TDD JavaScript example with QUnit: Triangle

We want to develop a simple software that based on the three sides of the triangle returns the type of the triangle based on their lengths.

For example:

```
[1, 1, 1]
```

Results in the following output:

```
“Equilateral”
```

In order to follow strictly the TDD rules we:

- Will not write any code to add functionality unless we have a failing test
- Every step should be a small and simple increment
- We can refactor code even if all the tests are green

Iteration 00

Let’s start with the testing part we will just start with an empty testing implementation referenced in the typical html test template:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Number Converter Tests</title>
  <!-- Mocha CSS -->
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/mocha/10.2.0/mocha.min.css" />
</head>
<body>
  <!-- Container where Mocha will display the test results -->
  <div id="mocha"></div>

  <!-- Load Mocha and Chai libraries -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/mocha/10.2.0/mocha.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/chai/4.3.7/chai.min.js"></script>

  <!-- Set up Mocha in BDD mode -->
  <script>
    mocha.setup('bdd');
  </script>

  <!-- Load your website library (conversion functions) -->
  <script src="script.js"></script>

  <!-- Load the test implementations -->
  <script src="tests.js"></script>

  <!-- Run the tests -->
  <script>
    mocha.run();
  </script>
</body>
</html>
```

tests.js is the file where the tests are implemented and so far is an empty file.

When this is executed (open test.html) the test is successful (no failure)

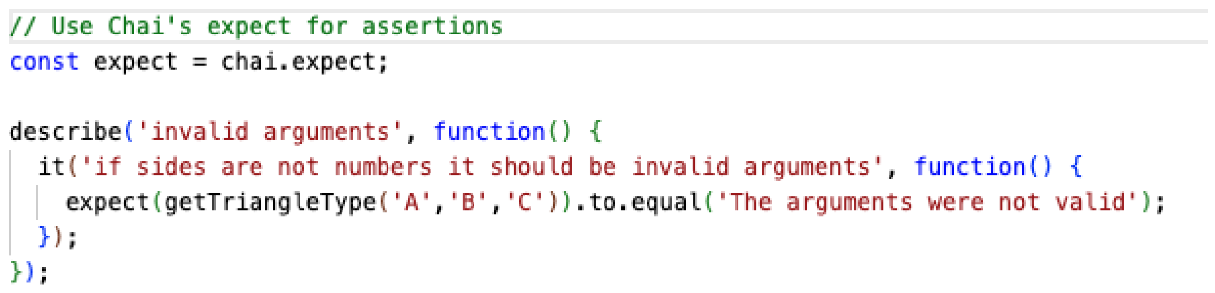
Let’s add the first skeleton of the working code to triangle.js

```javascript
function getTriangleType(lengthA, lengthB, lengthC) {
}
```

The test is still green! We cannot develop anything until we have a test that fails. Let’s do it.

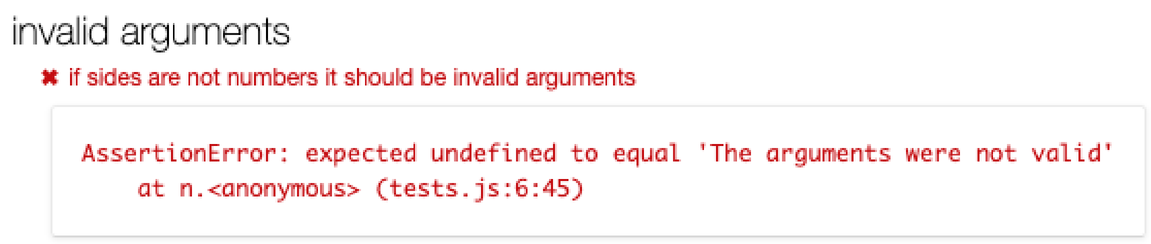
Iteration 01 - Adding a first test and a first part of the implementation

Let’s add a first test to `tests.js`.



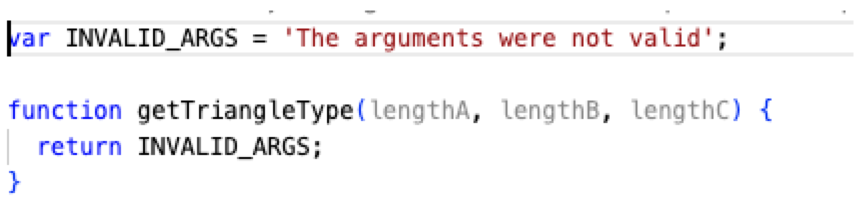
```
// Use Chai's expect for assertions
const expect = chai.expect;

describe('invalid arguments', function() {
  it('if sides are not numbers it should be invalid arguments', function() {
    expect(getTriangleType('A','B','C')).to.equal('The arguments were not valid');
  });
});
```

As `getTriangleType` is not yet defined, the tests fail as they cannot be executed:



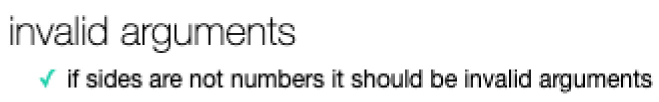
```
invalid arguments
  ✖ if sides are not numbers it should be invalid arguments

    AssertionError: expected undefined to equal 'The arguments were not valid'
        at n.<anonymous> (tests.js:6:45)
```

Now that we have a red test, let’s start adding functionality. The simplest way to make the test pass is returning always “The arguments were not valid” (INVALID_ARGS). Of course this is not what we want at the end, but it’s how TDD works, let’s do the minimal things to pass the tests we have, if the tests are defined ok, when we finish, we should a very robust implementation.



```
var INVALID_ARGS = 'The arguments were not valid';

function getTriangleType(lengthA, lengthB, lengthC) {
  return INVALID_ARGS;
}
```

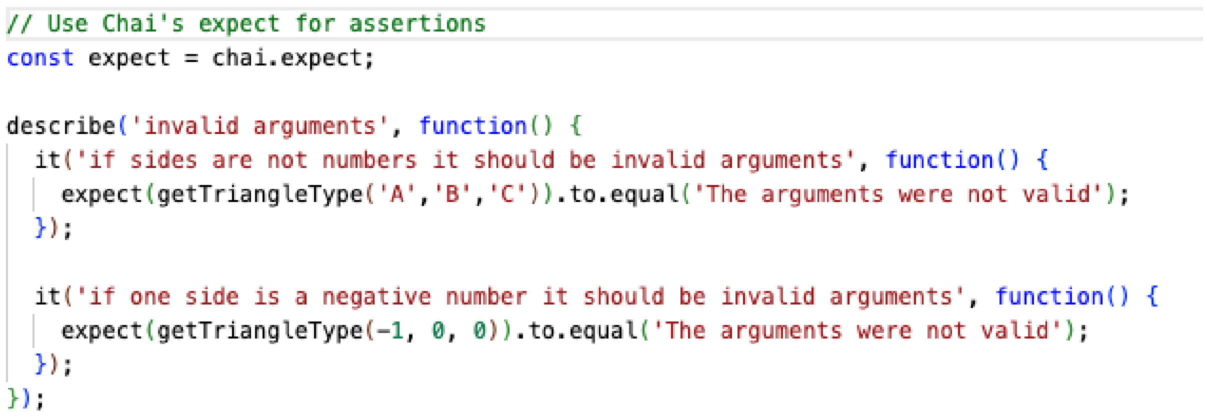
And the test should now be OK:



```
invalid arguments
  ✓ if sides are not numbers it should be invalid arguments
```

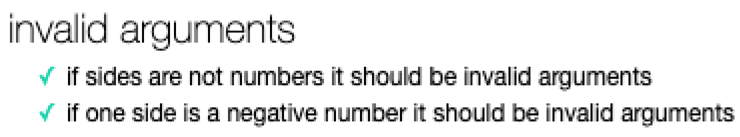
Iteration 02 - Let’s add more tests so we can add more functionality

Let’s add a second test for negative numbers to `tests.js`.



```
// Use Chai's expect for assertions
const expect = chai.expect;

describe('invalid arguments', function() {
  it('if sides are not numbers it should be invalid arguments', function() {
    expect(getTriangleType('A','B','C')).to.equal('The arguments were not valid');
  });

  it('if one side is a negative number it should be invalid arguments', function() {
    expect(getTriangleType(-1, 0, 0)).to.equal('The arguments were not valid');
  });
});
```

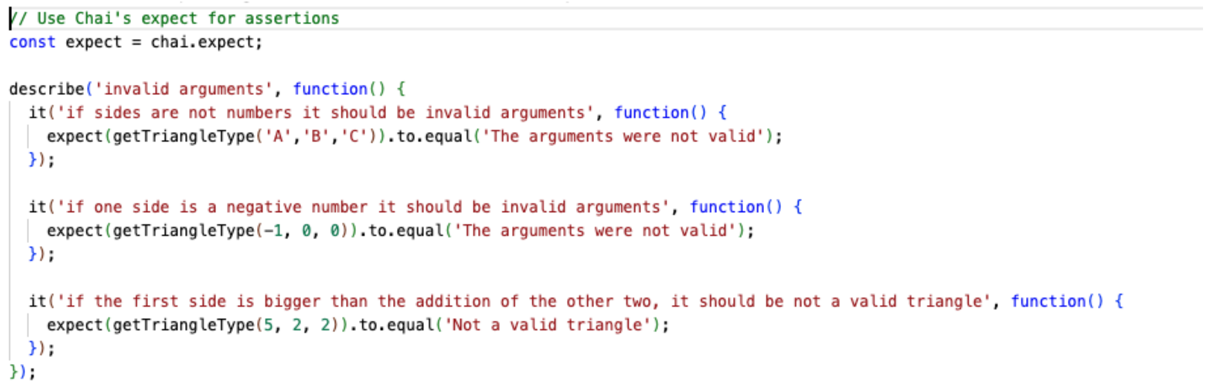
Now the test is still green because our implementation returns always INVALID_ARGS. We cannot add more code as we don’t have a “red test”.



```
invalid arguments
  ✓ if sides are not numbers it should be invalid arguments
  ✓ if one side is a negative number it should be invalid arguments
```

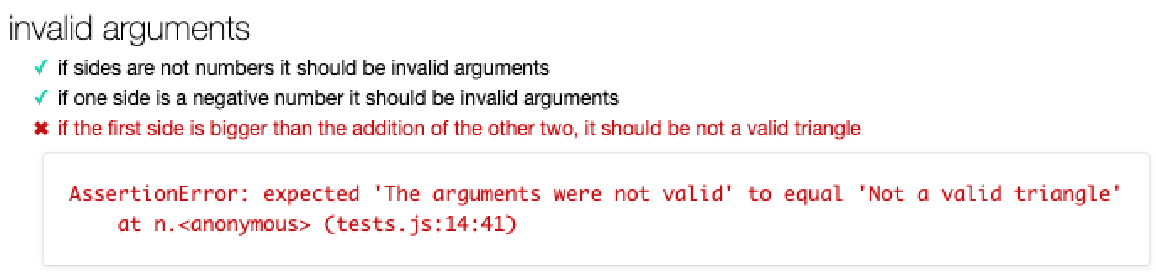
Iteration 03 - Adding support for not valid triangles

Let’s add another test for the case in which the first side is bigger than the addition of the other two.



```
// Use Chai's expect for assertions
const expect = chai.expect;

describe('invalid arguments', function() {
  it('if sides are not numbers it should be invalid arguments', function() {
    expect(getTriangleType('A','B','C')).to.equal('The arguments were not valid');
  });

  it('if one side is a negative number it should be invalid arguments', function() {
    expect(getTriangleType(-1, 0, 0)).to.equal('The arguments were not valid');
  });

  it('if the first side is bigger than the addition of the other two, it should be not a valid triangle', function() {
    expect(getTriangleType(5, 2, 2)).to.equal('Not a valid triangle');
  });
});
```

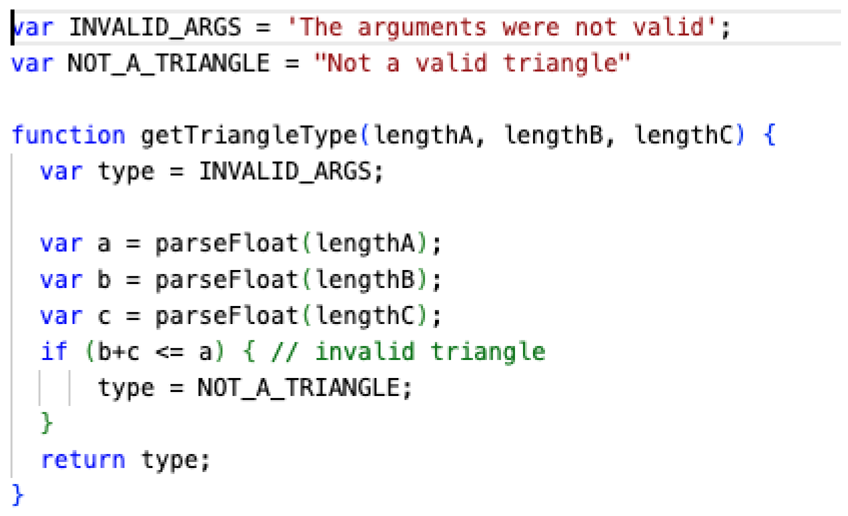
The new test fails:



```
invalid arguments
  ✓ if sides are not numbers it should be invalid arguments
  ✓ if one side is a negative number it should be invalid arguments
  ✖ if the first side is bigger than the addition of the other two, it should be not a valid triangle

    AssertionError: expected 'The arguments were not valid' to equal 'Not a valid triangle'
        at n.<anonymous> (tests.js:14:41)
```

Now that we have one test that fails we can keep adding functionality.

So it’s time to add that capability. This is the easiest way to achieve it:



```
var INVALID_ARGS = 'The arguments were not valid';
var NOT_A_TRIANGLE = "Not a valid triangle"

function getTriangleType(lengthA, lengthB, lengthC) {
  var type = INVALID_ARGS;

  var a = parseFloat(lengthA);
  var b = parseFloat(lengthB);
  var c = parseFloat(lengthC);
  if (b+c <= a) { // invalid triangle
      type = NOT_A_TRIANGLE;
  }
  return type;
}
```

All the tests are green again.

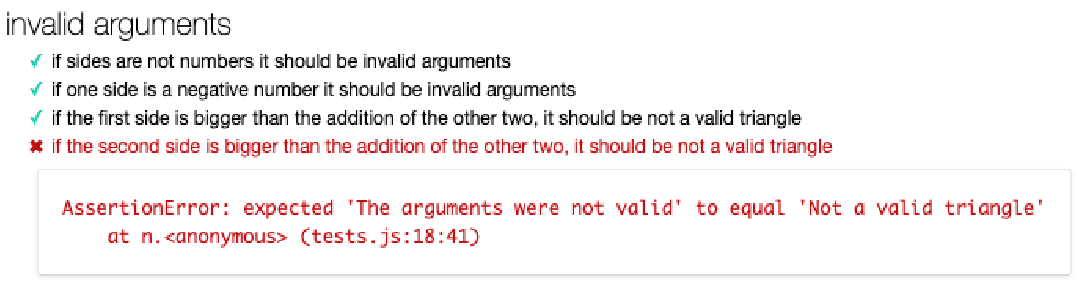
As our software is not complete we need to add new tests before we can complete the software functionality. Let’s do it step by step.

Iteration 04 - Adding support for not other type of non valid triangles

Let’s add another test for the case in which the second side is bigger than the addition of the other two.

```javascript
// Use Chai's expect for assertions
const expect = chai.expect;

describe('invalid arguments', function() {
  it('if sides are not numbers it should be invalid arguments', function() {
    expect(getTriangleType('A','B','C')).to.equal('The arguments were not valid');
  });

  it('if one side is a negative number it should be invalid arguments', function() {
    expect(getTriangleType(-1, 0, 0)).to.equal('The arguments were not valid');
  });

  it('if the first side is bigger than the addition of the other two, it should be not a valid triangle', function() {
    expect(getTriangleType(5, 2, 2)).to.equal('Not a valid triangle');
  });

  it('if the second side is bigger than the addition of the other two, it should be not a valid triangle', function() {
    expect(getTriangleType(1, 8, 6)).to.equal('Not a valid triangle');
  });
});
```

The last test makes the implementation fail:



```
invalid arguments
  ✓ if sides are not numbers it should be invalid arguments
  ✓ if one side is a negative number it should be invalid arguments
  ✓ if the first side is bigger than the addition of the other two, it should be not a valid triangle
  ✖ if the second side is bigger than the addition of the other two, it should be not a valid triangle

    AssertionError: expected 'The arguments were not valid' to equal 'Not a valid triangle'
        at n.<anonymous> (tests.js:18:41)
```

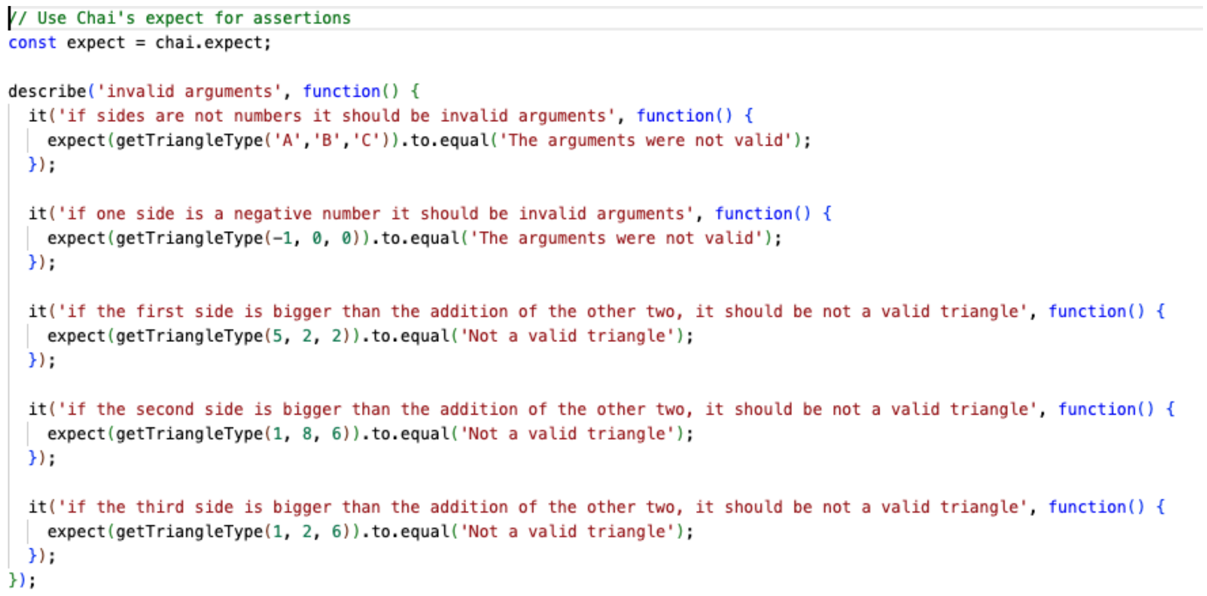
The following is the smallest change to make the code works. Please note that I needed to add a new checking (a<=0). This test has spotted that in some cases a+c <=b and at the same time, a could be a negative number. Without this new change, the test (-1,0,0) failed. This is one of the advantages of having automated the test before.

```javascript
var INVALID_ARGS = 'The arguments were not valid';
var NOT_A_TRIANGLE = "Not a valid triangle"

function getTriangleType(lengthA, lengthB, lengthC) {
  var type = INVALID_ARGS;

  var a = parseFloat(lengthA);
  var b = parseFloat(lengthB);
  var c = parseFloat(lengthC);

  if (a <= 0) {
      type = INVALID_ARGS;
  } else if ((b+c <= a) || (a+c <= b)) { // invalid triangle
      type = NOT_A_TRIANGLE;
  }
  return type;
}
```

Now the tests are all green. Time for adding a new failing test.

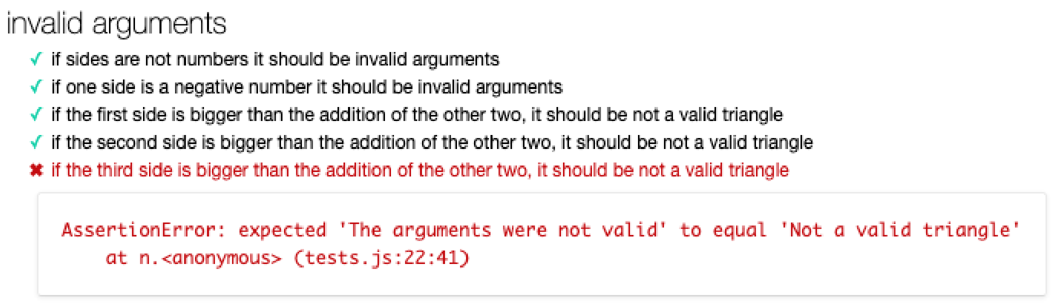
Iteration 05 - Adding support for the remaining case of non valid triangles

Let’s add another test for the case in which the third side is bigger than the addition of the other two.



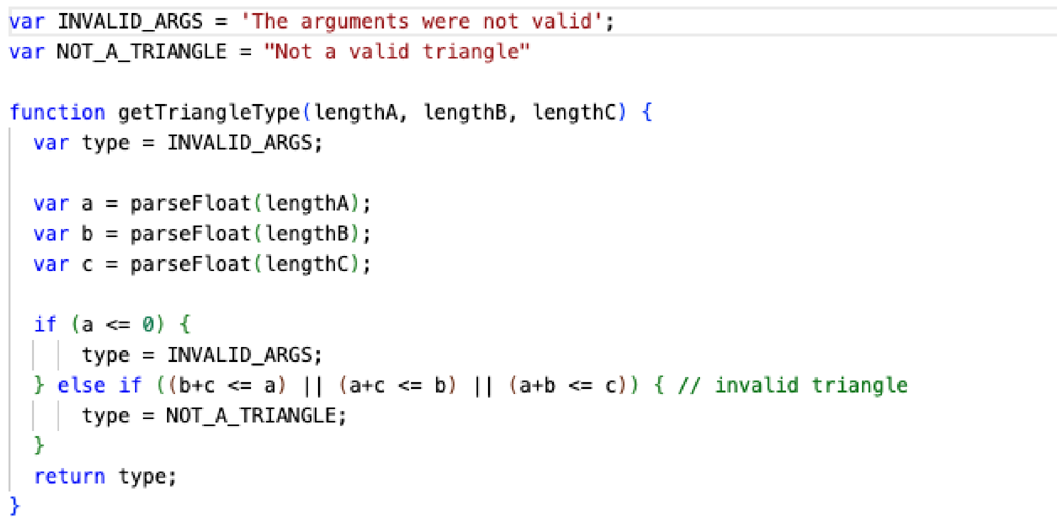
```
// Use Chai's expect for assertions
const expect = chai.expect;

describe('invalid arguments', function() {
  it('if sides are not numbers it should be invalid arguments', function() {
    expect(getTriangleType('A','B','C')).to.equal('The arguments were not valid');
  });

  it('if one side is a negative number it should be invalid arguments', function() {
    expect(getTriangleType(-1, 0, 0)).to.equal('The arguments were not valid');
  });

  it('if the first side is bigger than the addition of the other two, it should be not a valid triangle', function() {
    expect(getTriangleType(5, 2, 2)).to.equal('Not a valid triangle');
  });

  it('if the second side is bigger than the addition of the other two, it should be not a valid triangle', function() {
    expect(getTriangleType(1, 8, 6)).to.equal('Not a valid triangle');
  });

  it('if the third side is bigger than the addition of the other two, it should be not a valid triangle', function() {
    expect(getTriangleType(1, 2, 6)).to.equal('Not a valid triangle');
  });
});
```

The test makes the software fail:



```
invalid arguments
  ✓ if sides are not numbers it should be invalid arguments
  ✓ if one side is a negative number it should be invalid arguments
  ✓ if the first side is bigger than the addition of the other two, it should be not a valid triangle
  ✓ if the second side is bigger than the addition of the other two, it should be not a valid triangle
  ✖ if the third side is bigger than the addition of the other two, it should be not a valid triangle

    AssertionError: expected 'The arguments were not valid' to equal 'Not a valid triangle'
        at n.<anonymous> (tests.js:22:41)
```

Time to implement that condition:

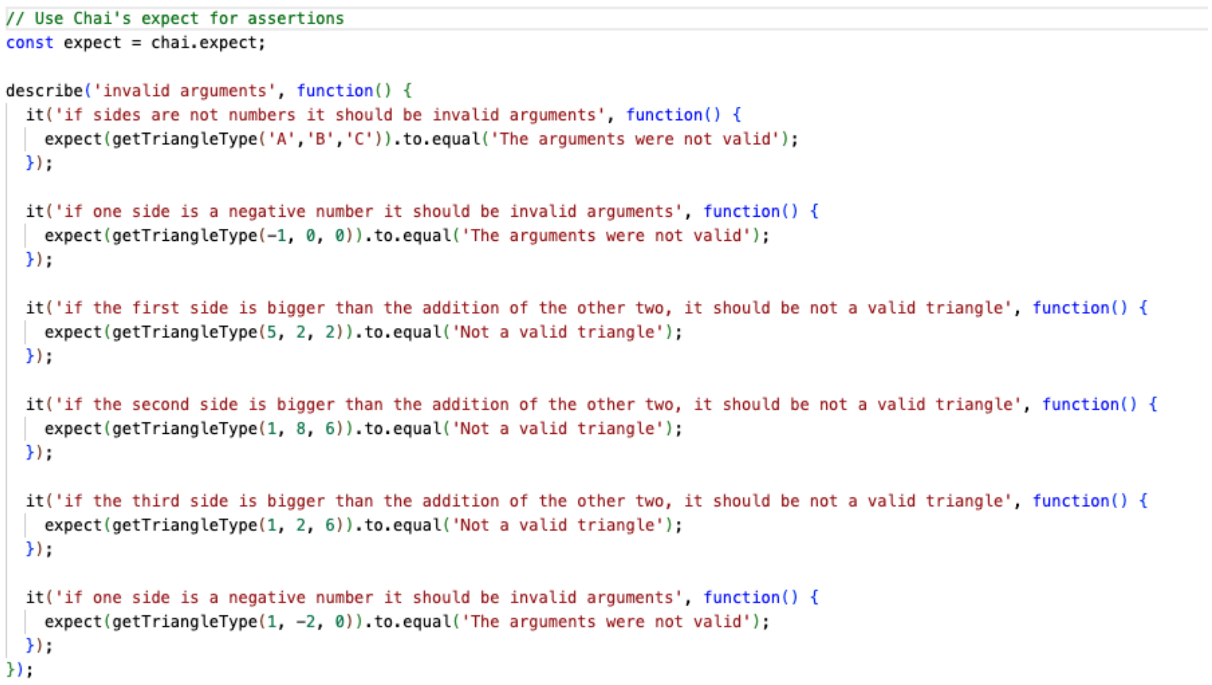


```
var INVALID_ARGS = 'The arguments were not valid';
var NOT_A_TRIANGLE = "Not a valid triangle"

function getTriangleType(lengthA, lengthB, lengthC) {
  var type = INVALID_ARGS;

  var a = parseFloat(lengthA);
  var b = parseFloat(lengthB);
  var c = parseFloat(lengthC);

  if (a <= 0) {
      type = INVALID_ARGS;
  } else if ((b+c <= a) || (a+c <= b) || (a+b <= c)) { // invalid triangle
      type = NOT_A_TRIANGLE;
  }
  return type;
}
```

Tests are all green again. Let’s add new failing one.

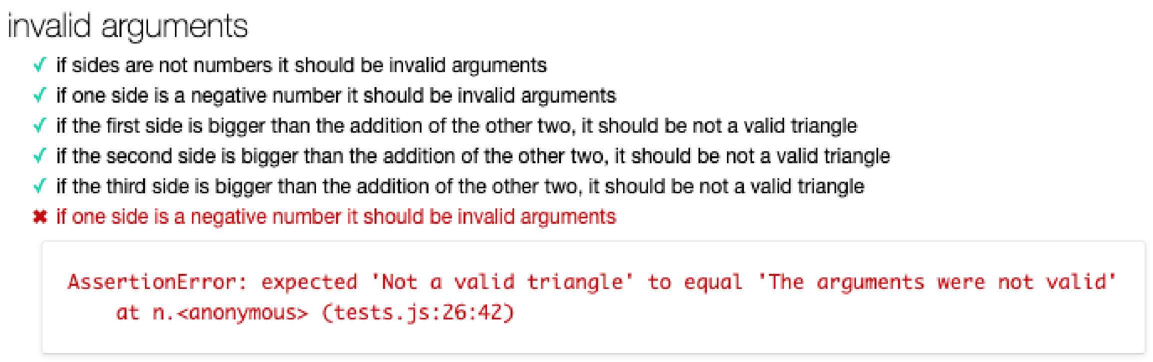
Iteration 06 - Second side is negative

The problem I faced when I added the condition a<=0 has made me think that I need to check also for b and c. However, I don’t have any negative tests for them. Let’s add cases for those situations one by one.



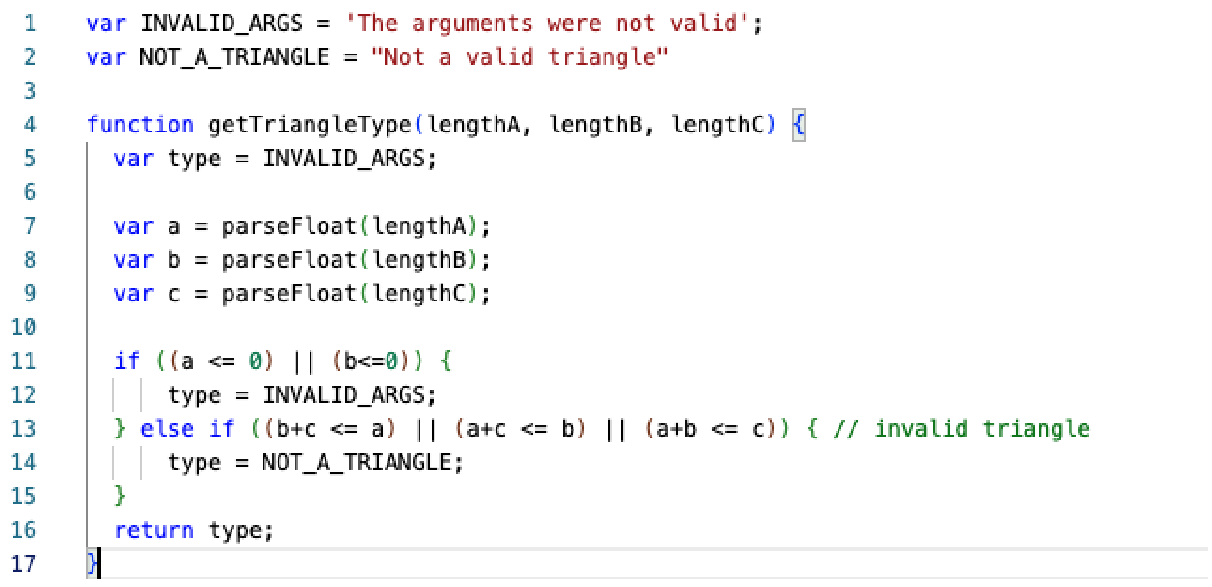
```
// Use Chai's expect for assertions
const expect = chai.expect;

describe('invalid arguments', function() {
  it('if sides are not numbers it should be invalid arguments', function() {
    expect(getTriangleType('A','B','C')).to.equal('The arguments were not valid');
  });

  it('if one side is a negative number it should be invalid arguments', function() {
    expect(getTriangleType(-1, 0, 0)).to.equal('The arguments were not valid');
  });

  it('if the first side is bigger than the addition of the other two, it should be not a valid triangle', function() {
    expect(getTriangleType(5, 2, 2)).to.equal('Not a valid triangle');
  });

  it('if the second side is bigger than the addition of the other two, it should be not a valid triangle', function() {
    expect(getTriangleType(1, 8, 6)).to.equal('Not a valid triangle');
  });

  it('if the third side is bigger than the addition of the other two, it should be not a valid triangle', function() {
    expect(getTriangleType(1, 2, 6)).to.equal('Not a valid triangle');
  });

  it('if one side is a negative number it should be invalid arguments', function() {
    expect(getTriangleType(1, -2, 0)).to.equal('The arguments were not valid');
  });
});
```

I made it fail:



```
invalid arguments
  ✓ if sides are not numbers it should be invalid arguments
  ✓ if one side is a negative number it should be invalid arguments
  ✓ if the first side is bigger than the addition of the other two, it should be not a valid triangle
  ✓ if the second side is bigger than the addition of the other two, it should be not a valid triangle
  ✓ if the third side is bigger than the addition of the other two, it should be not a valid triangle
  ✖ if one side is a negative number it should be invalid arguments

    AssertionError: expected 'Not a valid triangle' to equal 'The arguments were not valid'
        at n.<anonymous> (tests.js:26:42)
```

So now I can add the corresponding implementation:

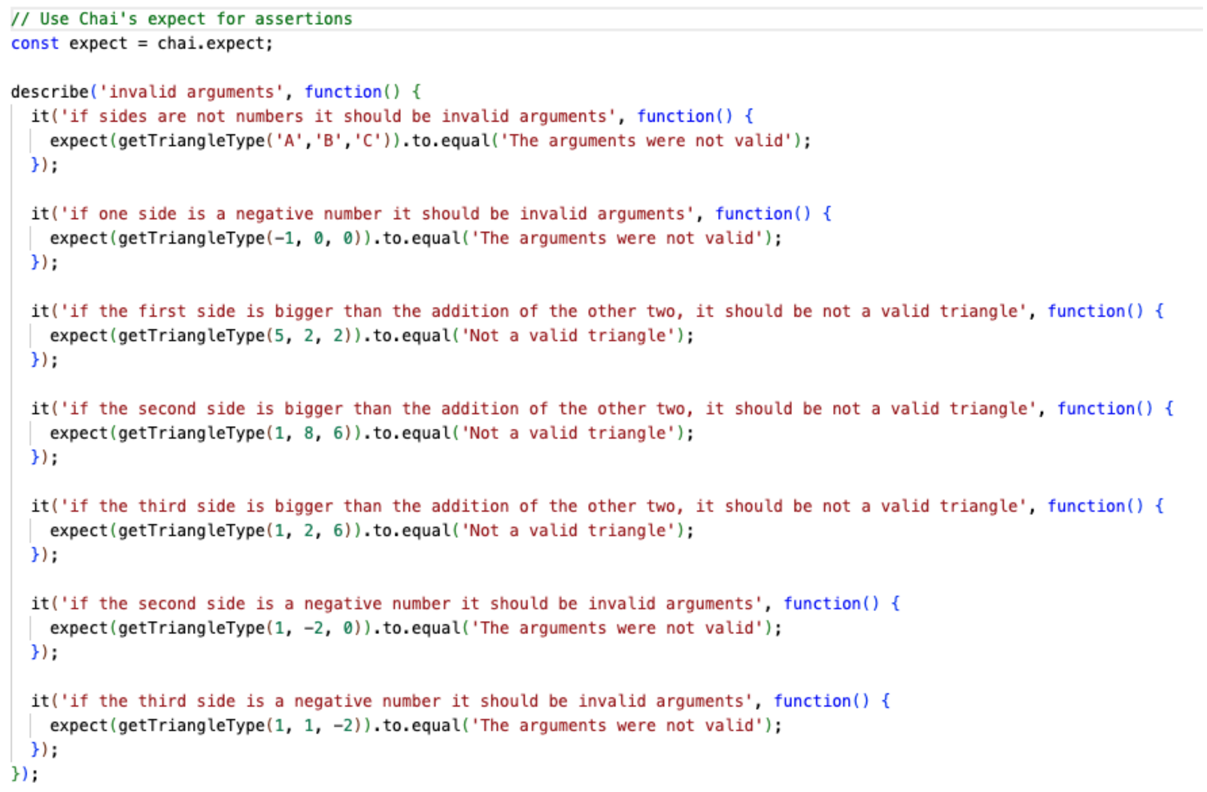


```javascript
var INVALID_ARGS = 'The arguments were not valid';
var NOT_A_TRIANGLE = "Not a valid triangle"

function getTriangleType(lengthA, lengthB, lengthC) {
  var type = INVALID_ARGS;

  var a = parseFloat(lengthA);
  var b = parseFloat(lengthB);
  var c = parseFloat(lengthC);

  if ((a <= 0) || (b<=0)) {
    type = INVALID_ARGS;
  } else if ((b+c <= a) || (a+c <= b) || (a+b <= c)) { // invalid triangle
    type = NOT_A_TRIANGLE;
  }
  return type;
}
```

Iteration 07 - Third side is negative

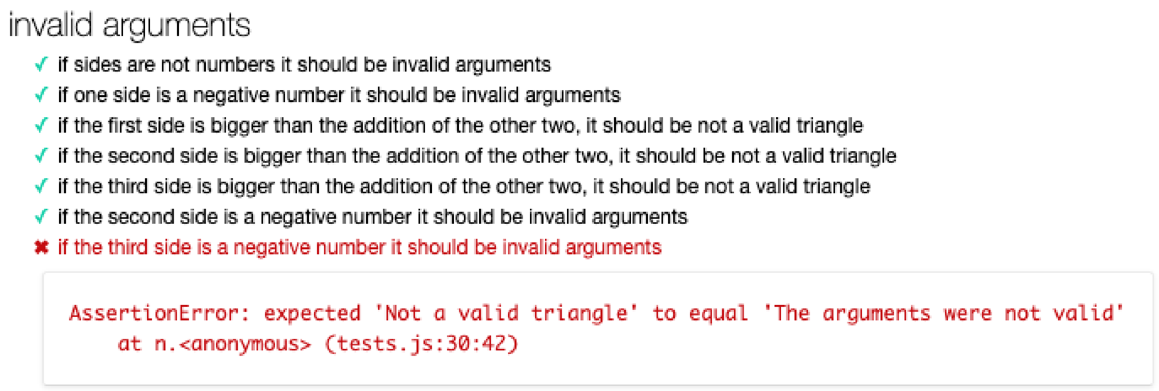
The same that in the previous one but for the third side.

New test:



```javascript
// Use Chai's expect for assertions
const expect = chai.expect;

describe('invalid arguments', function() {
  it('if sides are not numbers it should be invalid arguments', function() {
    expect(getTriangleType('A','B','C')).to.equal('The arguments were not valid');
  });

  it('if one side is a negative number it should be invalid arguments', function() {
    expect(getTriangleType(-1, 0, 0)).to.equal('The arguments were not valid');
  });

  it('if the first side is bigger than the addition of the other two, it should be not a valid triangle', function() {
    expect(getTriangleType(5, 2, 2)).to.equal('Not a valid triangle');
  });

  it('if the second side is bigger than the addition of the other two, it should be not a valid triangle', function() {
    expect(getTriangleType(1, 8, 6)).to.equal('Not a valid triangle');
  });

  it('if the third side is bigger than the addition of the other two, it should be not a valid triangle', function() {
    expect(getTriangleType(1, 2, 6)).to.equal('Not a valid triangle');
  });

  it('if the second side is a negative number it should be invalid arguments', function() {
    expect(getTriangleType(1, -2, 0)).to.equal('The arguments were not valid');
  });

  it('if the third side is a negative number it should be invalid arguments', function() {
    expect(getTriangleType(1, 1, -2)).to.equal('The arguments were not valid');
  });
});
```

Execution fails:



```
invalid arguments
  ✓ if sides are not numbers it should be invalid arguments
  ✓ if one side is a negative number it should be invalid arguments
  ✓ if the first side is bigger than the addition of the other two, it should be not a valid triangle
  ✓ if the second side is bigger than the addition of the other two, it should be not a valid triangle
  ✓ if the third side is bigger than the addition of the other two, it should be not a valid triangle
  ✓ if the second side is a negative number it should be invalid arguments
  ✖ if the third side is a negative number it should be invalid arguments

    AssertionError: expected 'Not a valid triangle' to equal 'The arguments were not valid'
        at n.<anonymous> (tests.js:30:42)
```

Implementation:

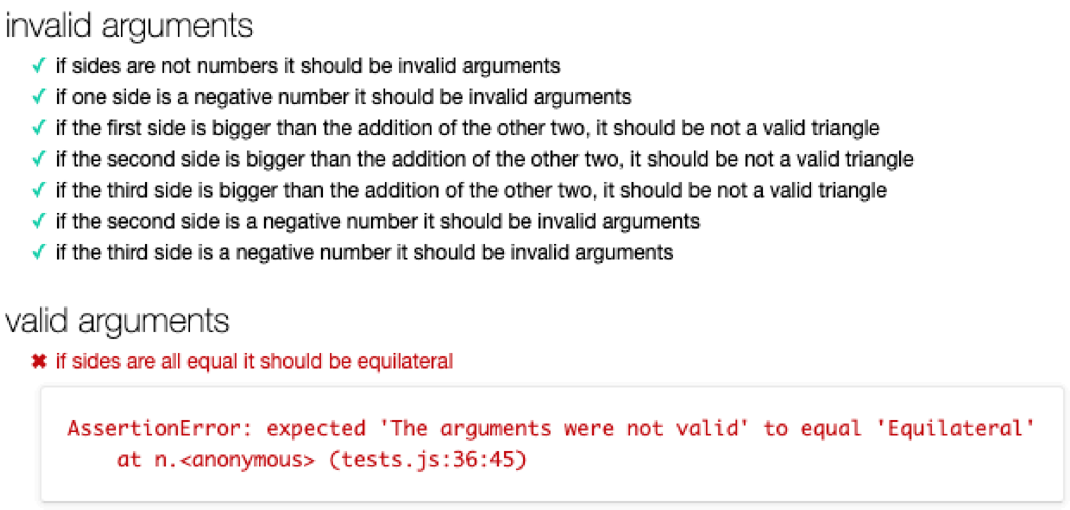
```javascript
var INVALID_ARGS = 'The arguments were not valid';
var NOT_A_TRIANGLE = "Not a valid triangle"

function getTriangleType(lengthA, lengthB, lengthC) {
  var type = INVALID_ARGS;

  var a = parseFloat(lengthA);
  var b = parseFloat(lengthB);
  var c = parseFloat(lengthC);

  if ((a <= 0) || (b<=0) || (c<=0)) {
    type = INVALID_ARGS;
  } else if ((b+c <= a) || (a+c <= b) || (a+b <= c)) { // invalid triangle
    type = NOT_A_TRIANGLE;
  }
  return type;
}
```

Iteration 08 - Equilateral

Let's add a test for equilateral triangles:

```javascript
describe('valid arguments', function() {
  it('if sides are all equal it should be equilateral', function() {
    expect(getTriangleType('2','2','2')).to.equal('Equilateral');
  });
});
```

So that if fails…



invalid arguments

✓ if sides are not numbers it should be invalid arguments
✓ if one side is a negative number it should be invalid arguments
✓ if the first side is bigger than the addition of the other two, it should be not a valid triangle
✓ if the second side is bigger than the addition of the other two, it should be not a valid triangle
✓ if the third side is bigger than the addition of the other two, it should be not a valid triangle
✓ if the second side is a negative number it should be invalid arguments
✓ if the third side is a negative number it should be invalid arguments

valid arguments

✖ if sides are all equal it should be equilateral

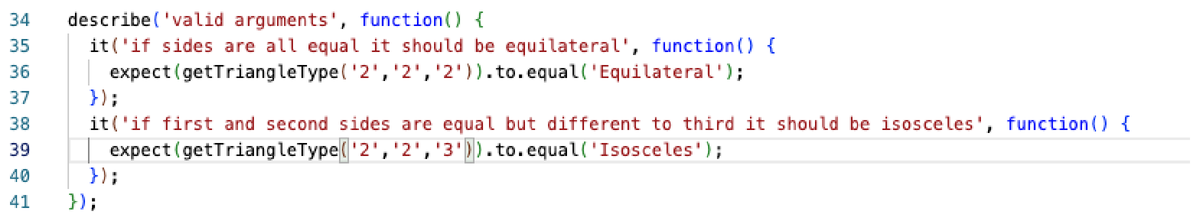
```
AssertionError: expected 'The arguments were not valid' to equal 'Equilateral'
    at n.<anonymous> (tests.js:36:45)
```

And we can add the most basic equilateral implementation:

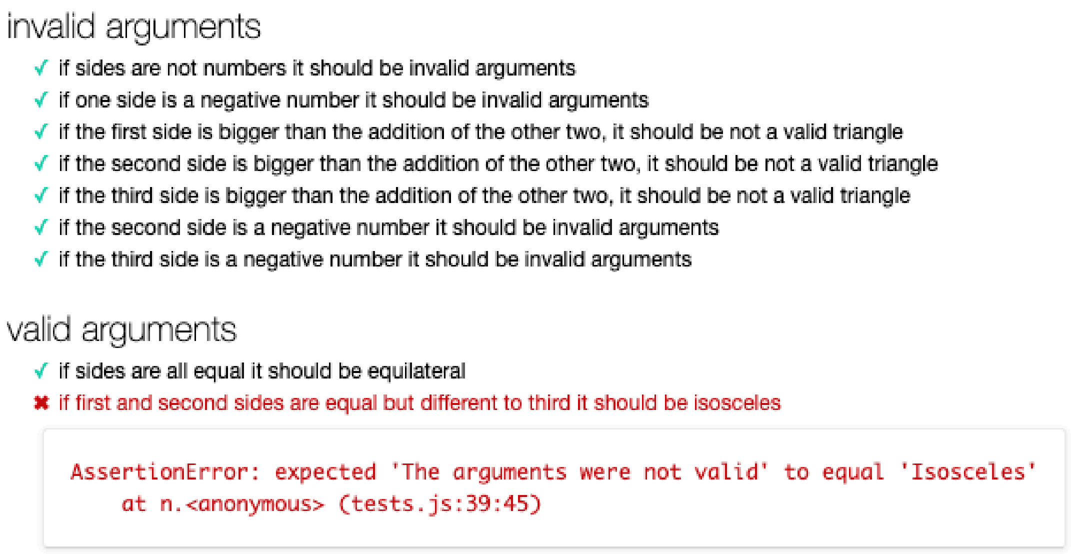
```javascript
var INVALID_ARGS = 'The arguments were not valid';
var NOT_A_TRIANGLE = "Not a valid triangle";
var EQUILATERAL = "Equilateral";

function getTriangleType(lengthA, lengthB, lengthC) {
  var type = INVALID_ARGS;

  var a = parseFloat(lengthA);
  var b = parseFloat(lengthB);
  var c = parseFloat(lengthC);

  if ((a <= 0) || (b<=0) || (c<=0)) {
      type = INVALID_ARGS;
  } else if ((b+c <= a) || (a+c <= b) || (a+b <= c)) { // invalid triangle
      type = NOT_A_TRIANGLE;
  } else if ((a===b) && (b===c)) {
    type = EQUILATERAL;
  }
  return type;
}
```

Iteration 09 - Isosceles - A/B equal. C different

The following three iteration are quite similar. Add one test, fails, add implementation to make the test pass.



```
34  describe('valid arguments', function() {
35    it('if sides are all equal it should be equilateral', function() {
36      expect(getTriangleType('2','2','2')).to.equal('Equilateral');
37    });
38    it('if first and second sides are equal but different to third it should be isosceles', function() {
39      expect(getTriangleType('2','2','3')).to.equal('Isosceles');
40    });
41  });
```



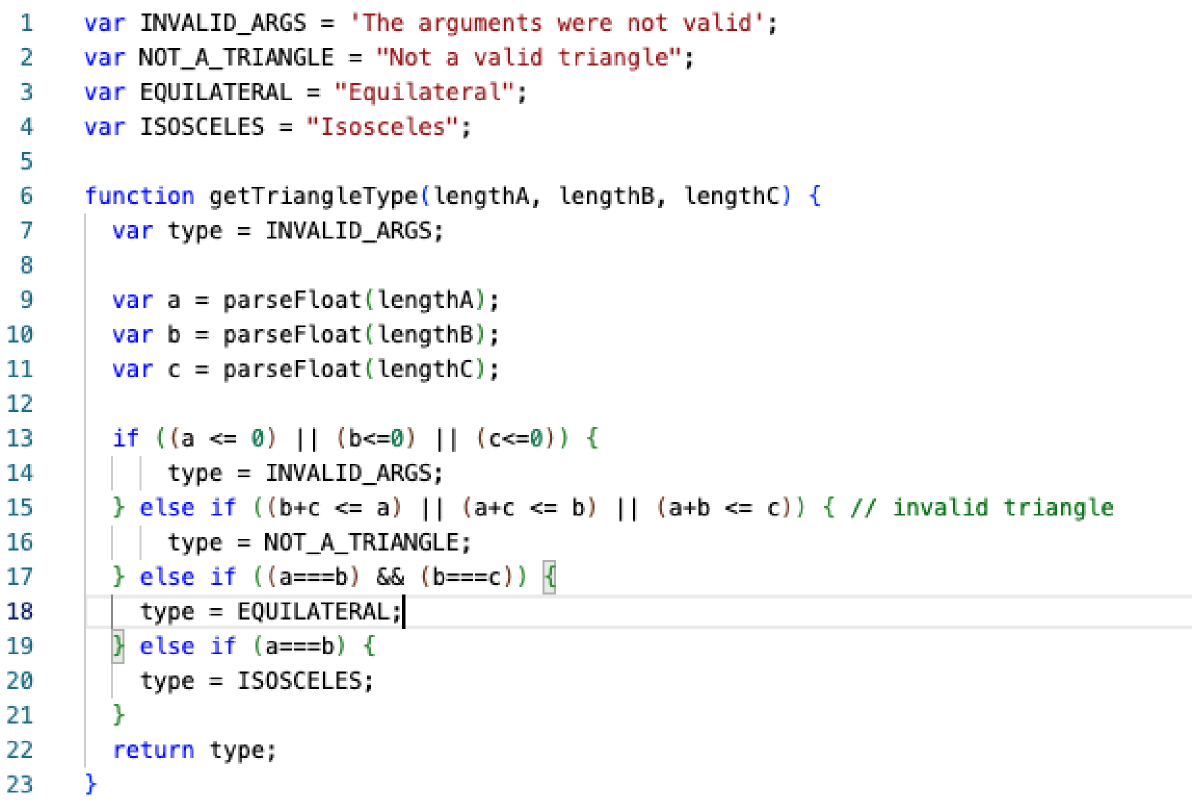
invalid arguments

- ✓ if sides are not numbers it should be invalid arguments
- ✓ if one side is a negative number it should be invalid arguments
- ✓ if the first side is bigger than the addition of the other two, it should be not a valid triangle
- ✓ if the second side is bigger than the addition of the other two, it should be not a valid triangle
- ✓ if the third side is bigger than the addition of the other two, it should be not a valid triangle
- ✓ if the second side is a negative number it should be invalid arguments
- ✓ if the third side is a negative number it should be invalid arguments

valid arguments

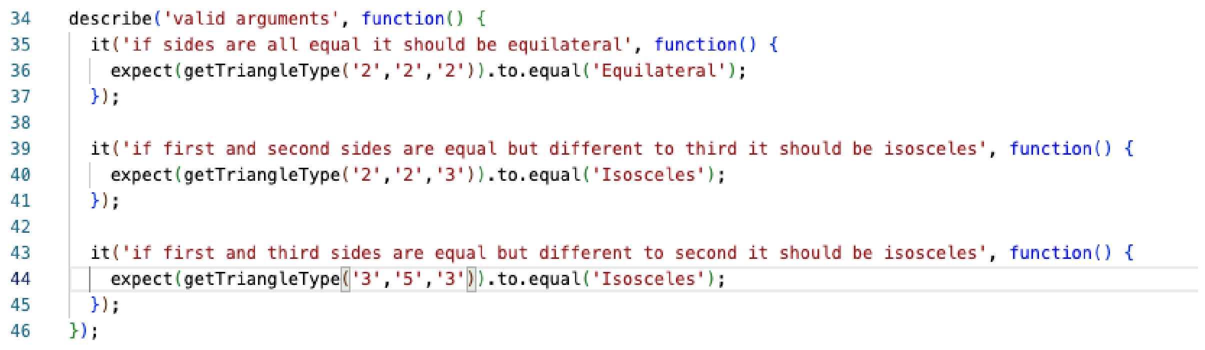
- ✓ if sides are all equal it should be equilateral
- ✖ if first and second sides are equal but different to third it should be isosceles

```
AssertionError: expected 'The arguments were not valid' to equal 'Isosceles'
    at n.<anonymous> (tests.js:39:45)
```

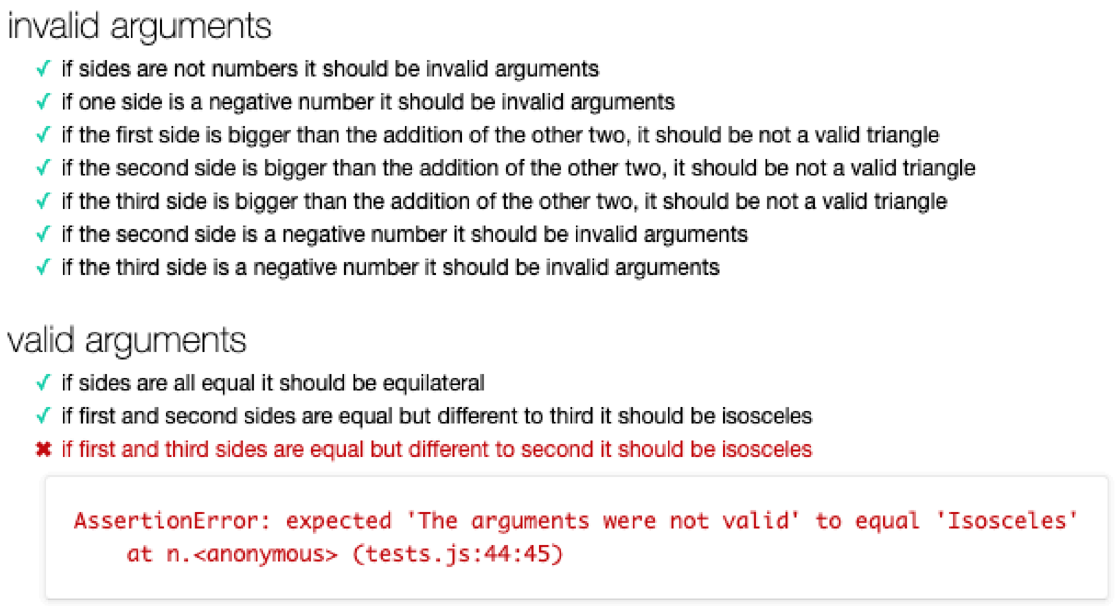


```
var INVALID_ARGS = 'The arguments were not valid';
var NOT_A_TRIANGLE = "Not a valid triangle";
var EQUILATERAL = "Equilateral";
var ISOSCELES = "Isosceles";

function getTriangleType(lengthA, lengthB, lengthC) {
  var type = INVALID_ARGS;

  var a = parseFloat(lengthA);
  var b = parseFloat(lengthB);
  var c = parseFloat(lengthC);

  if ((a <= 0) || (b<=0) || (c<=0)) {
      type = INVALID_ARGS;
  } else if ((b+c <= a) || (a+c <= b) || (a+b <= c)) { // invalid triangle
      type = NOT_A_TRIANGLE;
  } else if ((a===b) && (b===c)) {
    type = EQUILATERAL;
  } else if (a===b) {
    type = ISOSCELES;
  }
  return type;
}
```

Iteration 10 - Isosceles - A/C equal. B different



```
34  describe('valid arguments', function() {
35    it('if sides are all equal it should be equilateral', function() {
36      expect(getTriangleType('2','2','2')).to.equal('Equilateral');
37    });
38
39    it('if first and second sides are equal but different to third it should be isosceles', function() {
40      expect(getTriangleType('2','2','3')).to.equal('Isosceles');
41    });
42
43    it('if first and third sides are equal but different to second it should be isosceles', function() {
44      expect(getTriangleType('3','5','3')).to.equal('Isosceles');
45    });
46  });
```



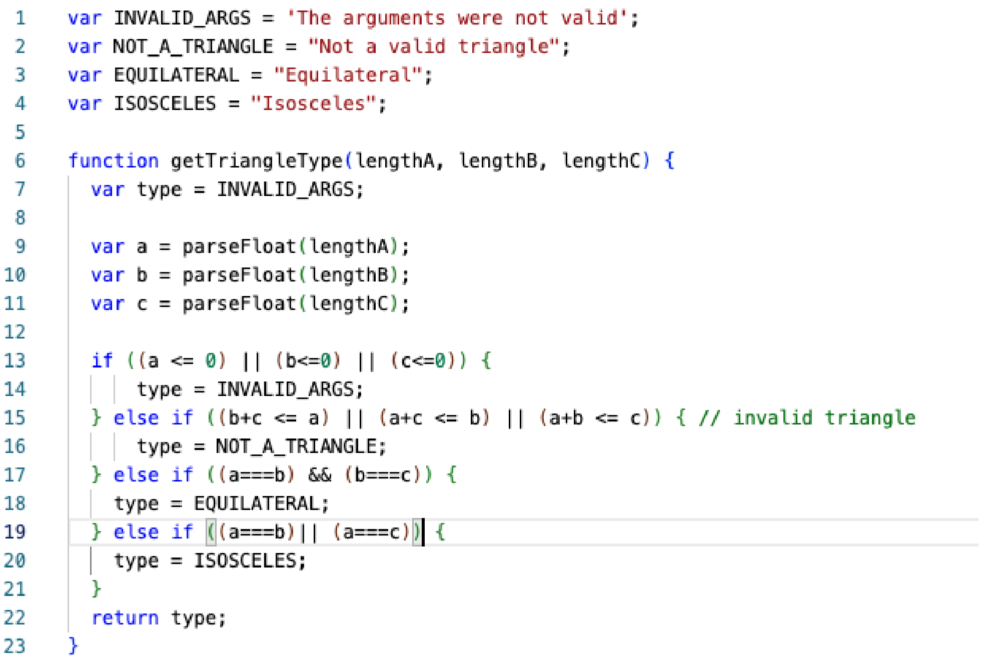
invalid arguments

- ✓ if sides are not numbers it should be invalid arguments
- ✓ if one side is a negative number it should be invalid arguments
- ✓ if the first side is bigger than the addition of the other two, it should be not a valid triangle
- ✓ if the second side is bigger than the addition of the other two, it should be not a valid triangle
- ✓ if the third side is bigger than the addition of the other two, it should be not a valid triangle
- ✓ if the second side is a negative number it should be invalid arguments
- ✓ if the third side is a negative number it should be invalid arguments

valid arguments

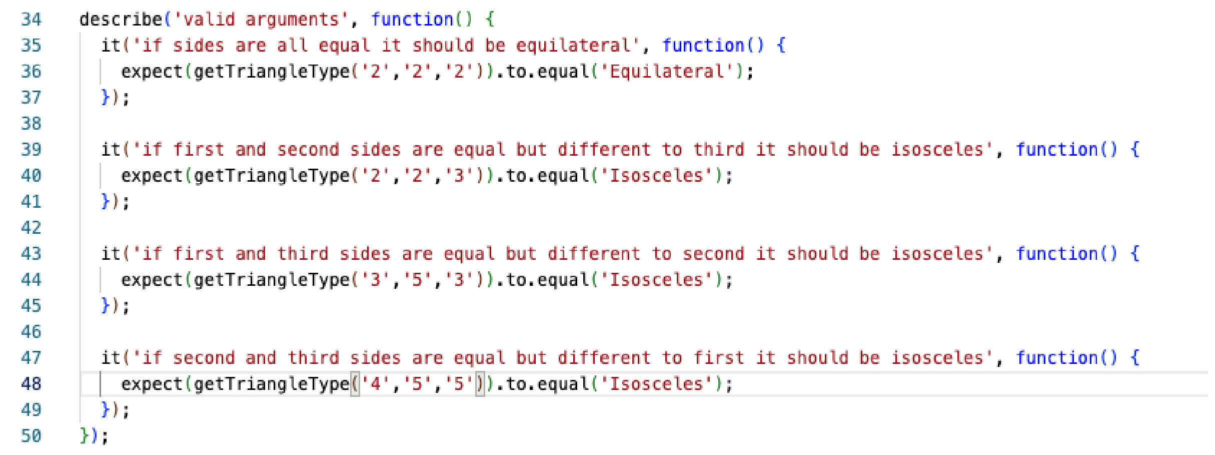
- ✓ if sides are all equal it should be equilateral
- ✓ if first and second sides are equal but different to third it should be isosceles
- ✖ if first and third sides are equal but different to second it should be isosceles

```
AssertionError: expected 'The arguments were not valid' to equal 'Isosceles'
    at n.<anonymous> (tests.js:44:45)
```

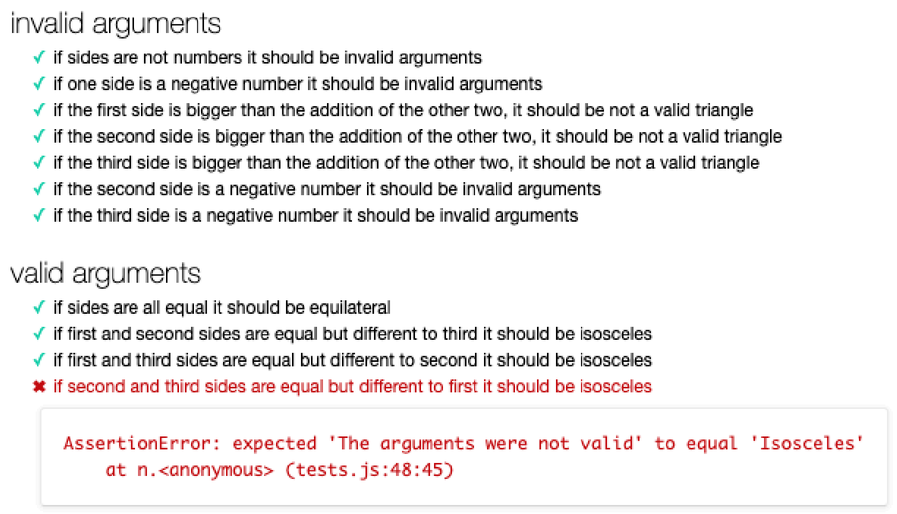


```
var INVALID_ARGS = 'The arguments were not valid';
var NOT_A_TRIANGLE = "Not a valid triangle";
var EQUILATERAL = "Equilateral";
var ISOSCELES = "Isosceles";

function getTriangleType(lengthA, lengthB, lengthC) {
  var type = INVALID_ARGS;

  var a = parseFloat(lengthA);
  var b = parseFloat(lengthB);
  var c = parseFloat(lengthC);

  if ((a <= 0) || (b<=0) || (c<=0)) {
      type = INVALID_ARGS;
  } else if ((b+c <= a) || (a+c <= b) || (a+b <= c)) { // invalid triangle
      type = NOT_A_TRIANGLE;
  } else if ((a===b) && (b===c)) {
    type = EQUILATERAL;
  } else if ((a===b)|| (a===c)) {
    type = ISOSCELES;
  }
  return type;
}
```

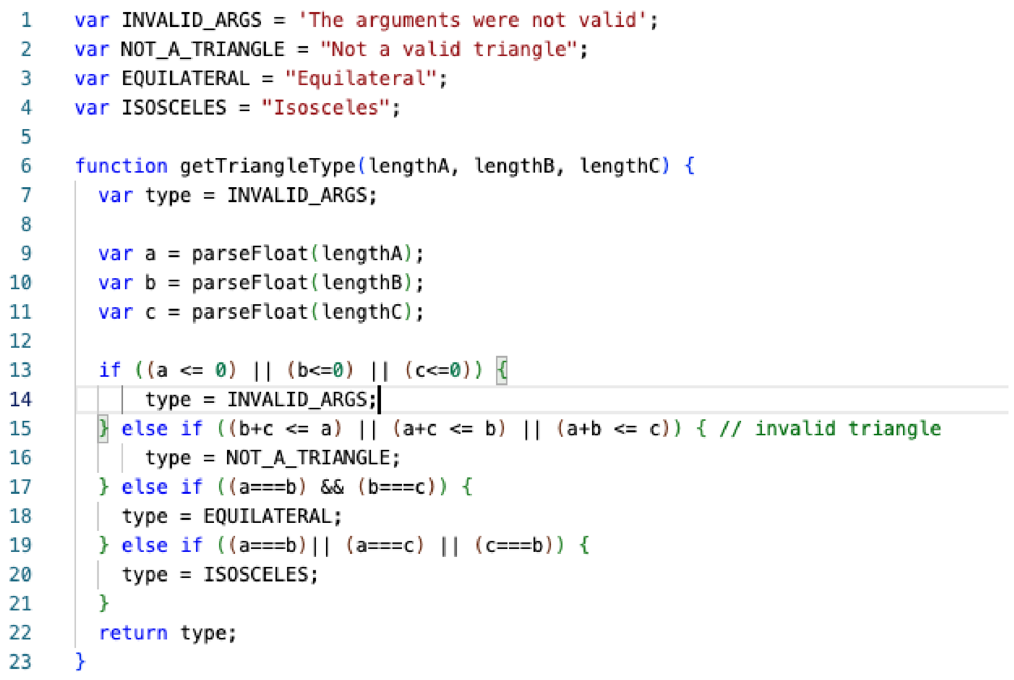
Iteration 11 - Isosceles - B/C equal. A different



```
34  describe('valid arguments', function() {
35    it('if sides are all equal it should be equilateral', function() {
36      expect(getTriangleType('2','2','2')).to.equal('Equilateral');
37    });
38
39    it('if first and second sides are equal but different to third it should be isosceles', function() {
40      expect(getTriangleType('2','2','3')).to.equal('Isosceles');
41    });
42
43    it('if first and third sides are equal but different to second it should be isosceles', function() {
44      expect(getTriangleType('3','5','3')).to.equal('Isosceles');
45    });
46
47    it('if second and third sides are equal but different to first it should be isosceles', function() {
48      expect(getTriangleType('4','5','5')).to.equal('Isosceles');
49    });
50  });
```



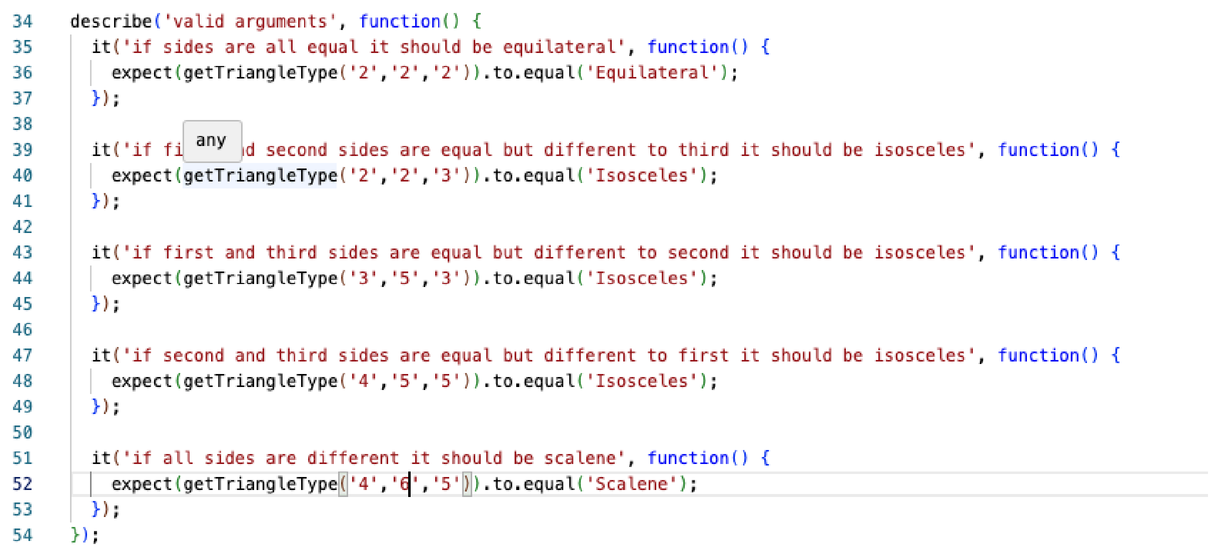
```
invalid arguments
  ✓ if sides are not numbers it should be invalid arguments
  ✓ if one side is a negative number it should be invalid arguments
  ✓ if the first side is bigger than the addition of the other two, it should be not a valid triangle
  ✓ if the second side is bigger than the addition of the other two, it should be not a valid triangle
  ✓ if the third side is bigger than the addition of the other two, it should be not a valid triangle
  ✓ if the second side is a negative number it should be invalid arguments
  ✓ if the third side is a negative number it should be invalid arguments

valid arguments
  ✓ if sides are all equal it should be equilateral
  ✓ if first and second sides are equal but different to third it should be isosceles
  ✓ if first and third sides are equal but different to second it should be isosceles
  ✖ if second and third sides are equal but different to first it should be isosceles

    AssertionError: expected 'The arguments were not valid' to equal 'Isosceles'
        at n.<anonymous> (tests.js:48:45)
```



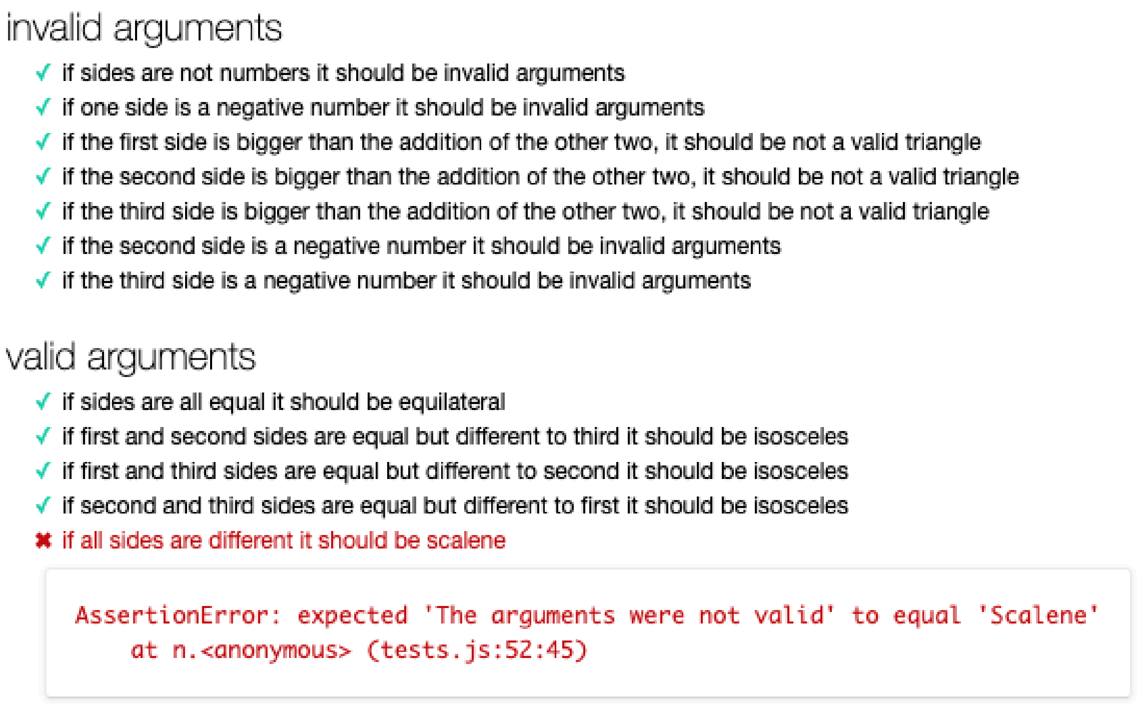
```
var INVALID_ARGS = 'The arguments were not valid';
var NOT_A_TRIANGLE = "Not a valid triangle";
var EQUILATERAL = "Equilateral";
var ISOSCELES = "Isosceles";

function getTriangleType(lengthA, lengthB, lengthC) {
  var type = INVALID_ARGS;

  var a = parseFloat(lengthA);
  var b = parseFloat(lengthB);
  var c = parseFloat(lengthC);

  if ((a <= 0) || (b<=0) || (c<=0)) {
    type = INVALID_ARGS;
  } else if ((b+c <= a) || (a+c <= b) || (a+b <= c)) { // invalid triangle
    type = NOT_A_TRIANGLE;
  } else if ((a===b) && (b===c)) {
    type = EQUILATERAL;
  } else if ((a===b)|| (a===c) || (c===b)) {
    type = ISOSCELES;
  }
  return type;
}
```

Iteration 12 - Scalene

When the test for scalene is added, the implementation fails in the last test.

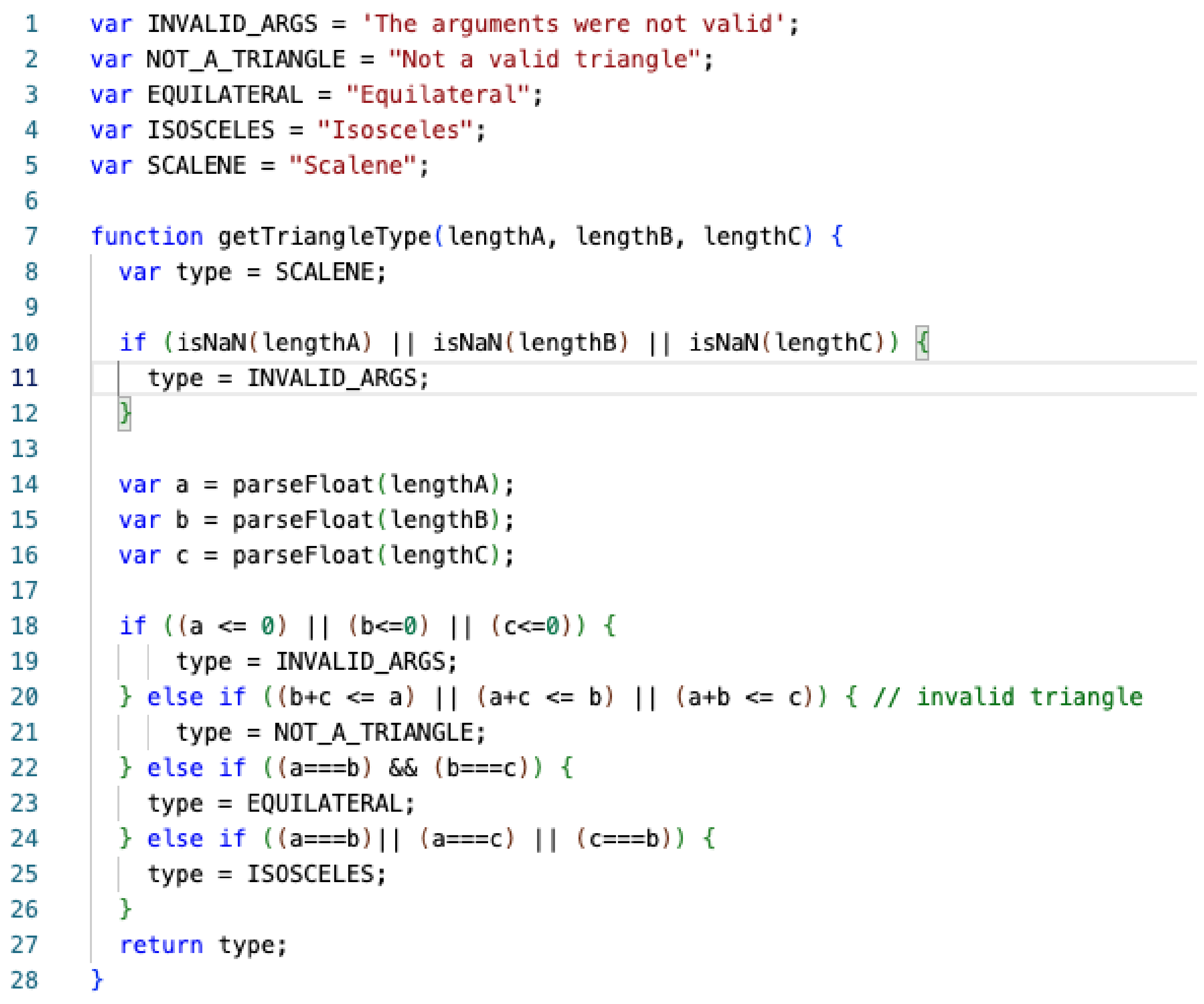


```
describe('valid arguments', function() {
  it('if sides are all equal it should be equilateral', function() {
    expect(getTriangleType('2','2','2')).to.equal('Equilateral');
  });

  it('if first and second sides are equal but different to third it should be isosceles', function() {
    expect(getTriangleType('2','2','3')).to.equal('Isosceles');
  });

  it('if first and third sides are equal but different to second it should be isosceles', function() {
    expect(getTriangleType('3','5','3')).to.equal('Isosceles');
  });

  it('if second and third sides are equal but different to first it should be isosceles', function() {
    expect(getTriangleType('4','5','5')).to.equal('Isosceles');
  });

  it('if all sides are different it should be scalene', function() {
    expect(getTriangleType('4','6','5')).to.equal('Scalene');
  });
});
```



```
invalid arguments
  ✓ if sides are not numbers it should be invalid arguments
  ✓ if one side is a negative number it should be invalid arguments
  ✓ if the first side is bigger than the addition of the other two, it should be not a valid triangle
  ✓ if the second side is bigger than the addition of the other two, it should be not a valid triangle
  ✓ if the third side is bigger than the addition of the other two, it should be not a valid triangle
  ✓ if the second side is a negative number it should be invalid arguments
  ✓ if the third side is a negative number it should be invalid arguments

valid arguments
  ✓ if sides are all equal it should be equilateral
  ✓ if first and second sides are equal but different to third it should be isosceles
  ✓ if first and third sides are equal but different to second it should be isosceles
  ✓ if second and third sides are equal but different to first it should be isosceles
  ✖ if all sides are different it should be scalene

    AssertionError: expected 'The arguments were not valid' to equal 'Scalene'
        at n.<anonymous> (tests.js:52:45)
```

Any attempt to make a very basic implementation of scalene fails because old tests start to fail. This is one of the benefits of test automation, we detect automatically that we've broken something that formerly was working on. It's time to do some refactor and change the implementation to make more efficient.



```javascript
var INVALID_ARGS = 'The arguments were not valid';
var NOT_A_TRIANGLE = "Not a valid triangle";
var EQUILATERAL = "Equilateral";
var ISOSCELES = "Isosceles";
var SCALENE = "Scalene";

function getTriangleType(lengthA, lengthB, lengthC) {
  var type = SCALENE;

  if (isNaN(lengthA) || isNaN(lengthB) || isNaN(lengthC)) {
    type = INVALID_ARGS;
  }

  var a = parseFloat(lengthA);
  var b = parseFloat(lengthB);
  var c = parseFloat(lengthC);

  if ((a <= 0) || (b<=0) || (c<=0)) {
      type = INVALID_ARGS;
  } else if ((b+c <= a) || (a+c <= b) || (a+b <= c)) { // invalid triangle
      type = NOT_A_TRIANGLE;
  } else if ((a===b) && (b===c)) {
    type = EQUILATERAL;
  } else if ((a===b)|| (a===c) || (c===b)) {
    type = ISOSCELES;
  }
  return type;
}
```