

Unit Testing in JavaScript: Jest

05/02/2024

About us



Aday Cuesta Correa
(alu0101483887)



Juan Rodríguez Suárez
(juan.suarez.12)

```
1 Index {
2
3
4     01 History
5     02 What is Unit Testing?
6     03 How to install
7     04 Syntax and How to use
8     05 Matchers
9     06 Tips & Tricks
10    07 Bibliography
11
12
13 }
14
```

```
1 01 {
2
3
4
5 [History]
6
7
8 < How it all began >
9
10
11
12 }
13
14
```

The beginning of 'Unit Testing' {

< **Unit testing** and **debugging** were
essentially the same thing >

< Code became so much **complex** >

< **Kent Bent** created JUnit. He
called the approach **unit test** >

}



< Kent Bent >

The evolution to 'TDD' {

< **Refactoring** suggests ways to transform code to make it more isolated and testable. >

< In **TDD**, code must be testable before it is even created >

}



< Martin Fowler >

02 {

[What is Unit Testing]

< How does it works? >

}

Software development methodology {

- Simple
- Quality code
- Supported by many languages

}



What is it about? {

1. Write your code
2. Write tests which verify certain functionalities of your code
3. Execute tests
4. Fix errors on your code and repeat

}



Other methodologies {

- Test Driven Development (TDD)
- Snapshot Testing (modern approach)
- Behavior-Driven Development, an extension of TDD (BDD)

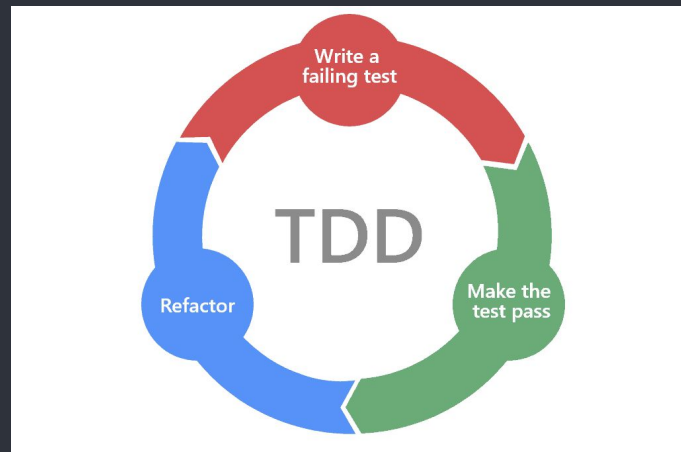
}



Approach to TDD {

- Code must be testable even before it's created

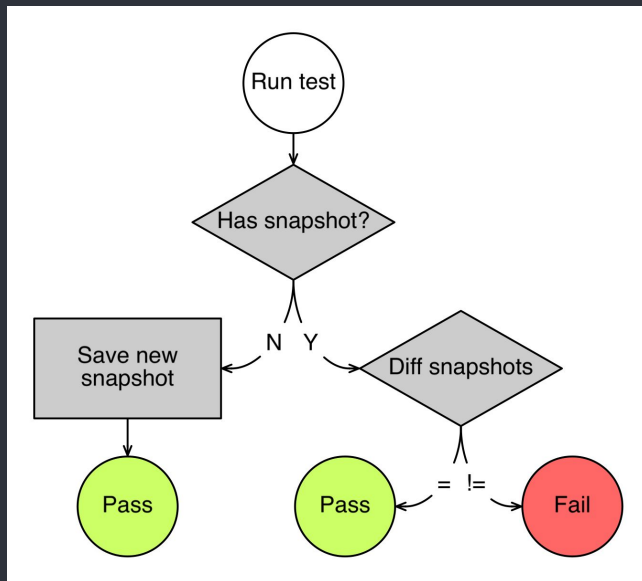
}



Modern approach of testing {

- Capturing code outputs (**snapshot**)
- Compare new outputs to the snapshot and see the differences

}



Advantages of UT {

- Early bug detection
- Eases code comprehension
- Measure of quality

}

Disadvantages of UT {

- High initial cost
- Possible lack of code covering
- Sensitive to frequent code changes
- UT does not increase development time if done well!

}

Unit Testing frameworks {

- Python: PyUnit
- Java: JUnit o TestNG
- C#: NUnit
- Ruby: RSpec o Test::Unit (LPP)
- C++: Google Test (IB)
- JavaScript: Mocha o Jest (PAI)

}

03 {

[Installation]

< In few steps >

}

About Jest {

'What is Jest?'

< Jest is a JavaScript testing framework designed to ensure correctness of any JavaScript codebase >

'Advantages/Features'

- 1.-- Zero configuration needed;
- 2.-- Fast;
- 3.-- Built in code coverage;
- 4.-- Isolated and Sandboxed tests;
- 5.-- Support Snapshot Testing;

}

Jest Installation 'Step by Step' {

Step 01 yarn add --dev jest

Step 02 npm install --save-dev jest

'What if I want to install the module globally?'

Step 01 npm install -g jest

}

Installation of Jest In a Node-based project 'Step by Step' {

Step 01 Create a folder/directory with a name as your project name, for example → `mkdir myFirstNodeProject`

Step 02 `cd myFirstNodeProject; npm init`

Step 03 Keep Pressing Enter

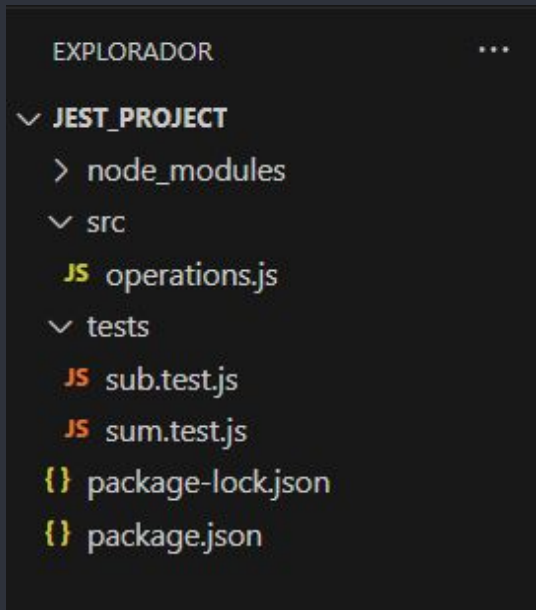
Step 04 `npm install --save-dev jest`

Step 05 Configure the npm test script to run the Jest tests i.e. when the command '**npm test**' is executed

}

```
1 package.json {
2   "name": "jest-e2e",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "jest"
8   },
9   "author": "",
10  "license": "ISC",
11  "dependencies": {
12    "jest": "^25.1.0"
13  }
14 }
```

Directory structure {



1. All 'useful' code in ./src/ directory
2. All tests in ./tests/ directory
3. **Test files** names like 'unit.test.js'

Note when using ES Modules {

- ES Modules are not the default option in Node.
- In order to use them instead of CommonJS, we need to change our package.json:

```
"type": "module",
  ▶ Depurar
  "scripts": {
    | "test": "node --experimental-vm-modules node_modules/jest/bin/jest.js"
  },
```

}

04 {

[Syntax and How to use]

< With examples! >

}

describe keyword {

- Defines a group of related **tests** about some specific feature in our code.
- Takes two arguments:
 - **String** : *name* → Description of our group of tests.
 - **Function** : *fn* → Function which contains all of our **tests**.
- Best practice: One **describe** per class or function

}

test keyword {

- Defines a group of **expectations** about an even more concrete aspect of our code described by the **describe** the test is surrounded by.
- Takes two arguments:
 - **String** : *name* → Description of our test.
 - **Function** : *fn* → Function which contains all of our **expectations**.
 - **Number** : *timeout* → Maximum time (milliseconds) for a test to run (default 5 ms).
- Best practice: One **test** per functionality.

}

expect keyword {

- The **expect** function is used every time we want to test a value.
- We will use **expect** along with a "**matcher**" function to assert something about a value.
- Takes two arguments:
 - The value that your code produces.
 - Any argument to the matcher should be the correct value.
- Best practice: Use fairly simple expects so the code is easier to understand.

}

First we need some code {

```
'myFirstNodeProject/src/calculator.js'
const mathOperations = {
  sum: function(first_number, second_number) {
    return first_number + second_number;
  },
  diff: function(first_number, second_number) {
    return first_number - second_number;
  },
  product: function(first_number, second_number) {
    return first_number * second_number;
  }
}
module.exports = mathOperations;
```

Let's create a test {

```
'myFirstNodeProject/test/calculator.test.js'  
  
//This should be always at the top of the file  
//In this case we are using our calculator example code  
const mathOperations = require('./calculator');  
  
describe('Calculator tests', () => {  
  test('adding 1 + 2 should return 3', () => {  
    expect(mathOperations.sum(1, 2)).toBe(3);  
  });  
})  
}
```

Let's rewrite it {

```
1  'myFirstNodeProject/test/calculator.test.js'  
2  
3  const mathOperations = require('./calculator');  
4  
5  
6  describe('Calculator tests', () => {  
7    test('adding 1 + 2 should return 3', () => {  
8      // arrange and act  
9      const RESULT = mathOperations.sum(1, 2);  
10     // assert  
11     expect(RESULT).toBe(3);  
12   });  
13 }  
14 }
```

Test output {

```
PASS ./calculator.test.js
  Calculator tests
    ✓ adding 1 + 2 should return 3 (2ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.798s, estimated 1s
```

}

Let's create another test {

```
1      'myFirstNodeProject/test/calculator.test.js'
2
3      describe('Calculator tests', () => {
4
5          test('adding 1 + 2 should return 10', () => {
6              // arrange and act
7              const RESULT = mathOperations.sum(1, 2);
8              // assert
9              expect(RESULT).toBe(10);
10          });
11      });
12
13  }
14
```

Test output {

```
FAIL ./calculator.test.js
Calculator tests
  ✕ adding 1 + 2 should return 10 (4ms)

• Calculator tests > adding 1 + 2 should return 10

expect(received).toBe(expected) // Object.is equality

Expected: 10
Received: 3

    7 | |
    8 | // assert
  >  9 | expect(result).toBe(10);
      |                  ^
    10 | });
    11 | })
    12 |

      at Object.<anonymous> (calculator.test.js:9:20)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:  0 total
Time:        0.846s, estimated 1s
```


One big describe block {

```
1 describe('Calculator tests', () => {
2   test('adding 1 + 2 should return 3', () => {
3     // arrange and act
4     const RESULT = mathOperations.sum(1, 2)
5     // assert
6     expect(RESULT).toBe(3);
7   });
8   test('subtracting 2 from 10 should return 8', () => {
9     // arrange and act
10    const RESULT = mathOperations.diff(10, 2)
11    // assert
12    expect(RESULT).toBe(8);
13  });
14  test('multiplying 2 and 8 should return 16', () => {
15    // arrange and act
16    const RESULT = mathOperations.product(2, 8)
17    // assert
18    expect(RESULT).toBe(16);
19  });
20 });
```

Test output {

```
PASS ./calculator.test.js
  Calculator tests
    ✓ adding 1 + 2 should return 3 (2ms)
    ✓ subtracting 2 from 10 should return 8
    ✓ multiplying 2 and 8 should return 16 (1ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.834s, estimated 1s
Ran all test suites.
```

}

1
2 05 {
3
4

5 [Matchers]
6
7

8 < Almost all of them >
9
10

11 }
12
13
14

Equality Matchers {

```
test('equality matchers', () => {  
  expect(2 * 2).toBe(4);  
  expect(4 - 2).not.toEqual(1);  
})
```

Hello World

Poetry Club

}

Exception Matchers {

```
function throwErrorExample() {  
  throw new Error('This is a custom error message');  
}
```

```
test('throwErrorExample throws an error', () => {  
  expect(() => throwErrorExample()).toThrow();  
});
```

```
}  
  Collatz Conjecture  
  Bank Account
```

Truthiness Matchers {

```
test('truthy operators', () => {  
  const NAME = 'Software testing help';  
  const SOME_VALUE = null;  
  expect(SOME_VALUE).toBeNull();  
  expect(NAME).not.toBeNull();  
  expect(NAME).toBeTruthy();  
  expect(SOME_VALUE).not.toBeTruthy();  
  expect(SOME_VALUE).toBeFalsy();  
  expect(0).toBeFalsy();  
})
```

Pangram

}

Floating Point Matchers {

```
test('adding works sanely with decimals', () => {  
  const FLOAT_1 = 0.2;  
  const FLOAT_2 = 0.1;  
  expect(FLOAT_1 + FLOAT_2).toBeCloseTo(0.3, 5);  
})
```

Freelancer Rates

}

String Matchers {

```
test('string matchers',() => {  
  const MY_SENTENCE = 'software testing help';  
  // test for success match  
  expect(MY_SENTENCE).toMatch(/test/);  
  // test for failure match  
  expect(MY_SENTENCE).not.toMatch(/abc/);  
})
```

Parse String

}

toHaveBeenCalled Matcher {

```
1
2
3     function drinkAll(callback, flavour) {
4         if (flavour === 'octopus') {
5             callback(flavour);
6         }
7     }
8
9     test('drinks something lemon-flavoured', () => {
10         const DRINK = jest.fn(); // Spy function
11         drinkAll(DRINK, 'lemon');
12         expect(DRINK).toHaveBeenCalled();
13     });
14 }
```

Fruit Picker

toHaveBeenCalledTimes Matcher {

```
1
2
3
4
5     test('sum function is called twice', () => {
6         jest.spyOn(mathOperations, 'sum'); // Spy the sum function
7
8         mathOperations.sum(2, 3);
9         mathOperations.sum(4, 5);
10
11         expect(mathOperations.sum).toHaveBeenCalledTimes(2);
12     });
13 }
14 Fruit Picker
```

```
1 toHaveBeenCalledWith Matcher {
```

```
2  
3     test('sum function is called with specific arguments', () => {  
4         jest.spyOn(mathOperations, 'sum');
```

```
5  
6         mathOperations.sum(2, 3);
```

```
7  
8         expect(mathOperations.sum).toHaveBeenCalled(2, 3);
```

```
9  
10        mathOperations.sum(4, 5);
```

```
11  
12        expect(mathOperations.sum).toHaveBeenCalled(4, 5);  
13    });
```

```
14 }
```

Fruit Picker

Number Matchers {

```
test('numeric operators', () => {  
  const FIRST_NUMBER = 100;  
  const SECOND_NUMBER = -20;  
  const THIRD_NUMBER = 0;  
  // greater than  
  expect(FIRST_NUMBER).toBeGreaterThan(10)  
  // less than or equal  
  expect(SECOND_NUMBER).toBeLessThanOrEqual(0)  
  // greater than or equal  
  expect(THIRD_NUMBER).toBeGreaterThanOrEqual(0)  
})
```

}

toHaveReturned Matcher {

```
1
2
3
4
5 test('sum function is called and has returned a value', () => {
6   jest.spyOn(mathOperations, 'sum');
7   mathOperations.sum(2, 3);
8   expect(mathOperations.sum).toHaveReturned();
9   });
10
11
12
13
14 }
```

toHaveReturnedTimes Matcher {

```
1
2
3     test('sum returns twice', () => {
4         jest.spyOn(mathOperations, 'sum');
5
6         mathOperations.sum.mockClear();
7
8         mathOperations.sum(2, 3);
9         mathOperations.sum(5, 4);
10
11         expect(mathOperations.sum).toHaveReturnedTimes(2);
12     });
13 }
14
```

toHaveReturnedWith Matcher {

```
1
2
3
4
5 test('sum function is called and returns a specific value', () => {
6   jest.spyOn(mathOperations, 'sum');
7   const RESULT= mathOperations.sum(3, 4);
8   expect(mathOperations.sum).toHaveReturnedWith(7);
9 });
10
11
12
13
14 }
```

toHaveLength Matcher {

```
1
2
3
4
5
6   test('arrayExample has a length of 5', () => {
7     const ARRAY_EXAMPLE = [1, 2, 3, 4, 5];
8     expect(ARRAY_EXAMPLE).toHaveLength(5);
9   });
10
11
12
13 }
14
```


toHaveProperty Matcher {

```
1
2
3     test('carObject has the expected properties', () => {
4         const CAR_OBJECT = {
5             model: 'Corolla',
6             features: {
7                 airConditioning: true
8             },
9         };
10        expect(CAR_OBJECT).toHaveProperty('model');
11        expect(CAR_OBJECT).toHaveProperty('features.airConditioning', true);
12    });
13 }
14
```

toBeDefined Matcher {

```
1
2
3
4   test('exampleObject properties are defined', () => {
5       const EXAMPLE_OBJECT = {
6         property1: 'Hello'
7       };
8       expect(EXAMPLE_OBJECT.property1).toBeDefined();
9     });
10
11
12
13
14 }
```

```
1  toBeUndefined Matcher {
```

```
2  
3  
4  
5      test('expecting undefined to be undefined', () => {  
6          expect(undefined).toBeUndefined();  
7      })  
8
```

```
9  
10  
11  
12  
13  
14  }
```

```
1  toBeInstanceOf Matcher {
2
3
4      test('isIntance of a class', () => {
5          class Coche {}
6
7          expect(new Coche()).toBeInstanceOf(Coche);
8          expect(() => {}).toBeInstanceOf(Function);
9      });
10
11
12
13
14 }
```

```
1 toBeNaN Matcher {
```

```
2  
3  
4     test('passes when value is NaN', () => {  
5         expect(NaN).toBeNaN();  
6         expect(1).not.toBeNaN();  
7     });  
8
```

```
9  
10  
11  
12  
13  
14 }
```

toContain Matcher {

```
1
2
3
4   test('arrayExample contains specific items', () => {
5       const ARRAY_EXAMPLE = ['apple', 'banana', 'orange'];
6       expect(ARRAY_EXAMPLE).toContain('banana');
7       expect(ARRAY_EXAMPLE).not.toContain('kiwi');
8   });
9
10
11
12
13
14 }
```

```
1 You can check some examples on our
2 GitHub! {
3
4
5
6
7
8
9
10
11
12
13
14 }
```

Just click the link:

https://github.com/ULL-ESIT-PAI-2023-2024/2023-2024-pai-testing-jest-juan-rs-aday-cc-ut_and_jest.git

06 {

[Tips & Tricks]

< Fancy ways to display your tests >

}

Jest Hooks {

- Special functions which execute in certain circumstances.
- They must be written inside a ***describe***, therefore, they only work inside that ***describe***.
- In Jest, we have four hooks:
 - `beforeAll()`
 - `afterAll()`
 - `beforeEach()`
 - `afterEach()`
- A hook takes a function as a parameter.

}

beforeAll & afterAll {

- Both functions will execute one time per ***describe***.
- **beforeAll** always executes first inside a ***describe***.
- **afterAll** always executes last inside a ***describe***.
- **beforeAll** is useful for setting up our tests environment (shared variables between tests ,etc.)
- **afterAll** is often used for cleaning up purposes (less used).

}

beforeAll & afterAll

```
4  ✓ describe('database connection', () => {
5      let database;
6  ✓  beforeAll(() => {
7      |   database = new Database();
8      |   database.connect();
9      | });
10
11  ✓  afterAll(() => {
12      |   database.disconnect();
13      | });
14
15  ✓  test('database should be connected', () => {
16      |   expect(database.isConnected).toBe(true);
17      | });
18
19  ✓  test('example test', () => {
20      |   expect(database.someMethod()).toBe(someExpectedValue);
21      | });
22  });
```

beforeEach & afterEach {

- Both functions will execute one time per **test**.
- **beforeEach** always executes first inside a **test**.
- **afterEach** always executes last inside a **test**.
- Both of them are useful for testing container data structures due to the frequent insertion or deletion of elements inside it.

}

beforeEach & afterEach

```
describe('list manager', () => {  
  let list;  
  beforeEach(() => {  
    list = new ListManager();  
    list.addItem('item1');  
    list.addItem('item2');  
  });  
  
  afterEach(() => {  
    list.clearList();  
  });  
  
  test('addItem should add an item to the list', () => {  
    list.addItem('item3');  
    expect(list.items).toEqual(['item1', 'item2', 'item3']);  
  });  
  
  test('removeItem should remove an item from the list', () => {  
    list.removeItem('item2');  
    expect(list.items).toEqual(['item1']);  
  });  
});
```

Jest HTML Reporter {

- Console is fine, but not very human-friendly (particularly in 2024).
- Show your test results in a kind and cute website.
- Very easy to setup.

- **Installation:**

```
npm install --save-dev jest-html-reporter
```

}

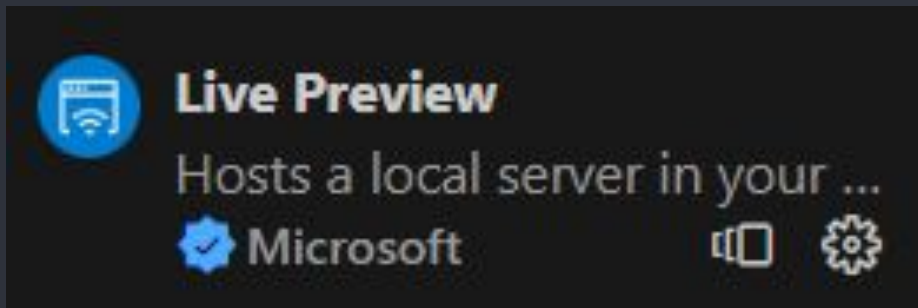
Jest HTML Reporter {

- Add the reporter to the Jest configuration in package.json:

```
"jest": {  
  "reporters": [  
    "default",  
    [  
      "./node_modules/jest-html-reporter",  
      {  
        "pageTitle": "My Amazing Test Report"  
      }  
    ]  
  ]  
}
```

Jest HTML Reporter {

- Now, everytime you execute your tests (`npm test`) a `test-report.html` file will be created.
- In VSC, we recommend Live Preview extension made by MS to show HTML files easily.
- Right Click → Show Preview.
 - Available now in either VSC or <http://localhost:3000/test-report.html>



Jest HTML Reporter

My Amazing Test Report

Started: 2024-01-30 19:17:27

Suites (2)

1 passed
1 failed
0 pending

Tests (6)

5 passed
1 failed
0 pending

√ C:\Users\Juan\Desktop\Jest_Project\tests\yet_more.test.js

0.344s

Yet more tests

subtracting 1 - 2 should return -1

passed

0.002s

Yet more tests

adding 1 + 3 should return 2

failed

0.003s

√ C:\Users\Juan\Desktop\Jest_Project\tests\calculator.test.js

0.276s

Calculator tests

adding 1 + 2 should return 3

passed

0.001s

Calculator tests

subtracting 1 - 2 should return -1

passed

0s

Calculator tests

multiplying 1 * 2 should return 2

passed

0s

Calculator tests

dividing 1 / 2 should return 0.5

passed

0s

Code coverage report {

- One of the most important metrics from a unit testing perspective.
- Measures what percentage of statements/branches are covered for the application under test.
- Even easier setup for Jest.
- **Installation**: Enable coverage report for Jest in package.json.

```
"jest": {  
  "collectCoverage": true  
}
```

Code coverage report

PASS tests/yet_more.test.js
PASS tests/calculator.test.js

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
operations.js	100	100	100	100	
yet_more_operations.js	100	100	100	100	

Code coverage report

PASS tests/yet_more.test.js
PASS tests/calculator.test.js

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	88.88	100	80	88.88	
operations.js	100	100	100	100	
yet_more_operations.js	50	100	0	50	3

```
1
2 07 {
3
4
5 [Bibliography]
6
7
8 < Our sources >
9
10
11
12 }
13
14
```

Bibliography

- History:
<https://www.techtarget.com/searchsoftwarequality/answer/Is-unit-testing-an-important-aspect-of-software-development>
- Advantages and Disadvantages:
https://en.wikipedia.org/wiki/Unit_testing
- Unit Testing: <https://testsigma.com/blog/unit-testing>
- General:
<https://www.turing.com/kb/detailed-guide-on-unit-tests-and-advantages>
- Jest: <https://www.softwaretestinghelp.com/jest-testing-tutorial>
- Documentation: <https://jestjs.io/docs>
- Exercism: <https://exercism.org/tracks/javascript>
- Jutge: <https://jutge.org>

```
1 Thanks for Watching!! {
```

```
2  
3  
4  
5     You can ask us any questions right now or here:
```

```
6     aday.cuesta.20@ull.edu.es
```

```
7  
8     juan.suarez.12@ull.edu.es
```

```
9  
10  
11  
12  
13  
14 }
```