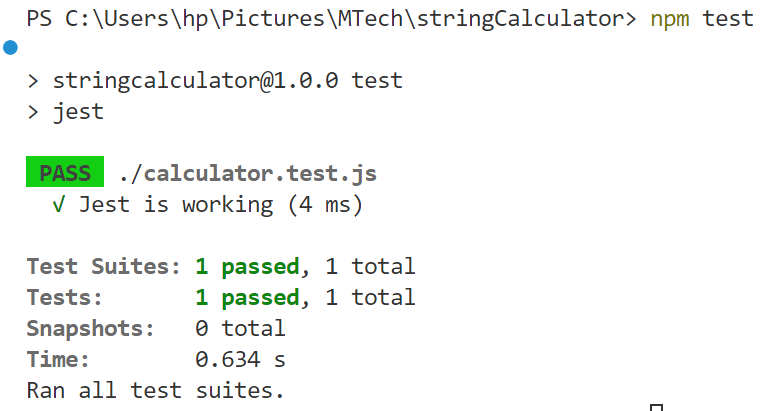
npm install --save-dev jest



test('Jest is working', () => {

expect(1 + 1).toBe(2);

});

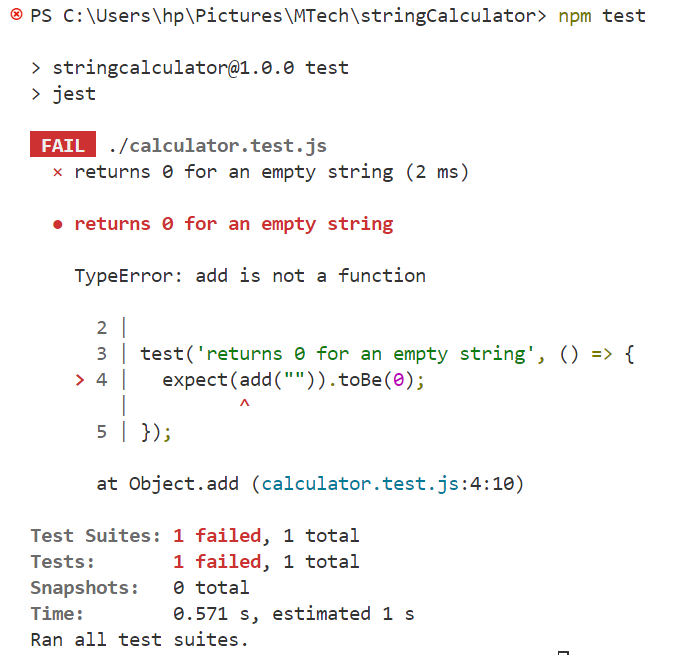


const { add } = require('./calculator');

test('returns 0 for an empty string', () => {

expect(add("")).toBe(0);

});

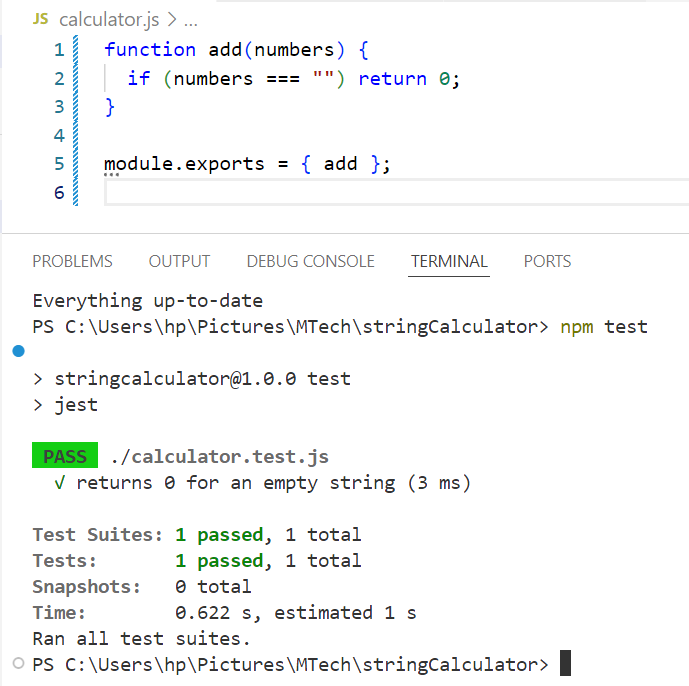


function add(numbers) {

  if (numbers === "") return 0;

}

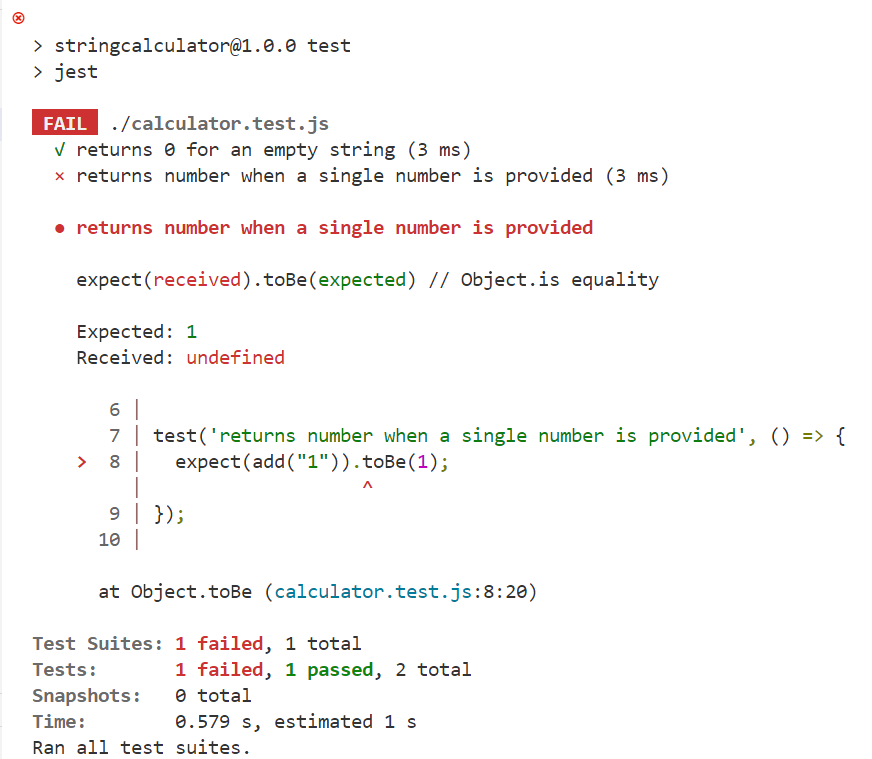
module.exports = { add };



test('returns number when a single number is provided', () => {

  expect(add("1")).toBe(1);

});



function add(numbers) {

// Specification 1: Return 0 for an empty string

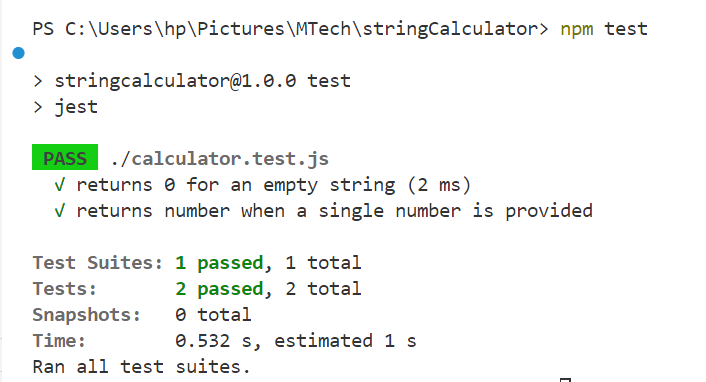
if (numbers === "") return 0;

// Specification 2: Return the number itself if only one number is given

return parseInt(numbers);

}

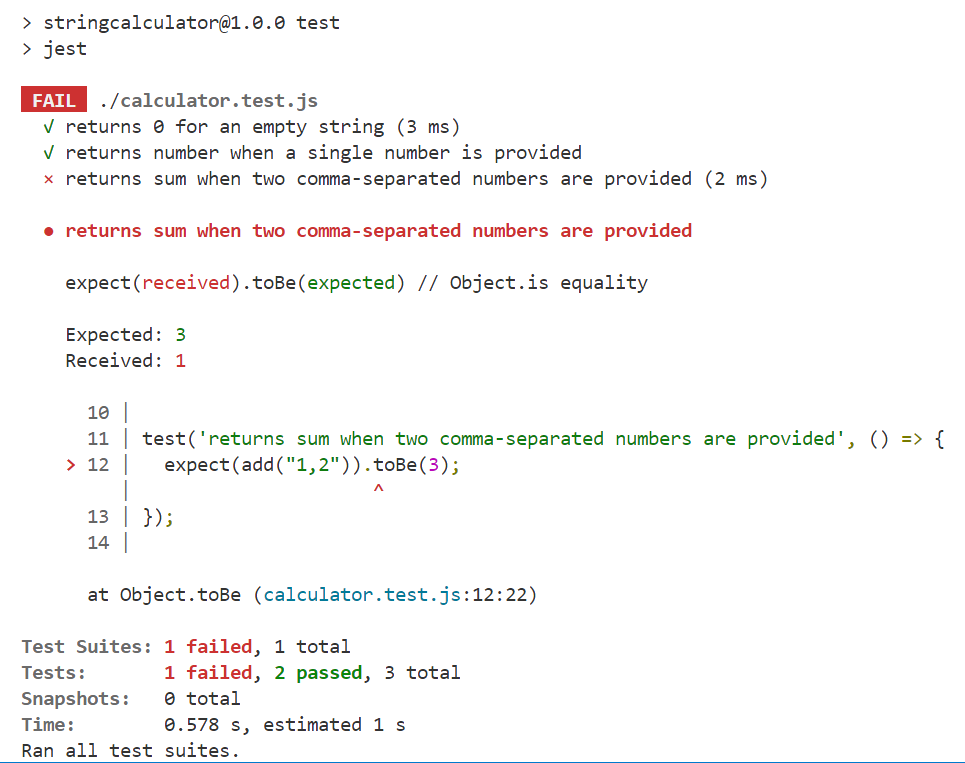
module.exports = { add };



test('returns sum when two comma-separated numbers are provided', () => {

expect(add("1,2")).toBe(3);

});



function add(numbers) {

// Specification 1: Return 0 for an empty string

if (numbers === "") return 0;

// Specification 2: Return the number itself if only one number is given

if (!numbers.includes(",")) {

return parseInt(numbers);

}

// Specification 3: If input contains commas, split and sum two numbers

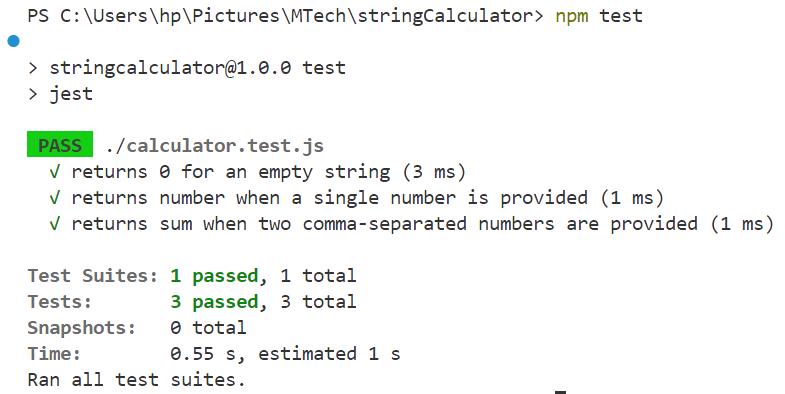
const parts = numbers.split(",");

const sum = parts.reduce((acc, num) => acc + parseInt(num), 0);

return sum;

}

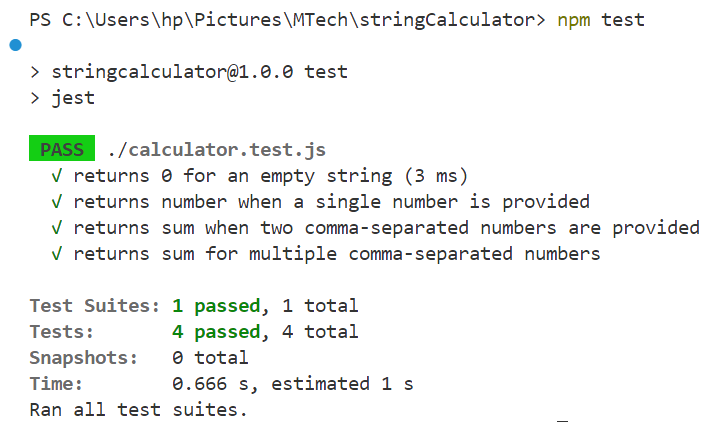
module.exports = { add };



test('returns sum for multiple comma-separated numbers', () => {

expect(add("1,2,3,4")).toBe(10);

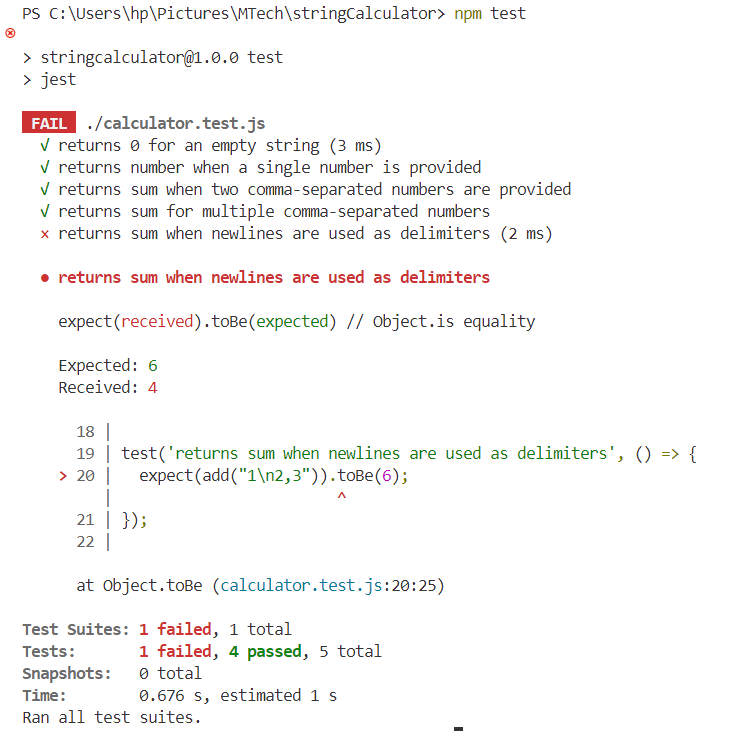
});



test('returns sum when newlines are used as delimiters', () => {

expect(add("1\n2,3")).toBe(6);

});



function add(numbers) {

// Specification 1: Return 0 for an empty string

if (numbers === "") return 0;

// Specification 2: Return the number itself if only one number is given

if (!numbers.includes(",") && !numbers.includes("\n")) {

return parseInt(numbers);

}

// Specification 3: Replace newlines with commas to normalize delimiters

const normalized = numbers.replace(/\n/g, ",");

// Specification 4: Split by comma and sum all numbers

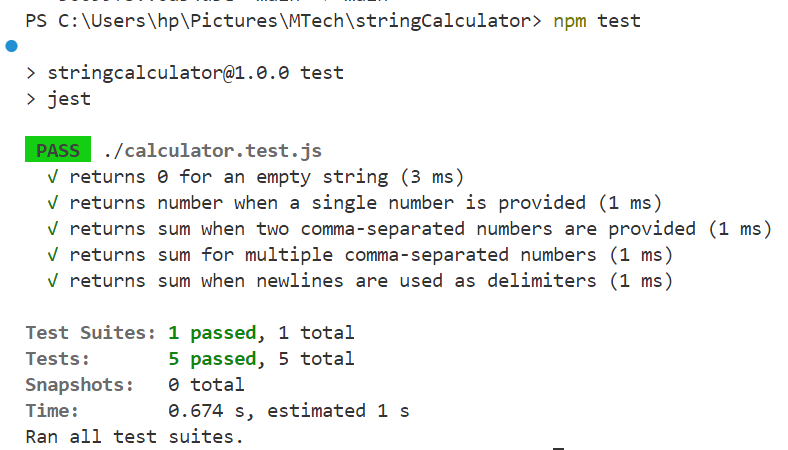
const parts = normalized.split(",");

const sum = parts.reduce((acc, num) => acc + parseInt(num), 0);

return sum;

}

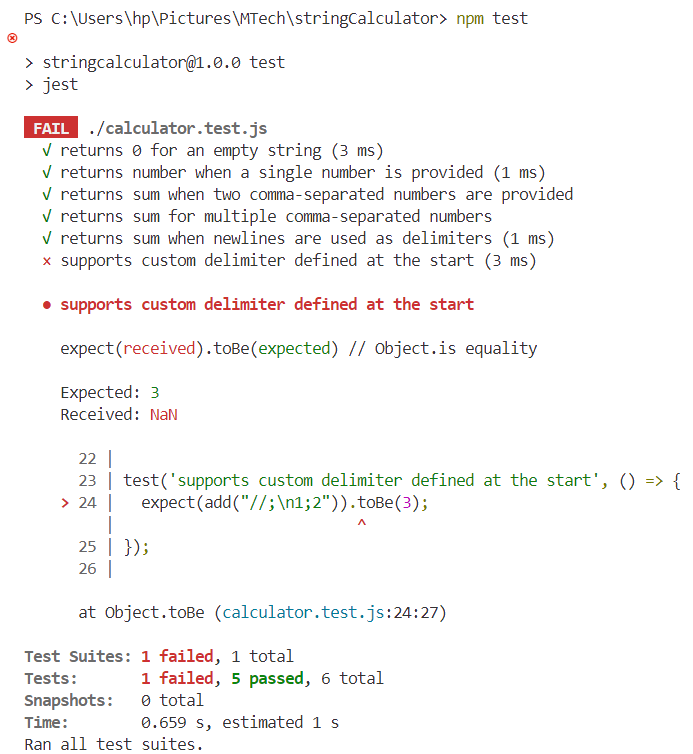
module.exports = { add };



test('supports custom delimiter defined at the start', () => {

expect(add("//;\n1;2")).toBe(3);

});



function add(numbers) {

// Specification 1: Return 0 for an empty string

if (numbers === "") return 0;

let delimiter = /,|\n/; // Default delimiters: comma or newline

// Specification 5: Handle custom delimiter syntax like "//;\n1;2"

if (numbers.startsWith("//")) {

// Extract delimiter from the first line

const delimiterLine = numbers.split("\n")[0];

delimiter = new RegExp(numbers[2]); // e.g., ";" becomes /;/

// Remove delimiter declaration from input

numbers = numbers.split("\n")[1];

}

// Split numbers using the resolved delimiter(s)

const parts = numbers.split(delimiter);

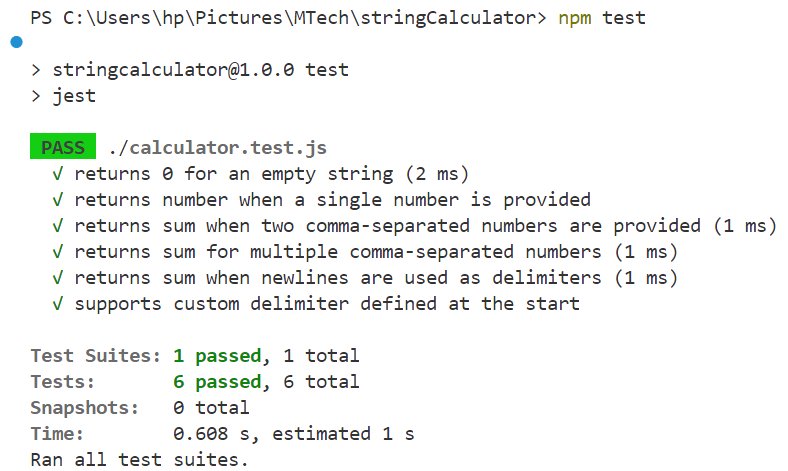
// Sum all the parts

const sum = parts.reduce((acc, num) => acc + parseInt(num), 0);

return sum;

}

module.exports = { add };



test('throws an exception when a negative number is used', () => {

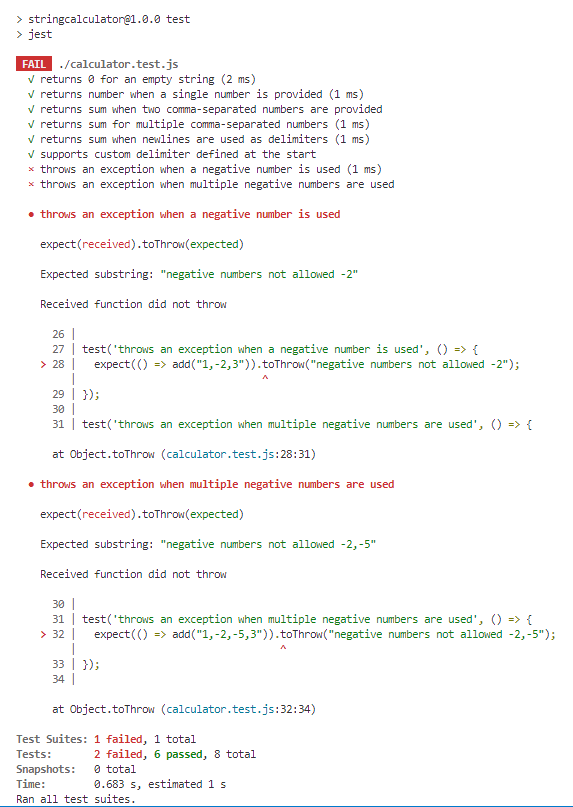
  expect(() => add("1,-2,3")).toThrow("negative numbers not allowed -2");

});

test('throws an exception when multiple negative numbers are used', () => {

  expect(() => add("1,-2,-5,3")).toThrow("negative numbers not allowed -2,-5");

});



function add(numbers) {

// Specification 1: Return 0 for an empty string

if (numbers === "") return 0;

let delimiter = /,|\n/; // Default delimiters: comma or newline

// Specification 5: Handle custom delimiter syntax like "//;\n1;2"

if (numbers.startsWith("//")) {

const delimiterLine = numbers.split("\n")[0];

delimiter = new RegExp(numbers[2]); // Extract delimiter from line

numbers = numbers.split("\n")[1]; // Remove delimiter declaration

}

// Split the numbers using resolved delimiter(s)

const parts = numbers.split(delimiter);

// Convert parts to integers

const parsedNumbers = parts.map(num => parseInt(num));

// Specification 6: Throw an exception for any negative numbers

const negatives = parsedNumbers.filter(n => n < 0);

if (negatives.length > 0) {

throw new Error(`negative numbers not allowed ${negatives.join(",")}`);

}

// Sum all valid numbers

const sum = parsedNumbers.reduce((acc, num) => acc + num, 0);

return sum;

}

module.exports = { add };

