmine.arrayContaining([3, 1]))
                expect(numArray).not.toEqual(jasmine.arrayContaining([6]))
        })
})
```

Mocking the JavaScript Timeout Functions

```
describe("Manually ticking the Jasmine Clock", function() {
        var timerCallback
        beforeEach(function() {
                timerCallback = jasmine.createSpy("timerCallback");
                jasmine.clock().uninstall()   // uninstall() is necessary before install()
                jasmine.clock().install()
        });
        it("causes a timeout to be called synchronously", function() {
                setTimeout(function() { timerCallback()  }, 100)
                expect(timerCallback).not.toHaveBeenCalled()
                jasmine.clock().tick(101)
                expect(timerCallback).toHaveBeenCalled()
        })
})
```

Asynchronous Support

- Jasmine also has support for running specs that require testing asynchronous operations
- The functions that you pass to beforeAll, afterAll, beforeEach, afterEach, and it can be asynchronous
- There are three different ways to indicate that a function is asynchronous:
  1. by taking an optional callback parameter,
  2. by returning a promise, or
  3. by using the async keyword in environments that support it.

References

1. https://angular.io/docs
2. https://jasmine.github.io/tutorials/your_first_suite
3. JavaScript Testing with Jasmine by Evan Hahn, First Edition, O'Reilly, 2013
4. https://www.tutorialspoint.com/jasminejs/index.htm