Gohel Akshay R.

Date:-05-07-2025

Testcase suite of Incubyte-TDD-Assessment

In this document,I have mentioned how I evolve my code.How i handle each test case and code also.

Table Of Content

Testcase 1:-Create a simple String calculator with a method signature like this: int add(string numbers)

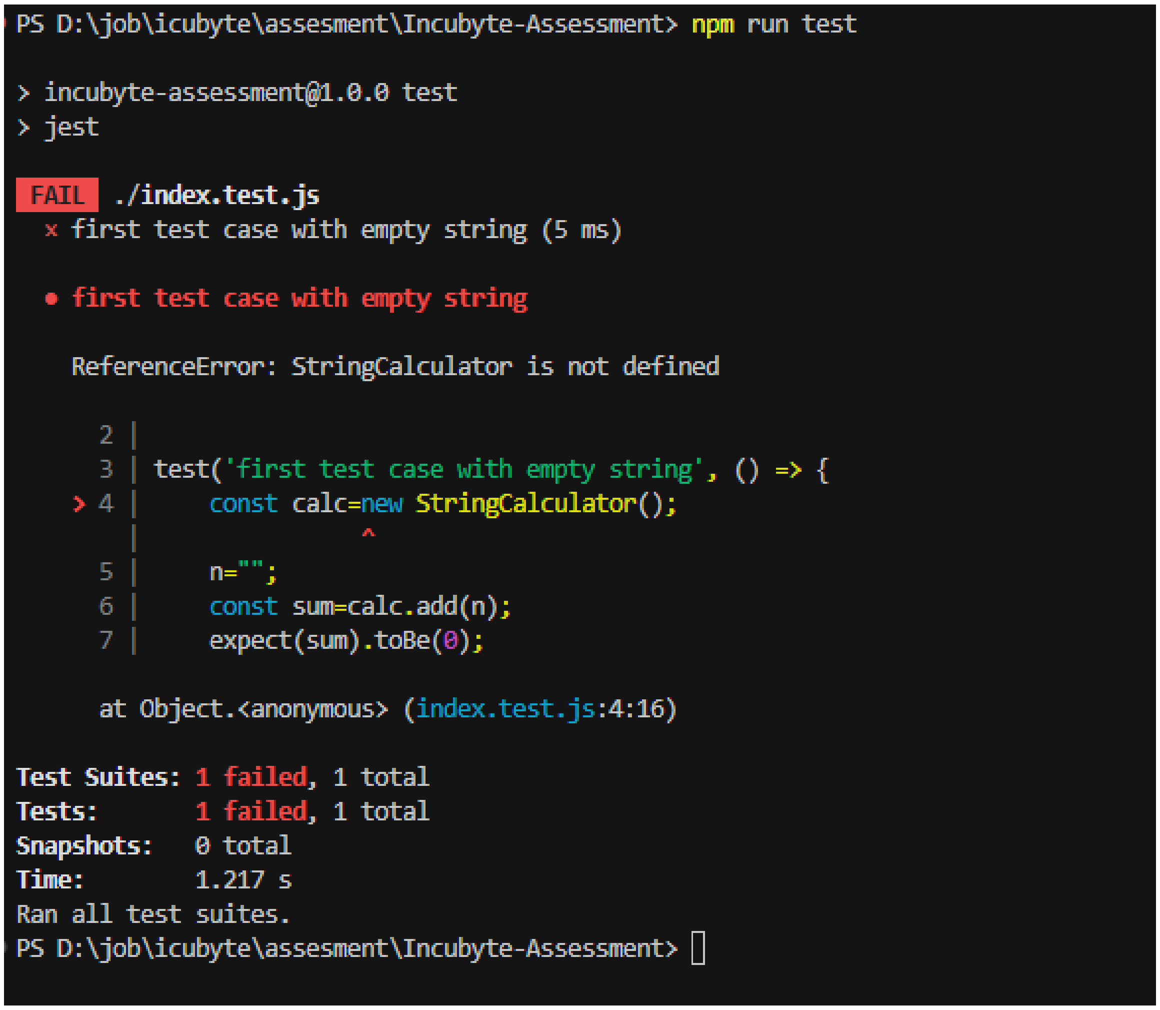
- Input: a string of comma-separated numbers
- Output: an integer, sum of the numbers
- Examples:
  - Input: “”, Output: 0
  - Input: “1”, Output: 1
  - Input: “1,5”, Output: 6

Code:- index.test.js

```
test('first test case with empty string', () => {
    const calc=new StringCalculator();
    n="";
    const sum=calc.add(n);
    expect(sum).toBe(0);
});
```

Explanation:-for beginning i take simple n as numbers for simplicity but i will update it soon. Here i have created one test case for checking empty string and have not written any code. based on rule of test driven development rules,i here failed test case and now i will refactor my code and then again repeats the cycle.here problem is that i have not declare any class named StringCalculator and i try to creating object of it thats raised error on my code.you can see output of that as Screenshot i have put it follow.

Output:-



```
PS D:\job\icubyte\assesment\Incubyte-Assessment> npm run test

> incubyte-assessment@1.0.0 test
> jest

 FAIL  ./index.test.js
  × first test case with empty string (5 ms)

  ● first test case with empty string

    ReferenceError: StringCalculator is not defined

      2 |
      3 | test('first test case with empty string', () => {
    > 4 |     const calc=new StringCalculator();
        |                  ^
      5 |     n="";
      6 |     const sum=calc.add(n);
      7 |     expect(sum).toBe(0);

      at Object.<anonymous> (index.test.js:4:16)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:   0 total
Time:        1.217 s
Ran all test suites.
PS D:\job\icubyte\assesment\Incubyte-Assessment>
```

Now refactoring i have completed.

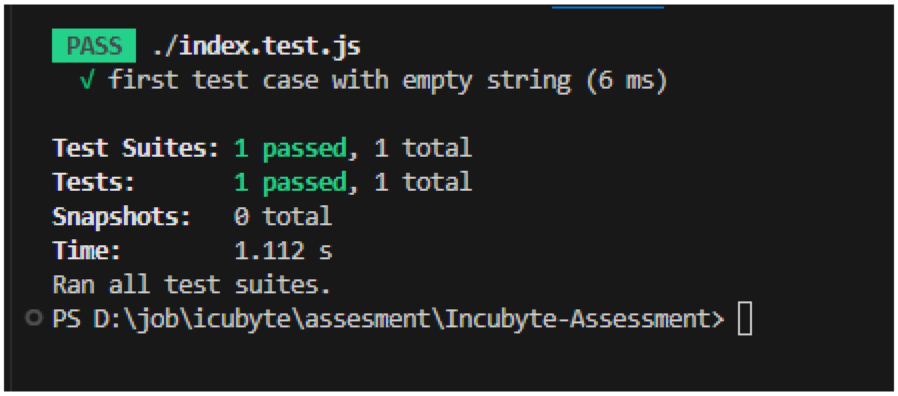
Code:-
index.test.js

```
const StringCalculator=require('./index')
test('first test case with empty string', () => {
    const calc=new StringCalculator();
    n="";
    const sum=calc.add(n);
    expect(sum).toBe(0);
});
```

index.js

```
class StringCalculator{
    add(n){
        if(n=="") return 0;
    }
}

module.exports=StringCalculator;
```

Explanation:- Here we can see that we did above code for pass the testcase mentioned in file.
Output:-



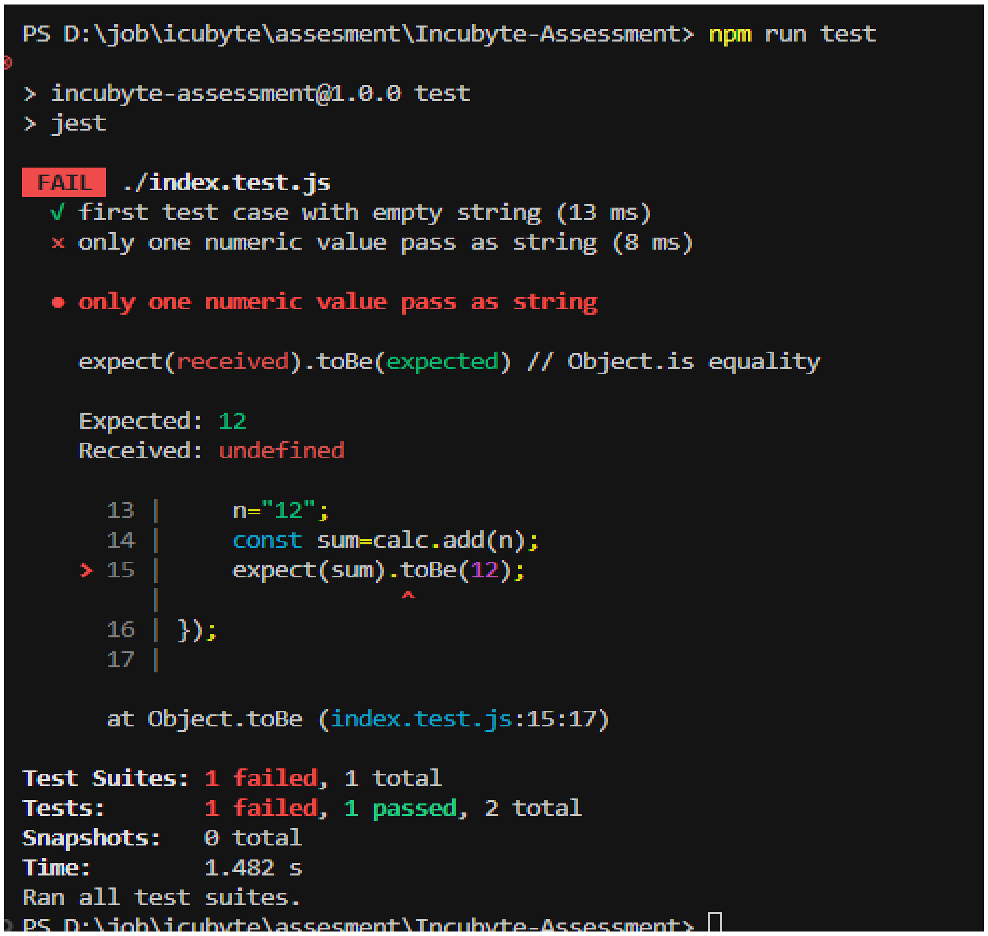
```
 PASS  ./index.test.js
  √ first test case with empty string (6 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.112 s
Ran all test suites.
PS D:\job\icubyte\assesment\Incubyte-Assessment>
```

Now i have write new test case in file index.test.js

```
const StringCalculator=require('./index')
test('only one numeric value pass as string', () => {
    const calc=new StringCalculator();
    n="12";
    const sum=calc.add(n);
    expect(sum).toBe(12);
});
```

Explanation:- Here in the add method i have not written any code related to handling input as numeric value as string passes.

Output:-

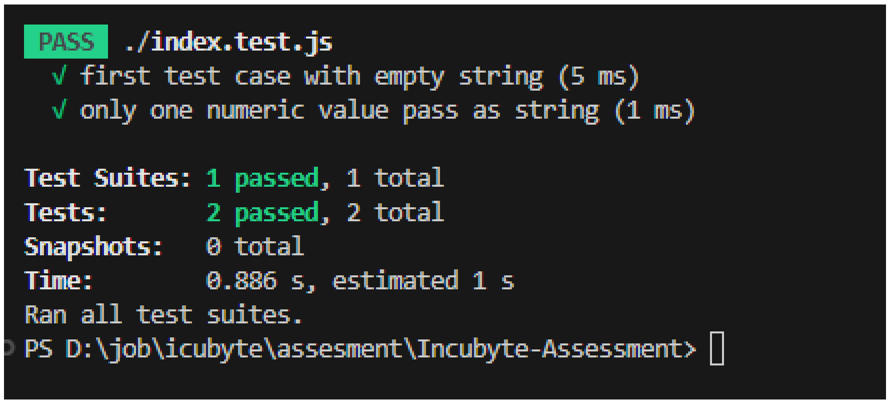


```
PS D:\job\icubyte\assesment\Incubyte-Assessment> npm run test

> incubyte-assessment@1.0.0 test
> jest

 FAIL  ./index.test.js
  √ first test case with empty string (13 ms)
  × only one numeric value pass as string (8 ms)

  ● only one numeric value pass as string

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

    Expected: 12
    Received: undefined

      13 |     n="12";
      14 |     const sum=calc.add(n);
    > 15 |     expect(sum).toBe(12);
         |                 ^
      16 | });
      17 |

      at Object.toBe (index.test.js:15:17)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 passed, 2 total
Snapshots:   0 total
Time:        1.482 s
Ran all test suites.
PS D:\job\icubyte\assesment\Incubyte-Assessment>
```

Now Again refactor the code and update it.
My updated version of index.js code as follow

```
class StringCalculator{
   add(n){
      if(n=="") return 0;
      return parseInt(n);
   }
}
module.exports=StringCalculator;
```

Explanation:- here i have use parseInt method for parse integer from string
Output:-



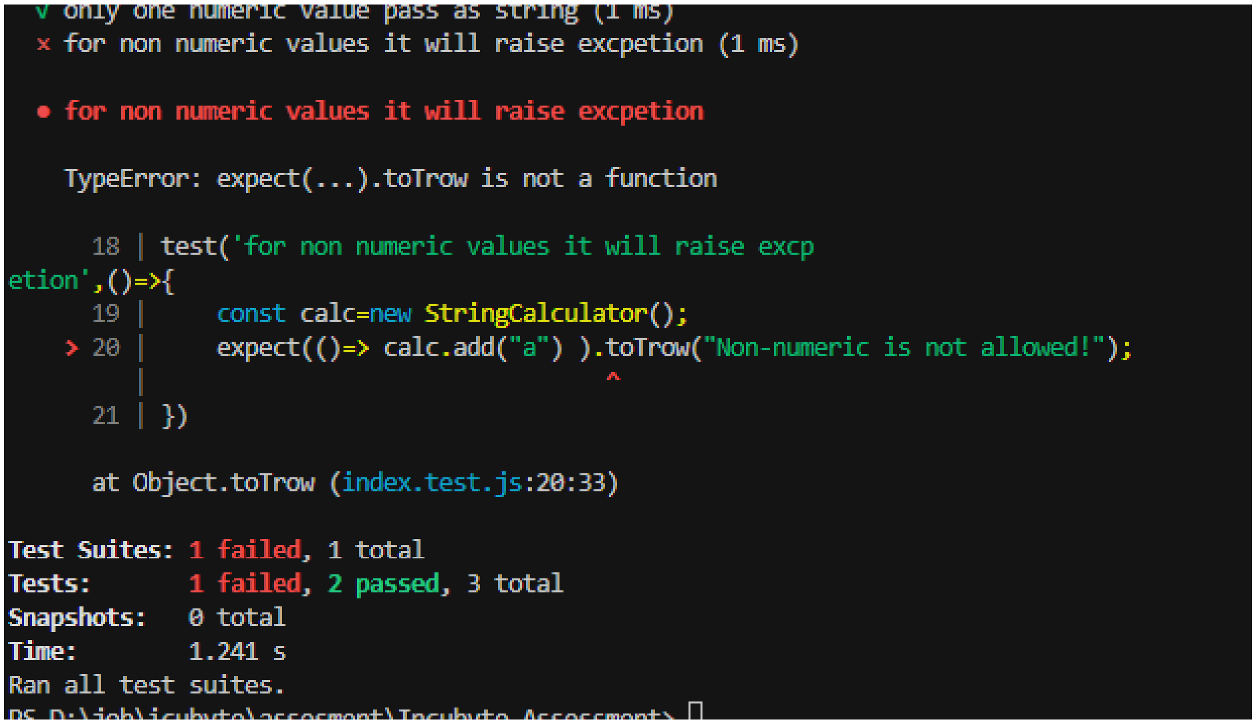
```
 PASS  ./index.test.js
  √ first test case with empty string (5 ms)
  √ only one numeric value pass as string (1 ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        0.886 s, estimated 1 s
Ran all test suites.
PS D:\job\icubyte\assesment\Incubyte-Assessment>
```

Now i add new test case as mentioned below

```
test('for non numeric values it will raise excpetion',()=>{
   const calc=new StringCalculator();
   expect(()=> calc.add("a") ).toTrow("Non-numeric is not allowed!");
})
```

Explanation:- this test case is used to show if non-numberic value pass as input it will raise an exception.

Output:-



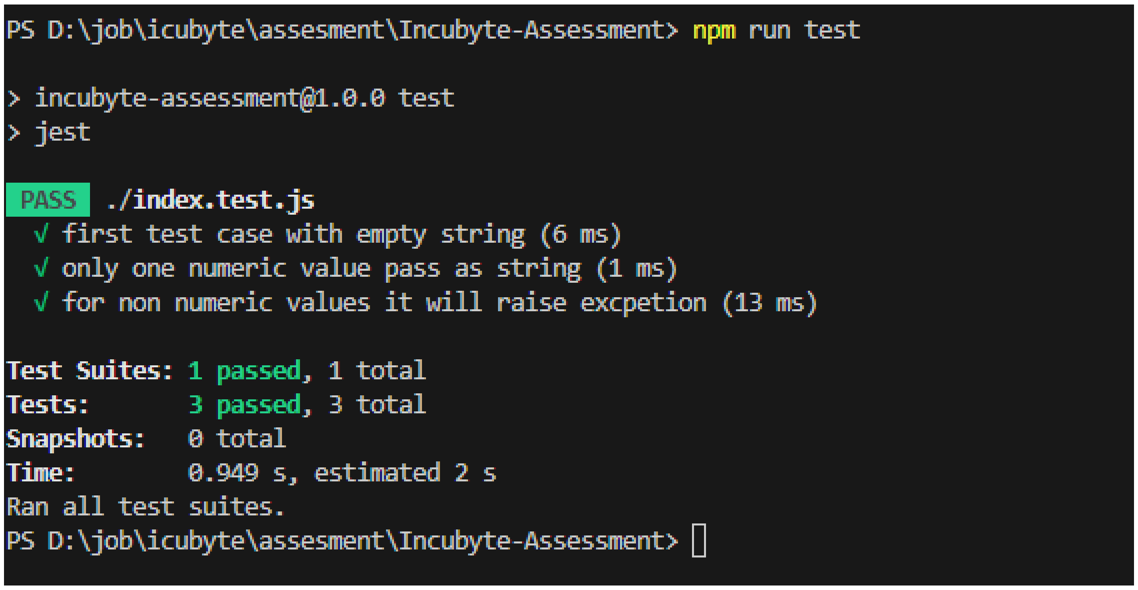
```
  √ only one numeric value pass as string (1 ms)
  × for non numeric values it will raise excpetion (1 ms)

  ● for non numeric values it will raise excpetion

    TypeError: expect(...).toTrow is not a function

      18 | test('for non numeric values it will raise excp
etion',()=>{
      19 |     const calc=new StringCalculator();
    > 20 |     expect(()=> calc.add("a") ).toTrow("Non-numeric is not allowed!");
         |                                 ^
      21 | })

      at Object.toTrow (index.test.js:20:33)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 2 passed, 3 total
Snapshots:   0 total
Time:        1.241 s
Ran all test suites.
PS D:\job\icubyte\assessment\Incubyte Assessment>
```

Now try to reflect this code.

```
class StringCalculator{
    add(n){
        if(n=="") return 0;
        if(!isNaN(parseInt(n))){
            return parseInt(n);
        }
        else{
            throw new Error("Non-numeric is not allowed!");
        }
    }
}

module.exports=StringCalculator;
```

Explanation:- here i check if number cant parse into int means it not a numeric value so i raise Exception .

Output:-



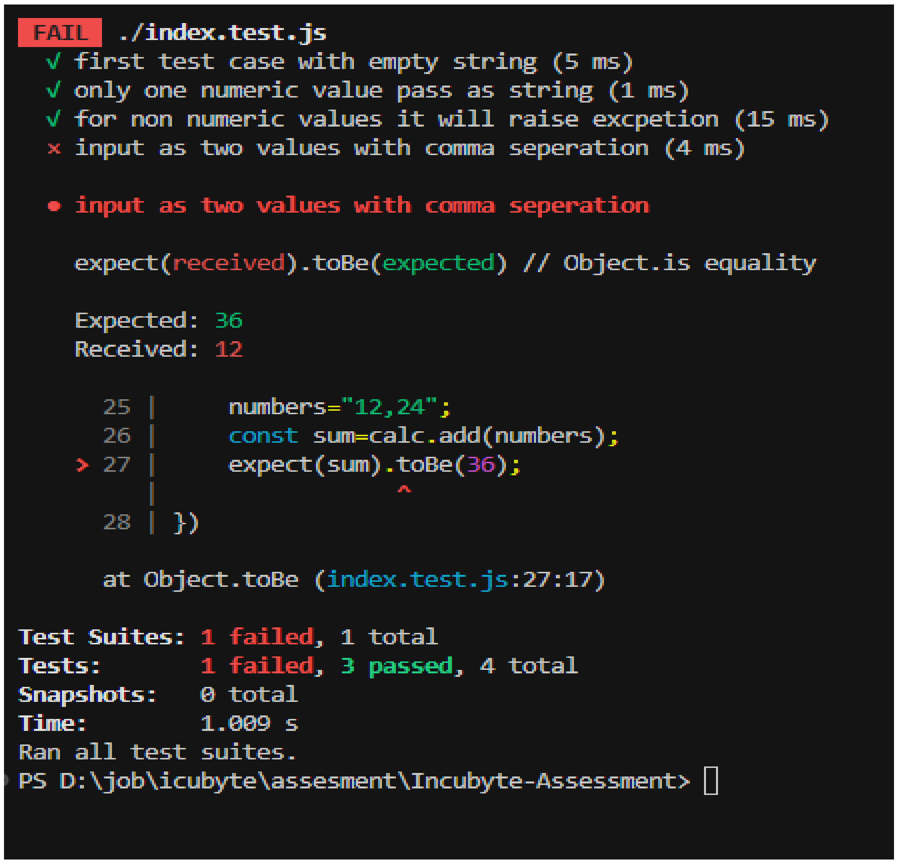
```
PS D:\job\icubyte\assesment\Incubyte-Assessment> npm run test

> incubyte-assessment@1.0.0 test
> jest

 PASS  ./index.test.js
  √ first test case with empty string (6 ms)
  √ only one numeric value pass as string (1 ms)
  √ for non numeric values it will raise excpetion (13 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.949 s, estimated 2 s
Ran all test suites.
PS D:\job\icubyte\assesment\Incubyte-Assessment> 
```

Now again need to add one more test case as mentioned below

```
test("input as two values with comma separation",()=>{
   const calc=new StringCalculator();
   numbers="12,24";
   const sum=calc.add(numbers);
   expect(sum).toBe(36);
})
```

Explanation:- this test case is for input as two values with comma separation.

Output:-



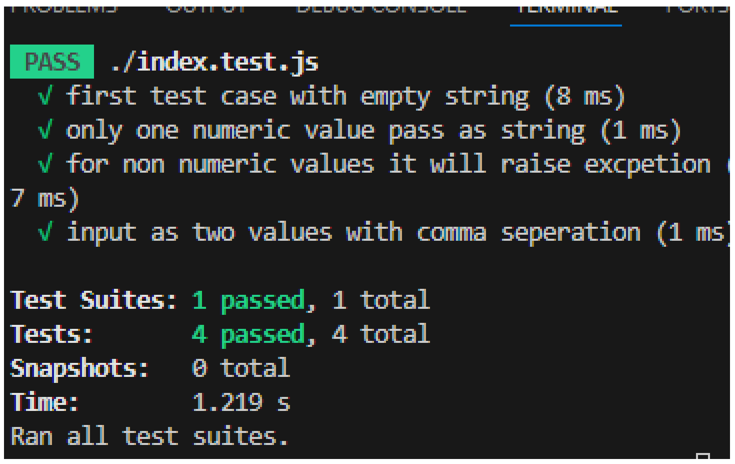
```
 FAIL  ./index.test.js
  √ first test case with empty string (5 ms)
  √ only one numeric value pass as string (1 ms)
  √ for non numeric values it will raise excpetion (15 ms)
  × input as two values with comma seperation (4 ms)

  ● input as two values with comma seperation

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

    Expected: 36
    Received: 12

      25 |      numbers="12,24";
      26 |      const sum=calc.add(numbers);
    > 27 |      expect(sum).toBe(36);
         |                  ^
      28 | })

      at Object.toBe (index.test.js:27:17)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 3 passed, 4 total
Snapshots:   0 total
Time:        1.009 s
Ran all test suites.
PS D:\job\icubyte\assesment\Incubyte-Assessment> 
```

I have again refactor my code for pass the test case .

index.js

```
class StringCalculator{
    add(numbers){
        if(numbers=="") return 0;
        numbers=numbers.split(",");
        if(numbers.length==1){
            if(!isNaN(parseInt(numbers))){
                return parseInt(numbers);
            }
            else{
                throw new Error("Non-numeric is not allowed!");
            }
        }
        else if(numbers.length==2){
            let number1=numbers[0];
            let number2=numbers[1];
            let sum=0;
            if(!isNaN(parseInt(number1))){
                sum+= parseInt(number1);
            }
            else{
                throw new Error("Non-numeric is not allowed!");
            }
            if(!isNaN(parseInt(number2))){
                sum+= parseInt(number2);
            }
            else{
                throw new Error("Non-numeric is not allowed!");
            }

            return sum;
        }

    }
}

module.exports=StringCalculator;
```

Explanation:- i have use split function for splitting numbers
Output:-



```
 PASS  ./index.test.js
  √ first test case with empty string (8 ms)
  √ only one numeric value pass as string (1 ms)
  √ for non numeric values it will raise excpetion (
7 ms)
  √ input as two values with comma seperation (1 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        1.219 s
Ran all test suites.
```

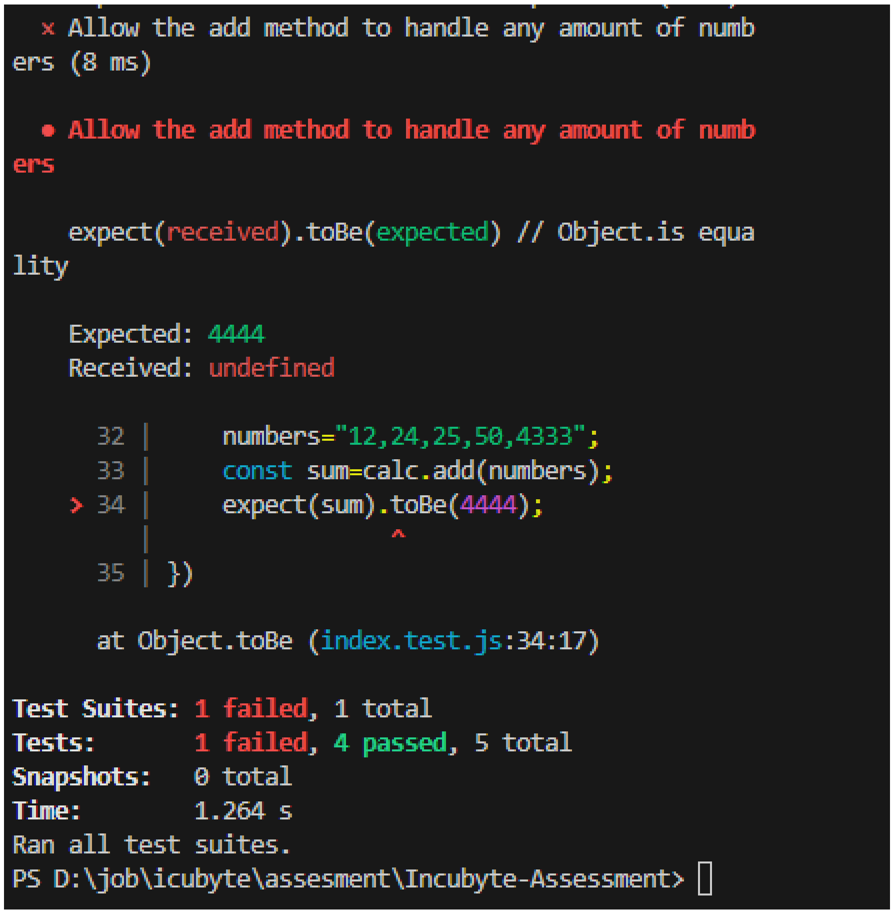
Testcase 2:-Allow the add method to handle any amount of numbers.

I wrote one test case for multiple numbers as mentioned below

```
test("Allow the add method to handle any amount of numbers" ,()=>{
    const calc=new StringCalculator();
   numbers="12,24,25,50,4333";
   const sum=calc.add(numbers);
   expect(sum).toBe(4444);
})
```

Explanation:- Here you can see, multiple numbers are passed by separating through commas.

Output:-



```
  × Allow the add method to handle any amount of numb
ers (8 ms)

  ● Allow the add method to handle any amount of numb
ers

    expect(received).toBe(expected) // Object.is equa
lity

    Expected: 4444
    Received: undefined

      32 |       numbers="12,24,25,50,4333";
      33 |       const sum=calc.add(numbers);
    > 34 |       expect(sum).toBe(4444);
         |                   ^
      35 | })

      at Object.toBe (index.test.js:34:17)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 4 passed, 5 total
Snapshots:   0 total
Time:        1.264 s
Ran all test suites.
PS D:\job\icubyte\assesment\Incubyte-Assessment>
```

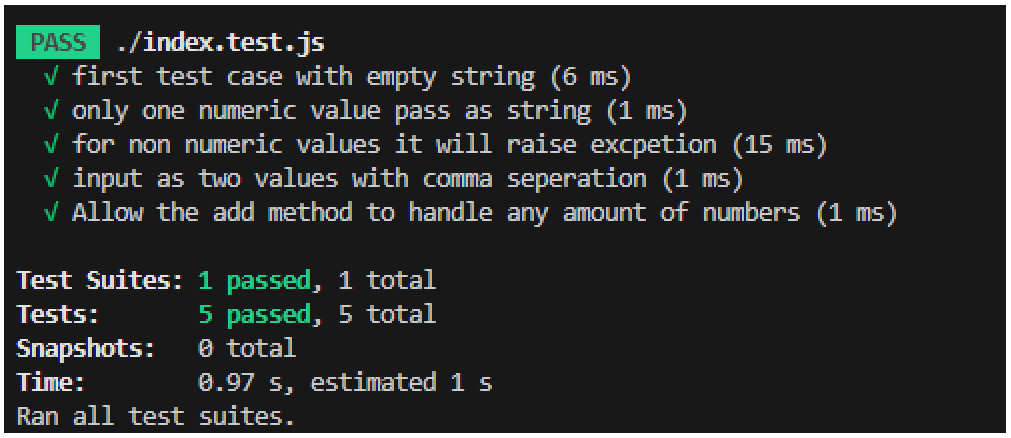
Now , I am referencing the code for passing the testcase.
My code looks more clean.
index.js

```
class StringCalculator{
   add(numbers){
      if(numbers=="") return 0;
      numbers=numbers.split(",");

      let sum=0;
      for(let i=0;i<numbers.length;i++){
         if(!isNaN(parseInt(numbers[i]))){
            sum+= parseInt(numbers[i]);
         }
```

```
            else{
                throw new Error("Non-numeric is not allowed!");
            }
        }
            return sum;
    }
}

module.exports=StringCalculator;
```

Explanation:- Here i use for loop for adding all numeric values.

Output:-



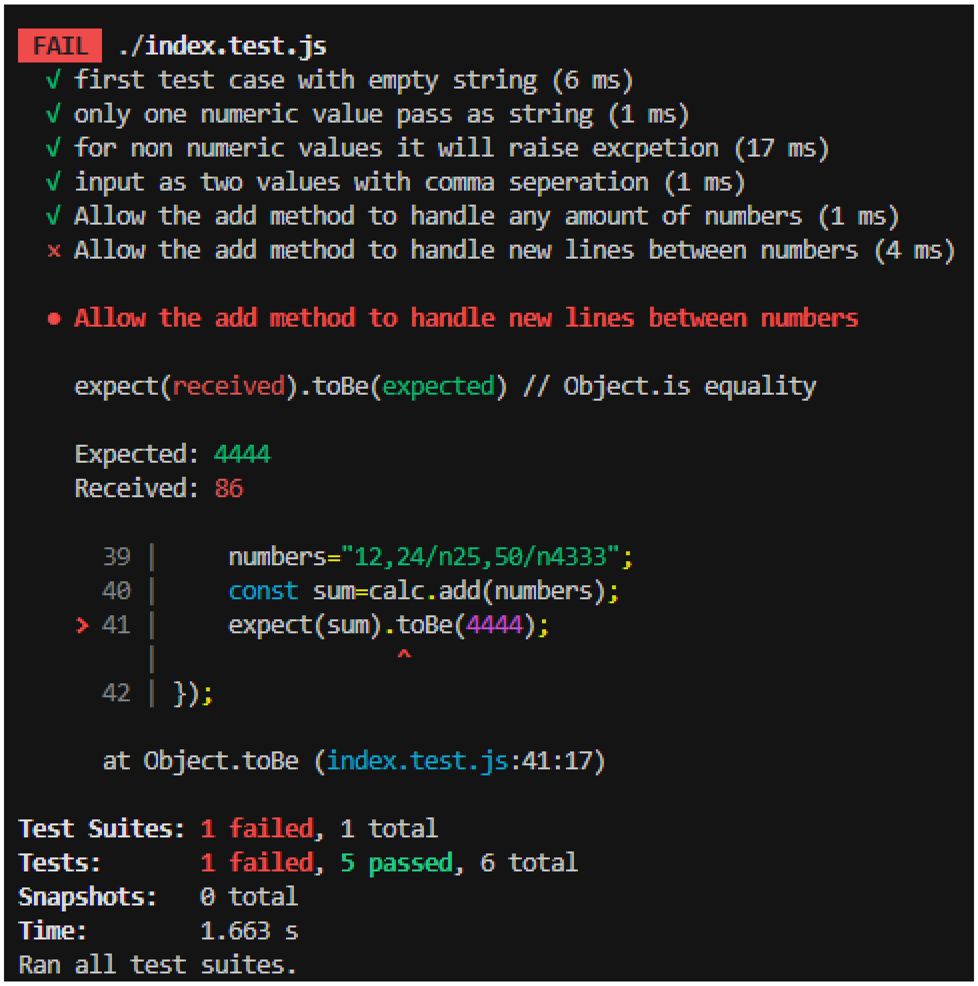
```
PASS  ./index.test.js
  √ first test case with empty string (6 ms)
  √ only one numeric value pass as string (1 ms)
  √ for non numeric values it will raise excpetion (15 ms)
  √ input as two values with comma seperation (1 ms)
  √ Allow the add method to handle any amount of numbers (1 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        0.97 s, estimated 1 s
Ran all test suites.
```

Testcase 3:Allow the add method to handle new lines between numbers

I write test case as mentioned below

```
test("Allow the add method to handle new lines between numbers",()=>{
    calc=new StringCalculator();
    numbers="12,24\n25,50\n4333";
    const sum=calc.add(numbers);
    expect(sum).toBe(4444);
});
```

Explanation:- here i use \n (new line) as delimiter.
Output:-



```
FAIL  ./index.test.js
  √ first test case with empty string (6 ms)
  √ only one numeric value pass as string (1 ms)
  √ for non numeric values it will raise excpetion (17 ms)
  √ input as two values with comma seperation (1 ms)
  √ Allow the add method to handle any amount of numbers (1 ms)
  × Allow the add method to handle new lines between numbers (4 ms)

  ● Allow the add method to handle new lines between numbers

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

    Expected: 4444
    Received: 86

          numbers="12,24/n25,50/n4333";
          const sum=calc.add(numbers);
    >     expect(sum).toBe(4444);
                      ^
      });

      at Object.toBe (index.test.js:41:17)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 5 passed, 6 total
Snapshots:   0 total
Time:        1.663 s
Ran all test suites.
```

For pass above test case, i refactored the code as mentioned below
index.js

```
class StringCalculator{
   add(numbers){
      if(numbers=="") return 0;
      numbers=numbers.split(/[\n,]/);

      let sum=0;
      for(let i=0;i<numbers.length;i++){
         if(!isNaN(parseInt(numbers[i]))){
            sum+= parseInt(numbers[i]);
         }
         else{
            throw new Error("Non-numeric is not allowed!");
         }
      }

         return sum;
   }
}

module.exports=StringCalculator;
```

Explanation:- i just use regular expression on split function for choose any of them for splitting from , or \n(new line) character.
Output:-

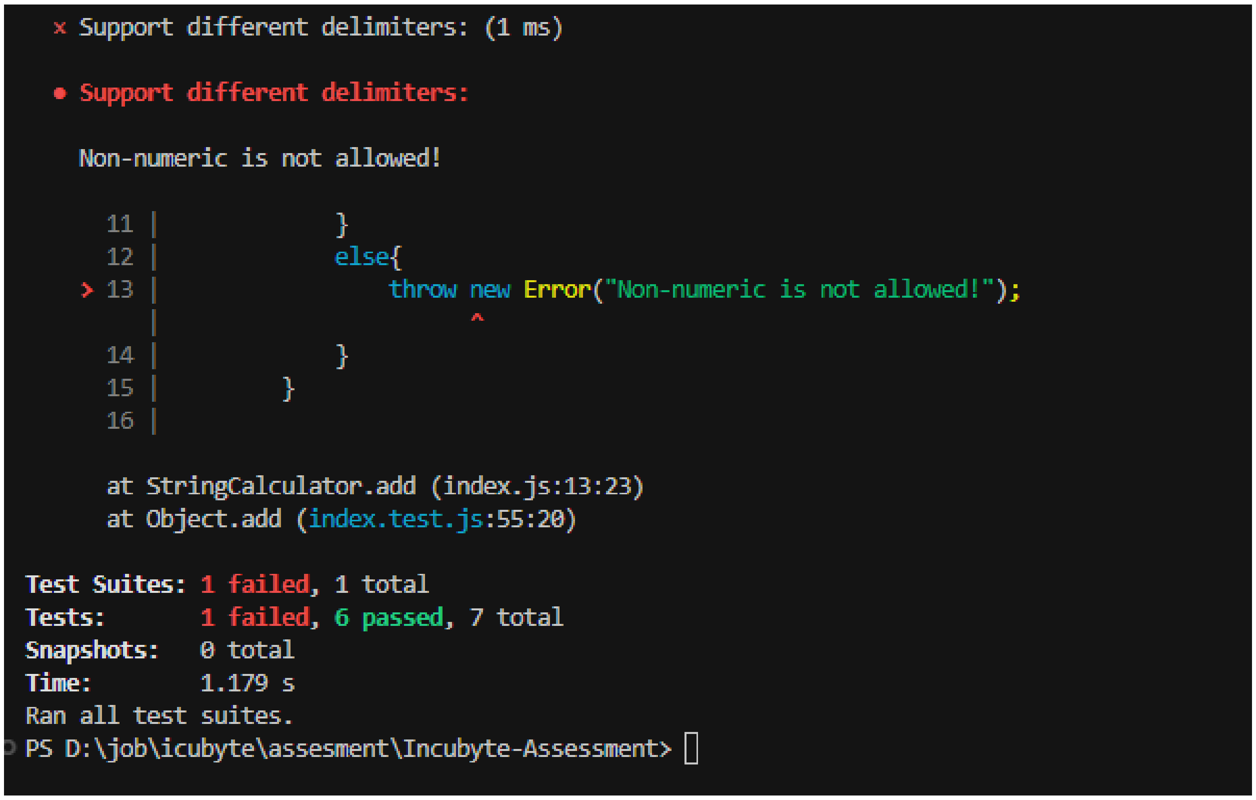
Testcase 4:-Support different delimiters:

Here below is mention code you can refer it.

```
/*
To change the delimiter, the beginning of the string will contain
a separate line that looks like this: "//[delimiter]\n[numbers…]".
//  For example, "//;\n1;2" where the delimiter is ";"
//  should return 3.
*/

test("Support different delimiters:",()=>{
   calc=new StringCalculator();
   numbers="//;\n12,24;25,50\n4333";
   const sum=calc.add(numbers);
   expect(sum).toBe(4444);
})
```

Explanation:- i have write one test case for testing input string with changing delimiter to ;, and it fails the test case as you can see in output section.

Output:-



```
  × Support different delimiters: (1 ms)

  ● Support different delimiters:

    Non-numeric is not allowed!

      11 |             }
      12 |             else{
    > 13 |                 throw new Error("Non-numeric is not allowed!");
         |                       ^
      14 |             }
      15 |         }
      16 |

      at StringCalculator.add (index.js:13:23)
      at Object.add (index.test.js:55:20)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 6 passed, 7 total
Snapshots:   0 total
Time:        1.179 s
Ran all test suites.
PS D:\job\icubyte\assesment\Incubyte-Assessment> 
```

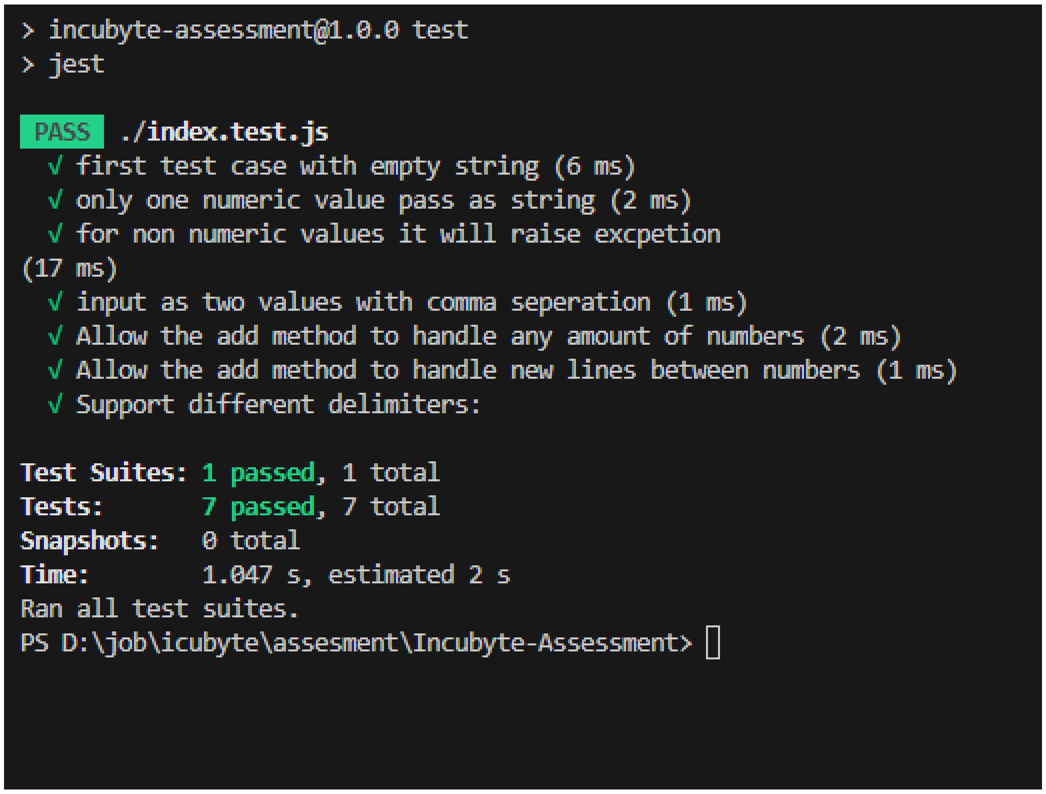
I had done refectoring of the code, you can see mentioned as follow

```
class StringCalculator{
   add(numbers){
      if(numbers=="") return 0;
      let delimiters = "\n,"; //default delimiters

      if (numbers.startsWith("//") && numbers[3] === '\n') {
         delimiters += numbers[2]; //find delimiter here
         numbers = numbers.substring(4); //changing position of numbers to avoid
first 3 characters
      }

     //adding escap characters to each
      const escapedDelimiters = delimiters
      .split('')
      .map(c => '\\' + c)
```

```
        .join('');

        // create RegExp from the dynamic string because we can not directly use
variable
        const delimiterRegex = new RegExp(`[${escapedDelimiters}]`);

        // now split using regex
        numbers = numbers.split(delimiterRegex); //here we can't split directly by
taking all delimiter inside a string we need object of RegExp
        let sum=0; //for initial value of sum as zero
        for(let i=0;i<numbers.length;i++){
            if(!isNaN(parseInt(numbers[i]))){ //check if non numeric values
                sum+= parseInt(numbers[i]); //do addition of each values
            }
            else{
                throw new Error("Non-numeric is not allowed!"); //if non numeric values
then raise Exception
            }
        }

            return sum; //return final sum
    }
}

module.exports=StringCalculator;
```

Explanation:- here i first check if in starting, there is // or not and if yes then check what is new delimiter and add it to our regex Expression bu RegExp object and pass it to split method.

Output:-



```
> incubyte-assessment@1.0.0 test
> jest

 PASS  ./index.test.js
  √ first test case with empty string (6 ms)
  √ only one numeric value pass as string (2 ms)
  √ for non numeric values it will raise excpetion
(17 ms)
  √ input as two values with comma seperation (1 ms)
  √ Allow the add method to handle any amount of numbers (2 ms)
  √ Allow the add method to handle new lines between numbers (1 ms)
  √ Support different delimiters:

Test Suites: 1 passed, 1 total
Tests:       7 passed, 7 total
Snapshots:   0 total
Time:        1.047 s, estimated 2 s
Ran all test suites.
PS D:\job\icubyte\assesment\Incubyte-Assessment>
```

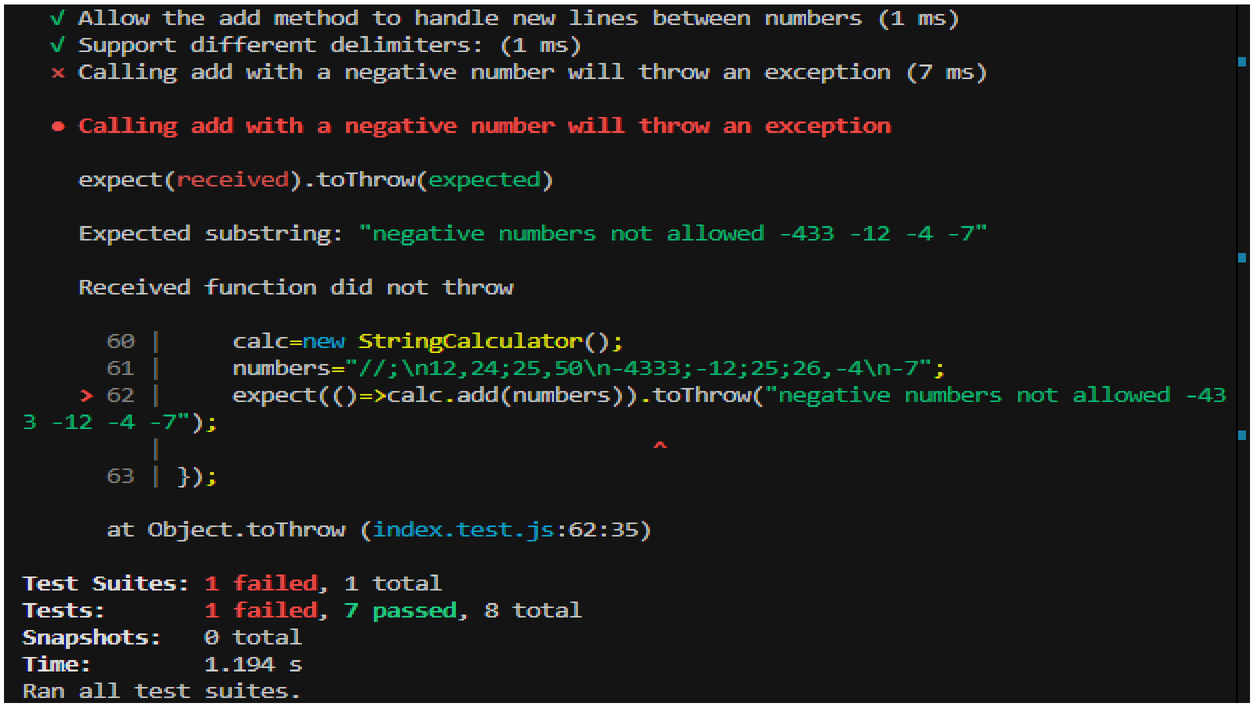
Testcase 5:- Calling add with a negative number will throw an exception

-"negative numbers not allowed <negative_number>
-If there are multiple negative numbers, show all of them in the exception message, separated by commas.

Here I have written one test code which fails to throw an Exception when negative numbers are present in the input string.

```
test("Calling add with a negative number will throw an exception",()=>{
   calc=new StringCalculator();
   numbers="//;\n12,24;25,50\n-4333;-12;25;26,-4\n-7";
   expect(()=>calc.add(numbers)).toThrow("negative numbers not allowed
-433,-12,-4,-7");
});
```

Explanation:- Here i have write one test case having -4333 -12 -4 -7 as negative number and i want to throw exception by my code.
Output:-



```
  √ Allow the add method to handle new lines between numbers (1 ms)
  √ Support different delimiters: (1 ms)
  × Calling add with a negative number will throw an exception (7 ms)

  ● Calling add with a negative number will throw an exception

    expect(received).toThrow(expected)

    Expected substring: "negative numbers not allowed -433 -12 -4 -7"

    Received function did not throw

      60 |     calc=new StringCalculator();
      61 |     numbers="//;\n12,24;25,50\n-4333;-12;25;26,-4\n-7";
    > 62 |     expect(()=>calc.add(numbers)).toThrow("negative numbers not allowed -43
3 -12 -4 -7");
         |                                   ^
      63 | });

      at Object.toThrow (index.test.js:62:35)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 7 passed, 8 total
Snapshots:   0 total
Time:        1.194 s
Ran all test suites.
```

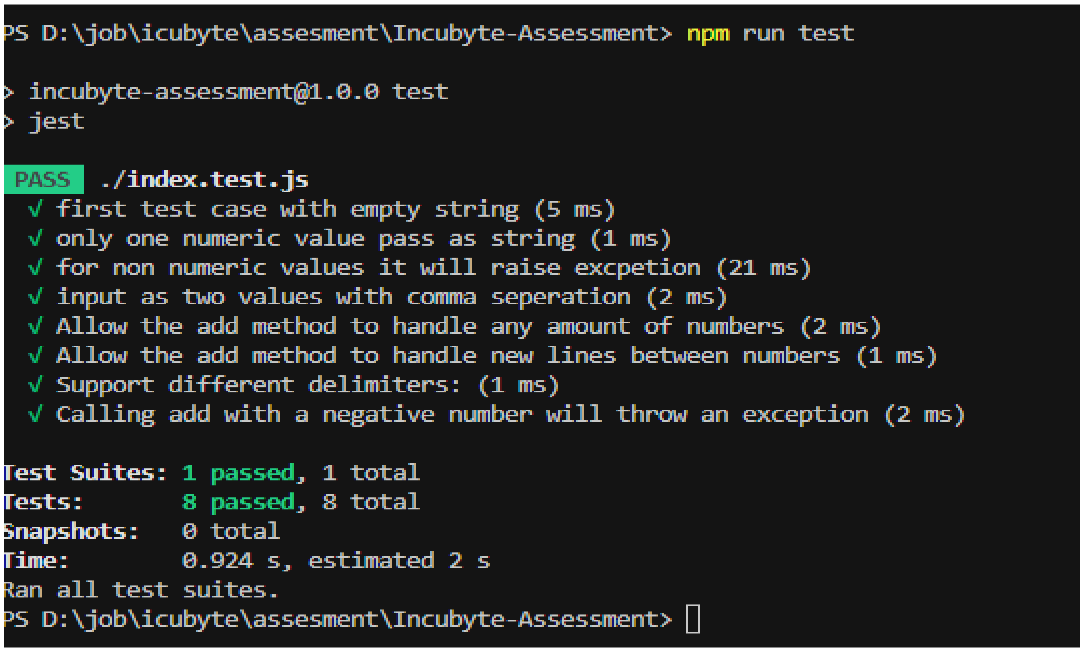
Now, I am again refectoring my code to maintain Kata.its love to do in TDD because it will work very easily and i have to just solve very small problems and achieve solutions for big problems.

index.js

```
class StringCalculator{
    add(numbers){
        if(numbers=="") return 0;
        let delimiters = "\n,"; //default delimiters
        let negative_numbers=[]; //contains negative numbers
        if (numbers.startsWith("//") && numbers[3] === '\n') {
            delimiters += numbers[2]; //find delimiter here
            numbers = numbers.substring(4); //changing position of numbers to avoid
first 3 characters
        }
       //adding escap characters to each
        const escapedDelimiters = delimiters
        .split('')
        .map(c => '\\' + c)
        .join('');
        // create RegExp from the dynamic string because we can not directly use
variable
        const delimiterRegex = new RegExp(`[${escapedDelimiters}]`);
        // now split using regex
        numbers = numbers.split(delimiterRegex); //here we can't split directly by
taking all delimiter inside a string we need object of RegExp
        let sum=0; //for initial value of sum as zero
        for(let i=0;i<numbers.length;i++){
            if(!isNaN(parseInt(numbers[i]))){ //check if non numeric values
                if(numbers[i]<0) negative_numbers.push(numbers[i]); //add negative
numbers here in array
                sum+= parseInt(numbers[i]); //do addition of each values
            }
            else{
                throw new Error("Non-numeric is not allowed!"); //if non numeric values
then raise Exception
            }
        }
        if(negative_numbers.length>0){ //check if there is negative number then raise
error
            let CommonMessage="negative numbers not allowed"; //numbers are
dynamic but base message here is common
            let ErrorMessage=CommonMessage+" "+negative_numbers.join(',');
            throw new Error(ErrorMessage);
        }
        return sum; //return final sum
    }
}
module.exports=StringCalculator;
```

Explanation:- I have created one array which contains a negative element and at the end i check if there is any element of the negative_numbers array so raise an Exception.

Output:-

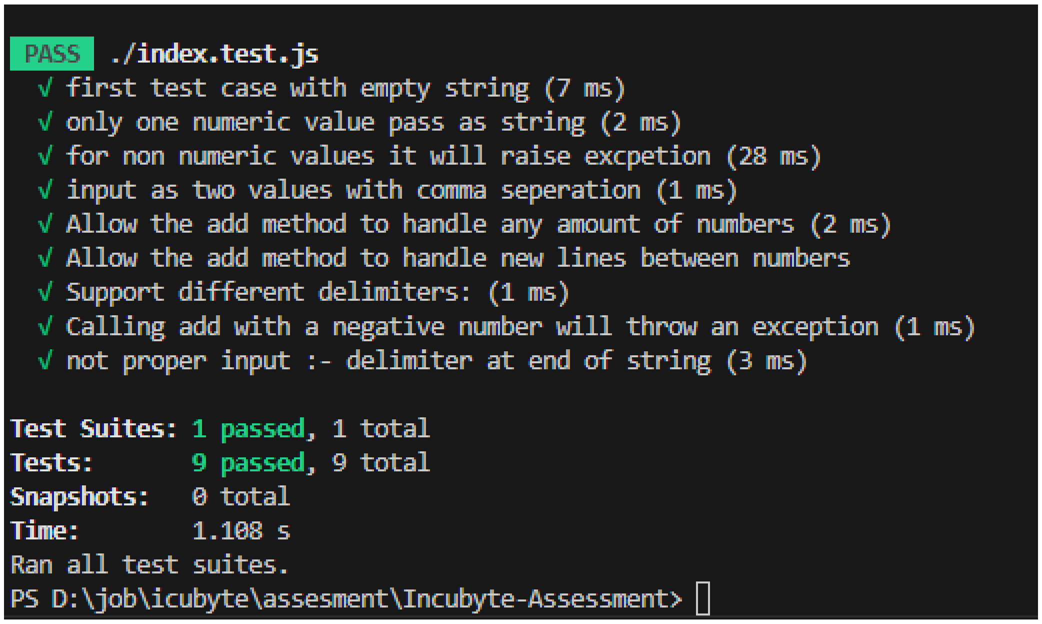


```
PS D:\job\icubyte\assesment\Incubyte-Assessment> npm run test

> incubyte-assessment@1.0.0 test
> jest

 PASS  ./index.test.js
  √ first test case with empty string (5 ms)
  √ only one numeric value pass as string (1 ms)
  √ for non numeric values it will raise excpetion (21 ms)
  √ input as two values with comma seperation (2 ms)
  √ Allow the add method to handle any amount of numbers (2 ms)
  √ Allow the add method to handle new lines between numbers (1 ms)
  √ Support different delimiters: (1 ms)
  √ Calling add with a negative number will throw an exception (2 ms)

Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        0.924 s, estimated 2 s
Ran all test suites.
PS D:\job\icubyte\assesment\Incubyte-Assessment> 
```

I try one more test case here,for not proper input :- delimiter at end of string
You can see test case as here:-

```
test('not proper input :- delimiter at end of string',()=>{
   const calc = new StringCalculator();
   numbers="2,-4,3,-1,,";
  expect(() => calc.add(numbers)).toThrow("Non-numeric is not allowed!");
})
```

Explanation:- Here ,i tried the above test case so what i did i see, it passed without falling because it automatically handles in the first testcase nonNumberc characters are not allowed.
Output:-



```
 PASS  ./index.test.js
  √ first test case with empty string (7 ms)
  √ only one numeric value pass as string (2 ms)
  √ for non numeric values it will raise excpetion (28 ms)
  √ input as two values with comma seperation (1 ms)
  √ Allow the add method to handle any amount of numbers (2 ms)
  √ Allow the add method to handle new lines between numbers
  √ Support different delimiters: (1 ms)
  √ Calling add with a negative number will throw an exception (1 ms)
  √ not proper input :- delimiter at end of string (3 ms)

Test Suites: 1 passed, 1 total
Tests:       9 passed, 9 total
Snapshots:   0 total
Time:        1.108 s
Ran all test suites.
PS D:\job\icubyte\assesment\Incubyte-Assessment>
```

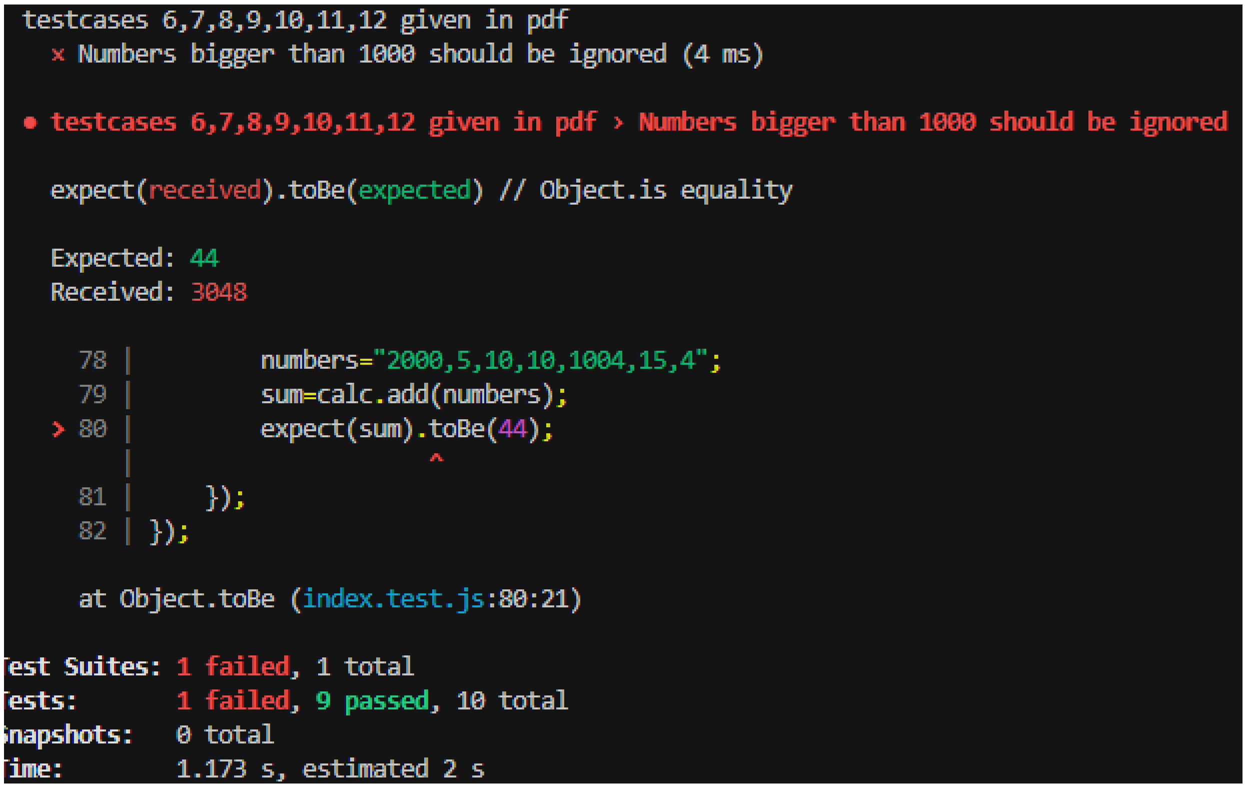
Testcase 6:-Numbers bigger than 1000 should be ignored.

Here you can see code for a test case which checks whether numbers bigger than 1000 are ignored or not.

```
describe("testcases 6,7,8,9,10,11,12 given in pdf",()=>{
    test("Numbers bigger than 1000 should be ignored",()=>{
        const calc=new StringCalculator();
        numbers="2000,5,10,10,1004,15,4";
        sum=calc.add(numbers);
        expect(sum).toBe(44);
    });
});
```

Explanation:- here i use 2000,1004 are more than 1000 so it will be ignored and sum will be 44 only.and here it fails .

Output:-



```
 testcases 6,7,8,9,10,11,12 given in pdf
   × Numbers bigger than 1000 should be ignored (4 ms)

 ● testcases 6,7,8,9,10,11,12 given in pdf › Numbers bigger than 1000 should be ignored

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

   Expected: 44
   Received: 3048

     78 |         numbers="2000,5,10,10,1004,15,4";
     79 |         sum=calc.add(numbers);
   > 80 |         expect(sum).toBe(44);
        |                     ^
     81 |     });
     82 | });

     at Object.toBe (index.test.js:80:21)

est Suites: 1 failed, 1 total
ests:       1 failed, 9 passed, 10 total
napshots:   0 total
ime:        1.173 s, estimated 2 s
```

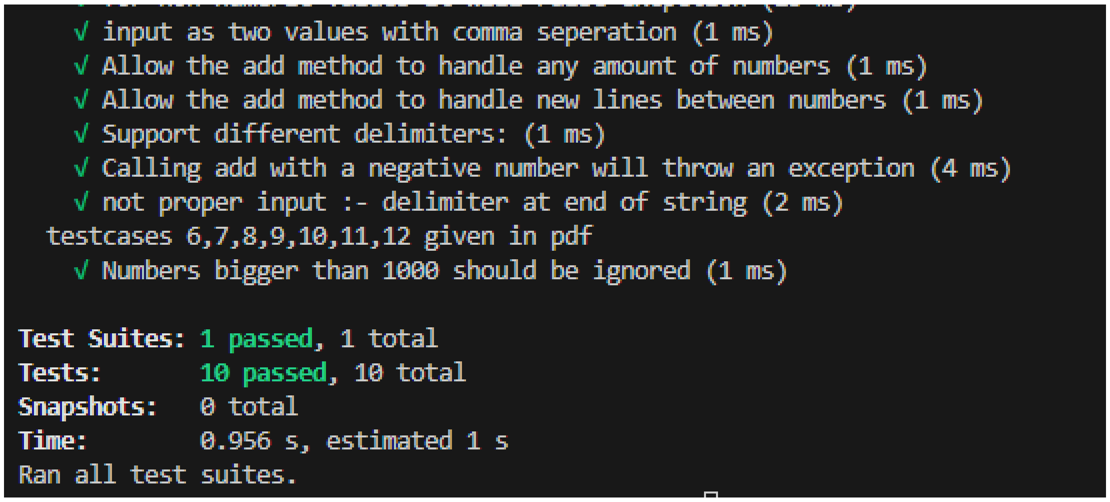
Now, it is to reflect the code.i also used many testcases more than 1000 values,so there i have to changes testcase for according sum matches and i update my code very little bit for pass this test case.

Code:-
index.js

```
class StringCalculator {
    add(numbers) {
        if (numbers == "") return 0;
        let delimiters = "\n,"; //default delimiters
        let negative_numbers = []; //contains negative numbers
        if (numbers.startsWith("//") && numbers[3] === '\n') {
            delimiters += numbers[2]; //find delimiter here
            numbers = numbers.substring(4); //changing position of numbers to avoid
first 3 characters
        }

        //adding escap characters to each
        const escapedDelimiters = delimiters
            .split('')
            .map(c => '\\' + c)
            .join('');

        // create RegExp from the dynamic string because we can not directly use
variable
        const delimiterRegex = new RegExp(`[${escapedDelimiters}]`);

        // now split using regex
        numbers = numbers.split(delimiterRegex); //here we can't split directly by
taking all delimiter inside a string we need object of RegExp
        let sum = 0; //for initial value of sum as zero
        for (let i = 0; i < numbers.length; i++) {
            if (!isNaN(parseInt(numbers[i]))) { //check if non numeric values
            if (numbers[i] < 0) negative_numbers.push(numbers[i]); //if number is
negative then add it here
            if (numbers[i] <= 1000) {
              sum += parseInt(numbers[i]); //do addition of each values if it is less
than or equal to 1000
            }
          }
          else {
            throw new Error("Non-numeric is not allowed!"); //if non numeric values
then raise Exception
          }
        }

        if (negative_numbers.length > 0) {
          let CommonMessage = "negative numbers not allowed"; //numbers are
dynamic but base message here is common

          let ErrorMessage = CommonMessage + " " + negative_numbers.join(',');
          throw new Error(ErrorMessage);
        }

        return sum; //return final sum
      }
}

module.exports = StringCalculator;
```

Explanation:- Here you can see that I have added just on condition for consider a value having only less than or equal to 1000 so it automatically ignores values which are more than 1000.

Output:-



```
    √ input as two values with comma seperation (1 ms)
    √ Allow the add method to handle any amount of numbers (1 ms)
    √ Allow the add method to handle new lines between numbers (1 ms)
    √ Support different delimiters: (1 ms)
    √ Calling add with a negative number will throw an exception (4 ms)
    √ not proper input :- delimiter at end of string (2 ms)
  testcases 6,7,8,9,10,11,12 given in pdf
    √ Numbers bigger than 1000 should be ignored (1 ms)

Test Suites: 1 passed, 1 total
Tests:       10 passed, 10 total
Snapshots:   0 total
Time:        0.956 s, estimated 1 s
Ran all test suites.
```

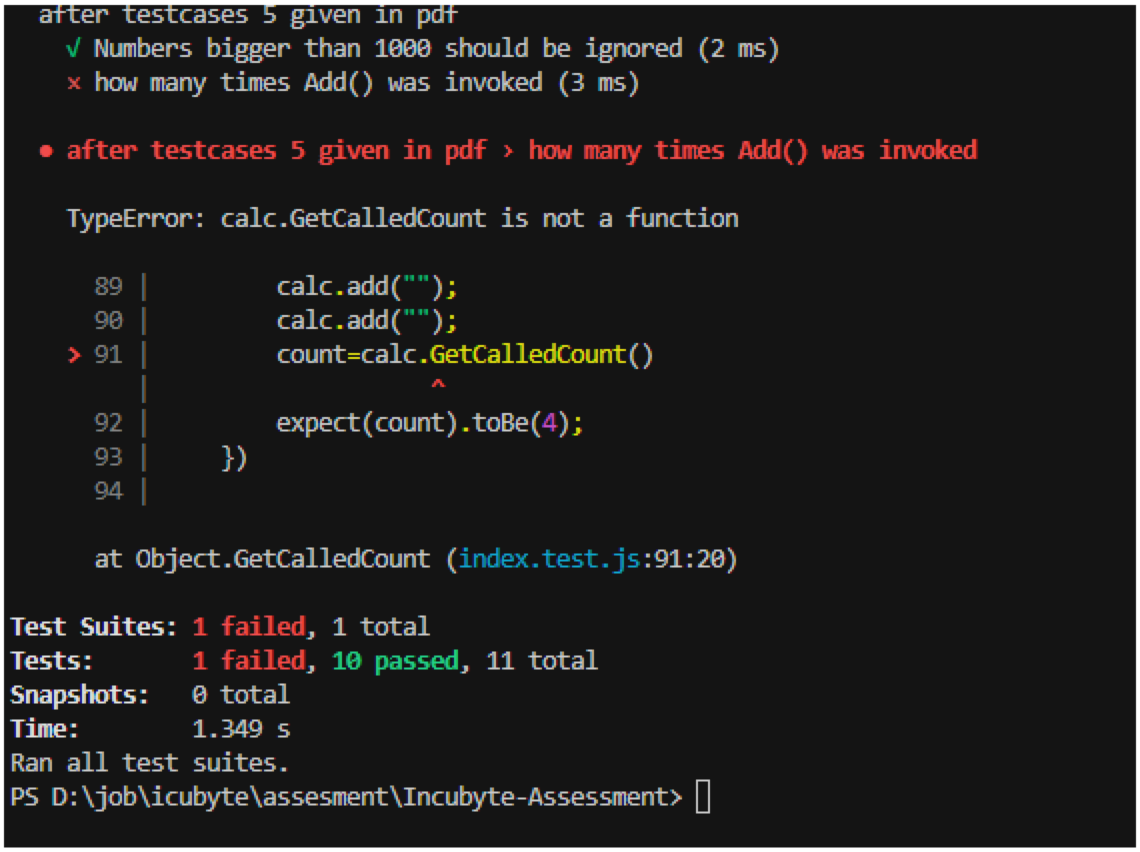
Testcase 7:-returns how many times Add() was invoked.

First, we write a test case without declaring any method and call it.

```
  test("how many times Add() was invoked",()=>{
      const calc=new StringCalculator();
      calc.add("");
      calc.add("");
      calc.add("");
      calc.add("");
      count=calc.GetCalledCount()
      expect(count).toBe(4);
  })
});
```

Explanation:-Here , we can see that i call add method 4 times,i expect GetCalledCount method returns 4 but it failed the test case because i have not write method in class.

Output:-



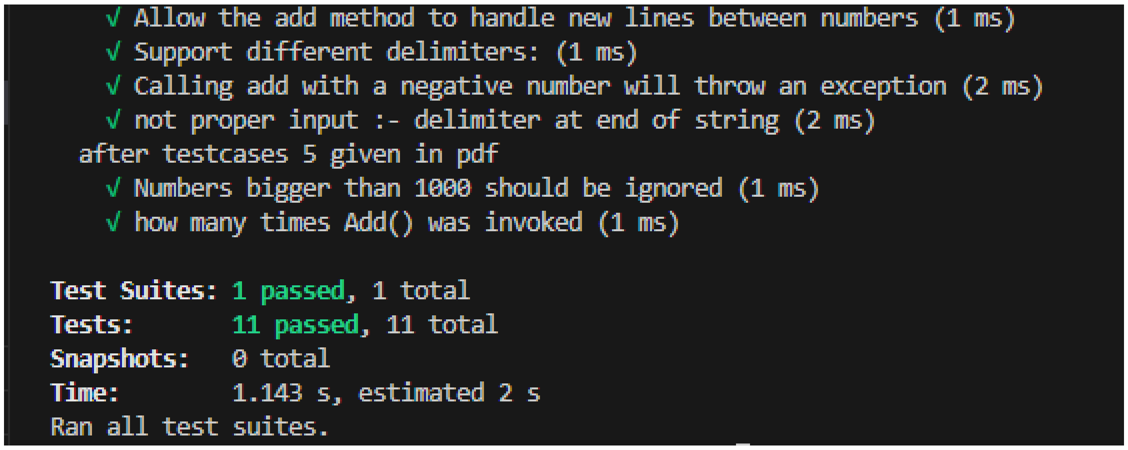
```
  after testcases 5 given in pdf
    √ Numbers bigger than 1000 should be ignored (2 ms)
    × how many times Add() was invoked (3 ms)

  ● after testcases 5 given in pdf › how many times Add() was invoked

    TypeError: calc.GetCalledCount is not a function

      89 |         calc.add("");
      90 |         calc.add("");
    > 91 |         count=calc.GetCalledCount()
         |                    ^
      92 |         expect(count).toBe(4);
      93 |     })
      94 |

      at Object.GetCalledCount (index.test.js:91:20)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 10 passed, 11 total
Snapshots:   0 total
Time:        1.349 s
Ran all test suites.
PS D:\job\icubyte\assesment\Incubyte-Assessment>
```

Now for refactoring code, i did very minimal changes in my code,just add method with logic

```
 #counting_of_add=0;

    GetCalledCount(){
        return this.#counting_of_add;
    }

    add(numbers) {
        this.#counting_of_add++;
        if (numbers == "") return 0;
        let delimiters = "\n,"; //default delimiters
```

Explanation:- Here you can see a snipshot of my code where i take one variable for counting how many times add method calls and return that to outside through public method GetCalledCount().

Output:-



```
      √ Allow the add method to handle new lines between numbers (1 ms)
      √ Support different delimiters: (1 ms)
      √ Calling add with a negative number will throw an exception (2 ms)
      √ not proper input :- delimiter at end of string (2 ms)
    after testcases 5 given in pdf
      √ Numbers bigger than 1000 should be ignored (1 ms)
      √ how many times Add() was invoked (1 ms)

  Test Suites: 1 passed, 1 total
  Tests:       11 passed, 11 total
  Snapshots:   0 total
  Time:        1.143 s, estimated 2 s
  Ran all test suites.
```

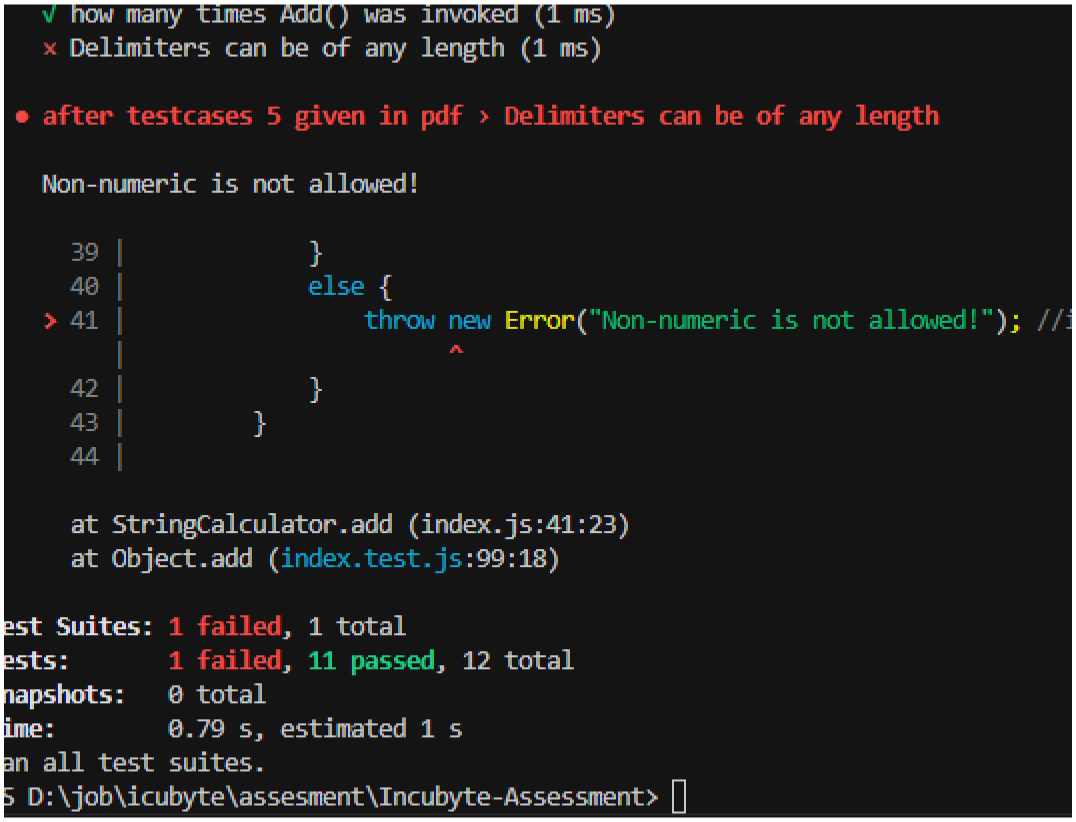
Testcase 8:-Delimiters can be of any length

Here you see a testcase for it.

```
test("Delimiters can be of any length", () => {
    const calc = new StringCalculator();
    numbers="//[***]\n1***2***3";
    sum=numbers.add(numbers);
    expect(sum).toBe(6);
});
```

Explanation:- Currently, this test case fails because there is no handlater code for any length delimiters.

Output:-



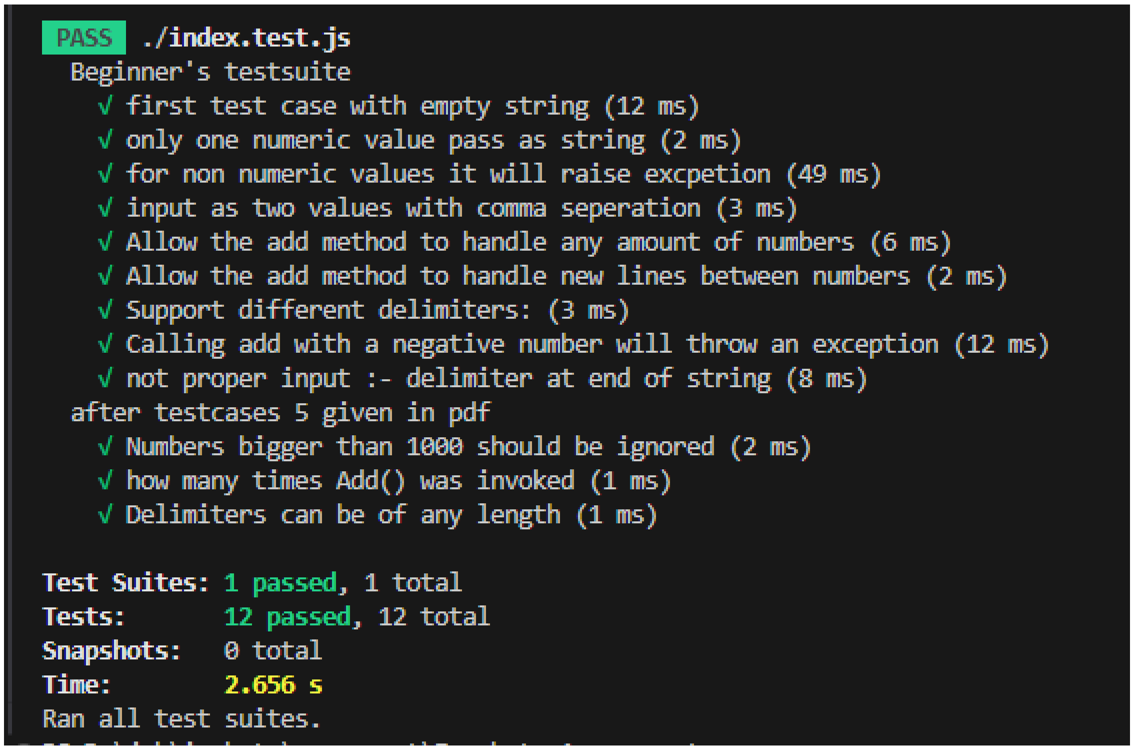
```
  √ how many times Add() was invoked (1 ms)
  × Delimiters can be of any length (1 ms)

● after testcases 5 given in pdf › Delimiters can be of any length

  Non-numeric is not allowed!

    39 |             }
    40 |             else {
  > 41 |                 throw new Error("Non-numeric is not allowed!"); //i
       |                       ^
    42 |             }
    43 |         }
    44 |

    at StringCalculator.add (index.js:41:23)
    at Object.add (index.test.js:99:18)

est Suites: 1 failed, 1 total
ests:       1 failed, 11 passed, 12 total
napshots:   0 total
ime:        0.79 s, estimated 1 s
an all test suites.
S D:\job\icubyte\assesment\Incubyte-Assessment>
```

I again reflector my code for resolve this
index.js

```
class StringCalculator {
  #counting_of_add = 0;

  GetCalledCount() {
    return this.#counting_of_add;
  }

  add(numbers) {
    this.#counting_of_add++;
    if (numbers == "") return 0;

    let negative_numbers = []; //contains negative numbers
    let delimiter = /[\n,]/;


    if (numbers.startsWith("//")) {
      const delimiterLineEnd = numbers.indexOf("\n");
      const customDelim = numbers.substring(2, delimiterLineEnd);
      if(customDelim[0]=='['){
        customDelim=customDelim.substring(1,customDelim.length-1);
      }
      delimiter = new RegExp(`\\n|,|${this.escapeRegex(customDelim)}`);
      numbers = numbers.substring(delimiterLineEnd + 1);
    }

    numbers = numbers.split(delimiter);
```

```
   let sum = 0; //for initial value of sum as zero
    for (let i = 0; i < numbers.length; i++) {
      if (!isNaN(parseInt(numbers[i]))) { //check if non numeric values
        if (numbers[i] < 0) negative_numbers.push(numbers[i]); //if number is
negative then add it here
        if (numbers[i] <= 1000) {
          sum += parseInt(numbers[i]); //do addition of each values if it is less
than or equal to 1000
        }
      }
      else {
        throw new Error("Non-numeric is not allowed!"); //if non numeric values
then raise Exception
      }
    }

    if (negative_numbers.length > 0) {
      let CommonMessage = "negative numbers not allowed"; //numbers are
dynamic but base message here is common

      let ErrorMessage = CommonMessage + " " + negative_numbers.join(',');
      throw new Error(ErrorMessage);
    }

    return sum; //return final sum
  }
  escapeRegex(str) {
    return str.replace(/[.*+?^${}()|[\]\\]/g, '\\$&');
  }
}

module.exports = StringCalculator;
```

Explanation:- here i use to check if there is [is or not after // and if yes then remove [ ] surrounded by delimiters and i have created one function escapeRegex which just replaces the character with \\$& if the character belongs to a given list. The character belongs to the string from the replace function call.



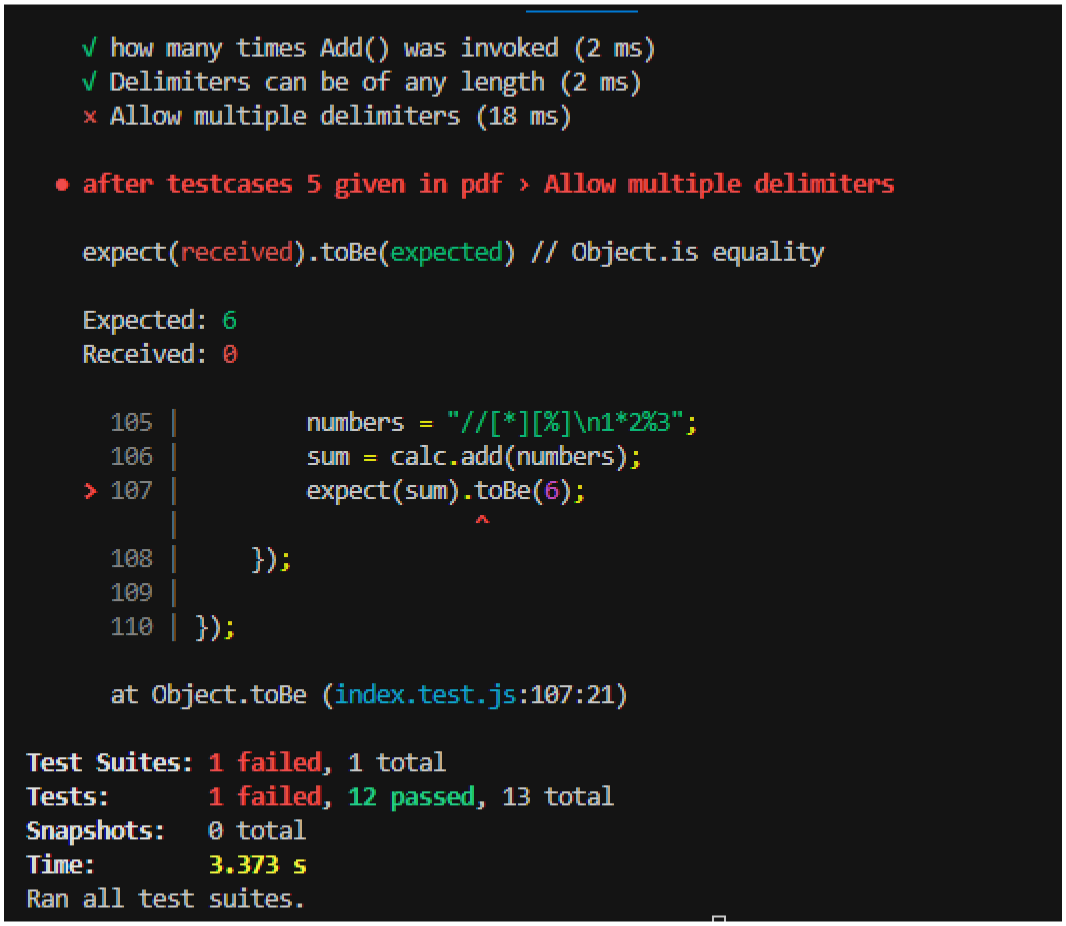
```
PASS  ./index.test.js
  Beginner's testsuite
    √ first test case with empty string (12 ms)
    √ only one numeric value pass as string (2 ms)
    √ for non numeric values it will raise excpetion (49 ms)
    √ input as two values with comma seperation (3 ms)
    √ Allow the add method to handle any amount of numbers (6 ms)
    √ Allow the add method to handle new lines between numbers (2 ms)
    √ Support different delimiters: (3 ms)
    √ Calling add with a negative number will throw an exception (12 ms)
    √ not proper input :- delimiter at end of string (8 ms)
  after testcases 5 given in pdf
    √ Numbers bigger than 1000 should be ignored (2 ms)
    √ how many times Add() was invoked (1 ms)
    √ Delimiters can be of any length (1 ms)

Test Suites: 1 passed, 1 total
Tests:       12 passed, 12 total
Snapshots:   0 total
Time:        2.656 s
Ran all test suites.
```

Testcase 9:-Allow multiple delimiters

You can use multiple delimiters if you want to use it.
I have written one test case for it and it fails.
You can see my testcase for it as below.

```
  test("Allow multiple delimiters", () => {
      const calc = new StringCalculator();
      numbers = "//[*][%]\n1*2%3";
      sum = calc.add(numbers);
      expect(sum).toBe(6);
  });
```

Explanation:- Here i pass multiple delimiters via passing it inside [] and you can use [] as you want.
Output:-



```
    √ how many times Add() was invoked (2 ms)
    √ Delimiters can be of any length (2 ms)
    × Allow multiple delimiters (18 ms)

  ● after testcases 5 given in pdf › Allow multiple delimiters

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

    Expected: 6
    Received: 0

      105 |         numbers = "//[*][%]\n1*2%3";
      106 |         sum = calc.add(numbers);
    > 107 |         expect(sum).toBe(6);
          |                     ^
      108 |     });
      109 |
      110 | });

      at Object.toBe (index.test.js:107:21)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 12 passed, 13 total
Snapshots:   0 total
Time:        3.373 s
Ran all test suites.
```

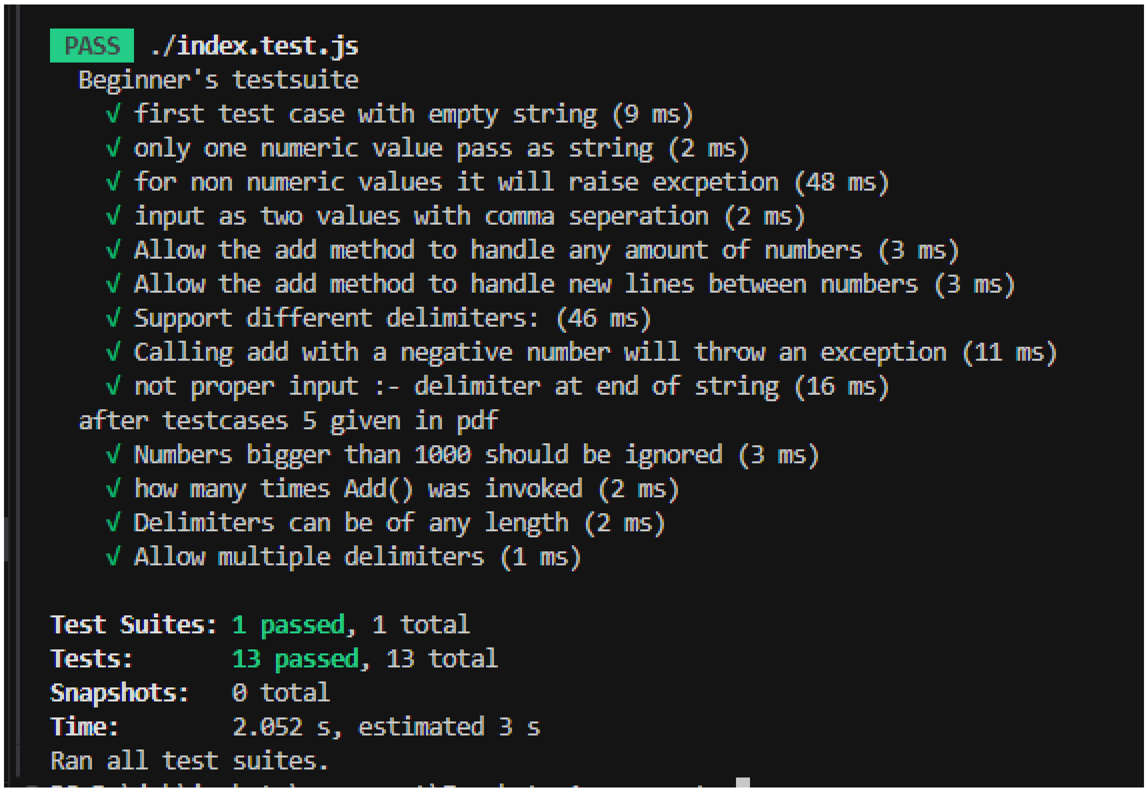
I reflector this code,you can see code below

```
class StringCalculator {
  #counting_of_add = 0;

  //return Delimiter expression so it will be easy to pass in split method
  //i create this seperat function because of dont want all logic in one function and
so then it will be easy to understand.
  #GetDelimiter(customDelim) {

    if (customDelim[0] == '[') { //here check if there is [ or not if not then only
one delim is take
      customDelim = customDelim.substring(1, customDelim.length - 1);
//remove first [ and last ] brackets
      customDelim=customDelim
      .split('][')
      .map( delim => `${this.escapeRegex(delim)}`).join('|'); //now i replace ][
with / and also i use excapeRegex function
    }
    let delimiter = new RegExp(`\\n|,|${customDelim}`);
    return delimiter;
  }


  GetCalledCount() {
    return this.#counting_of_add;
  }

  add(numbers) {
    this.#counting_of_add++;
    if (numbers == "") return 0;

    let negative_numbers = []; //contains negative numbers
    let delimiter = /[\n,]/;
```

```
        if (numbers.startsWith("//")) {
            const delimiterLineEnd = numbers.indexOf("\n");
            let customDelim = numbers.substring(2, delimiterLineEnd);
            delimiter=this.#GetDelimiter(customDelim)
            numbers = numbers.substring(delimiterLineEnd + 1);
        }

        numbers = numbers.split(delimiter);



        let sum = 0; //for initial value of sum as zero
        for (let i = 0; i < numbers.length; i++) {
            if (!isNaN(parseInt(numbers[i]))) { //check if non numeric values
                if (numbers[i] < 0) negative_numbers.push(numbers[i]); //if number is
negative then add it here
                if (numbers[i] <= 1000) {
                    sum += parseInt(numbers[i]); //do addition of each values if it is less
than or equal to 1000
                }
            }
            else {
                throw new Error("Non-numeric is not allowed!"); //if non numeric values
then raise Exception
            }
        }

        if (negative_numbers.length > 0) {
            let CommonMessage = "negative numbers not allowed"; //numbers are
dynamic but base message here is common

            let ErrorMessage = CommonMessage + " " + negative_numbers.join(',');
            throw new Error(ErrorMessage);
        }
        return sum; //return final sum
    }
    //this function is use for if string contain any regix specific characters so it will
be replace by \\character
    escapeRegex(str) {
        return str.replace(/[.*+?^${}()|[\]\\]/g, '\\$&');
    }
}
module.exports = StringCalculator;
```

Explanation:- i have create one function GetDelimiter which job is only for ready delimiter , i just doing adding all custom delimiter in string and between all delimiter i use '|' character to show or operate. So any of this character split method found inside the string will take that point as a split point and it will split that string on that point.

Output:-



```
 PASS  ./index.test.js
  Beginner's testsuite
    √ first test case with empty string (9 ms)
    √ only one numeric value pass as string (2 ms)
    √ for non numeric values it will raise excpetion (48 ms)
    √ input as two values with comma seperation (2 ms)
    √ Allow the add method to handle any amount of numbers (3 ms)
    √ Allow the add method to handle new lines between numbers (3 ms)
    √ Support different delimiters: (46 ms)
    √ Calling add with a negative number will throw an exception (11 ms)
    √ not proper input :- delimiter at end of string (16 ms)
  after testcases 5 given in pdf
    √ Numbers bigger than 1000 should be ignored (3 ms)
    √ how many times Add() was invoked (2 ms)
    √ Delimiters can be of any length (2 ms)
    √ Allow multiple delimiters (1 ms)

Test Suites: 1 passed, 1 total
Tests:       13 passed, 13 total
Snapshots:   0 total
Time:        2.052 s, estimated 3 s
Ran all test suites.
```

Testcase 10:-handle multiple delimiters with length longer than one char

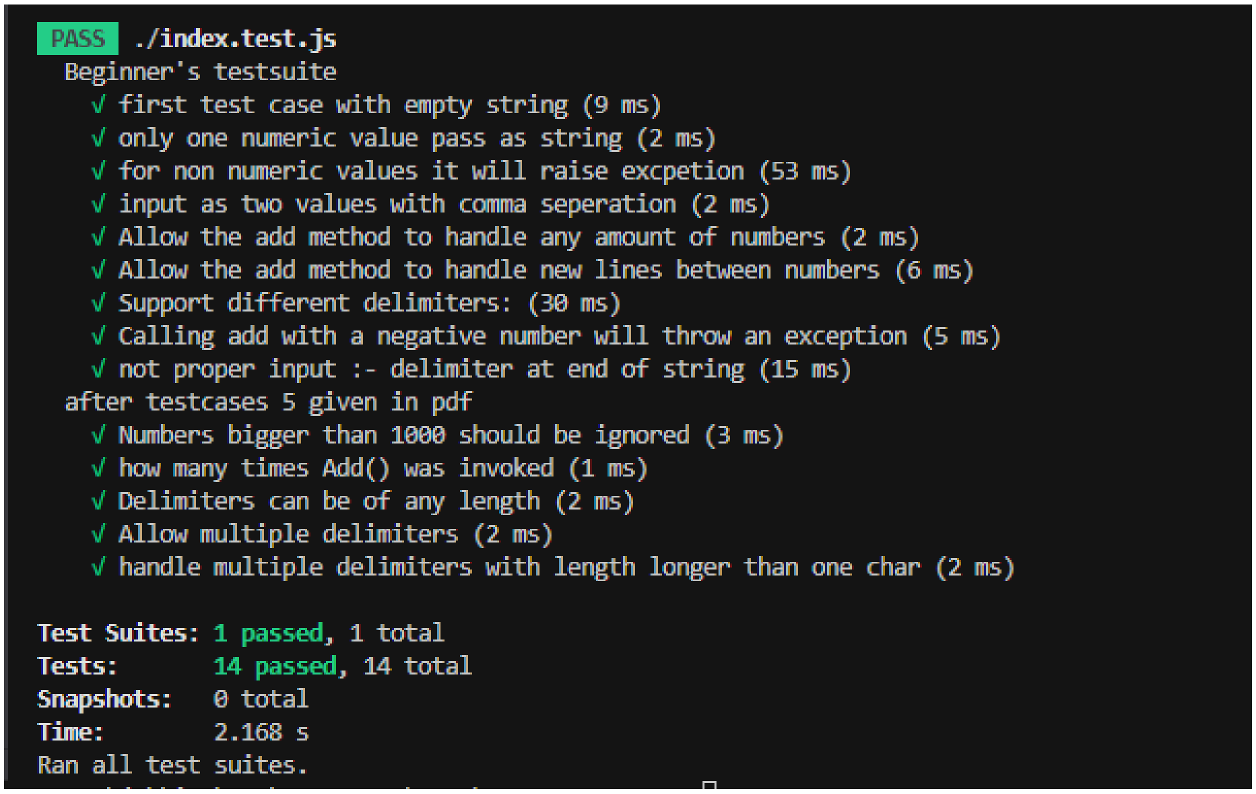
In my previous code version it automatically handled.Here i mentioned test case and output which shows passing it output only.

Testcase

```
test("handle multiple delimiters with length longer than one char", () => {
    const calc = new StringCalculator();
    numbers = "//[**][%%]\n1**2%%3";
    sum = calc.add(numbers);
    expect(sum).toBe(6);
  });
```

Explanation:- This is the same as the previous testcase. The only change is only here you can use a delimiter with more than one character.

Output:-



```
 PASS  ./index.test.js
  Beginner's testsuite
    √ first test case with empty string (9 ms)
    √ only one numeric value pass as string (2 ms)
    √ for non numeric values it will raise excpetion (53 ms)
    √ input as two values with comma seperation (2 ms)
    √ Allow the add method to handle any amount of numbers (2 ms)
    √ Allow the add method to handle new lines between numbers (6 ms)
    √ Support different delimiters: (30 ms)
    √ Calling add with a negative number will throw an exception (5 ms)
    √ not proper input :- delimiter at end of string (15 ms)
  after testcases 5 given in pdf
    √ Numbers bigger than 1000 should be ignored (3 ms)
    √ how many times Add() was invoked (1 ms)
    √ Delimiters can be of any length (2 ms)
    √ Allow multiple delimiters (2 ms)
    √ handle multiple delimiters with length longer than one char (2 ms)

Test Suites: 1 passed, 1 total
Tests:       14 passed, 14 total
Snapshots:   0 total
Time:        2.168 s
Ran all test suites.
```