# Solutions-Exercises: Unit Testing and Error Handling

# Error Handling

## 1.Request Validator

function validateRequest(request){

let methods = ['GET', 'POST', 'DELETE', 'CONNECT'];

let uriPattern = /^[\w.]+$/;

let versions = ['HTTP/0.9', 'HTTP/1.0', 'HTTP/1.1', 'HTTP/2.0'];

let messagePattern = /^[^<>\\&'"]\*$/g;

let headerKeys = Object.keys(request);

if(!headerKeys.includes('method') || !methods.includes(request.method)){

throw new Error('Invalid request header: Invalid Method');

}

if(!headerKeys.includes('uri') || !uriPattern.test(request.uri)){

throw new Error('Invalid request header: Invalid URI');

}

if(!headerKeys.includes('version') || !versions.includes(request.version)){

throw new Error('Invalid request header: Invalid Version');

}

if(!headerKeys.includes('message') || !messagePattern.test(request.message)){

throw new Error('Invalid request header: Invalid Message');

}

return request;

}

console.log(validateRequest({ method: 'GET', uri: 'svn.public.catalog', version: 'HTTP/1.1', message: ''}));

validateRequest({

method: 'OPTIONS',

uri: 'git.master',

version: 'HTTP/1.1',

message: '-recursive'

});

validateRequest({

method: 'POST',

uri: 'home.bash',

message: 'rm -rf /\*'

});

|  |
| --- |
| function requestValidator(obj) { |
|  | let validMethods = ["GET", "POST", "DELETE", "CONNECT"]; |
|  |  |
|  | if(! (obj.method && validMethods.includes(obj.method))){ |
|  | throw new Error("Invalid request header: Invalid Method"); |
|  | } |
|  |  |
|  | let uriRegex = /^[\w.]+$/; |
|  |  |
|  | if(! (obj.uri && ( uriRegex.test(obj.uri) || obj.uri == "\*"))){ |
|  | throw new Error("Invalid request header: Invalid URI"); |
|  | } |
|  |  |
|  | let validVerisons = ["HTTP/0.9", "HTTP/1.0", "HTTP/1.1", "HTTP/2.0"]; |
|  |  |
|  | if(! (obj.version && validVerisons.includes(obj.version))){ |
|  | throw new Error("Invalid request header: Invalid Version"); |
|  | } |
|  |  |
|  | let messageRegex = /^[^<>**\\**&'"]\*$/; |
|  |  |
|  | if(! ( obj.hasOwnProperty("message") && (messageRegex.test(obj.message) || obj.message == ""))) { |
|  | throw new Error("Invalid request header: Invalid Message"); |
|  | } |
|  |  |
|  | return obj; |
|  | } |

|  |
| --- |
| function validateRequest(request) { |
|  | const METHODS = ["GET", "POST", "DELETE", "CONNECT"]; |
|  | const URI\_REGEX = /^[\w.]+$/; |
|  | const VERSIONS = ["HTTP/0.9", "HTTP/1.0", "HTTP/1.1", "HTTP/2.0"]; |
|  | const MESSAGE\_REGEX = /^[^<>**\\**&'"]\*$/g; |
|  | let objectKeys = Object.keys(request); |
|  | if (!objectKeys.includes("method") || !METHODS.includes(request.method)) { |
|  | throwError("Method"); |
|  | } |
|  | if (!objectKeys.includes("uri") || !URI\_REGEX.test(request.uri)) { |
|  | throwError("URI"); |
|  | } |
|  | if (!objectKeys.includes("version") || !VERSIONS.includes(request.version)) { |
|  | throwError("Version"); |
|  | } |
|  | if (!objectKeys.includes("message") || !MESSAGE\_REGEX.test(request.message)) { |
|  | throwError("Message"); |
|  | } |
|  |  |
|  | return request; |
|  |  |
|  | function throwError(invalidHeader) { |
|  | throw new Error(`Invalid request header: Invalid ${invalidHeader}`); |
|  | } |
|  | } |

# Unit Testing

## 2.Even or Odd

function isOddOrEven(string) {

if (typeof(string) !== 'string') {

return undefined;

}

if (string.length % 2 === 0) {

return "even";

}

return "odd";

}

module.exports = {isOddOrEven};

let expect = require('chai').expect;

let isOddOrEven = require('../02EvenOrOdd').isOddOrEven;

//In Judge must be paste without this above

describe('isOddOrEven - checks if string length is odd or even', function(){

it('should return undefined for []', function(){

expect(isOddOrEven([])).to.be.undefined;

});

it('should return undefined for {}', function(){

expect(isOddOrEven({})).to.be.undefined;

});

it('should return undefined for 10', function(){

expect(isOddOrEven(10)).to.be.undefined;

});

it('should return undefined for NaN', function(){

expect(isOddOrEven(NaN)).to.be.undefined;

});

it('should return undefined for new Date(2019-11-11)', function(){

expect(isOddOrEven(new Date(2019-11-11))).to.be.undefined;

});

it('should return undefined for null', function(){

expect(isOddOrEven(null)).to.be.undefined;

});

it('should return undefined for undefined', function(){

expect(isOddOrEven(undefined)).to.be.undefined;

});

it('should return odd for a', function(){

expect(isOddOrEven('a')).to.be.equal('odd');

});

it('should return odd for aba', function(){

expect(isOddOrEven('aba')).to.be.equal('odd');

});

it('should return odd for abcde', function(){

expect(isOddOrEven('abcde')).to.be.equal('odd');

});

it('should return odd for 1234567', function(){

expect(isOddOrEven('1234567')).to.be.equal('odd');

});

it('should return odd for one space', function(){

expect(isOddOrEven(' ')).to.be.equal('odd');

});

it('should return even for empty string', function(){

expect(isOddOrEven('')).to.be.equal('even');

});

it('should return even for ab', function(){

expect(isOddOrEven('ab')).to.be.equal('even');

});

it('should return even for abkl;p', function(){

expect(isOddOrEven('abkl;p')).to.be.equal('even');

});

it('should return even for 1234567890', function(){

expect(isOddOrEven('1234567890')).to.be.equal('even');

});

it('should return even for six spaces', function(){

expect(isOddOrEven(' ')).to.be.equal('even');

});

});

|  |
| --- |
| function isOddOrEven(string) { |
|  | if (typeof(string) !== 'string') { |
|  | return undefined; |
|  | } |
|  | if (string.length % 2 === 0) { |
|  | return "even"; |
|  | } |
|  |  |
|  | return "odd"; |
|  | } |
|  |  |
|  | module.exports = {isOddOrEven}; |

|  |
| --- |
| let expect = require('chai').expect; |
|  | let should = require('chai').should; |
|  | let assert = require('chai').assert; |
|  | let isOddOrEven = require('../01. Even or Odd').isOddOrEven; |
|  |  |
|  | describe('isOddOrEven', function () { |
|  | it('with a number parameter, should return undefined', function () { |
|  | expect(isOddOrEven(13)).to.equal(undefined, "Function did not return the correct result!"); |
|  | }); |
|  | it('with a object parameter, should return undefined', function () { |
|  | isOddOrEven({name: 'pesho'}).should.equal(undefined, "Function did not return the correct result!"); |
|  | }); |
|  | it('with an even length string should return "even"', function () { |
|  | assert.equal(isOddOrEven("roar"), "even", "Function did not return correct result!"); |
|  | }); |
|  | it('with an odd length string, should return "odd"', function () { |
|  | expect(isOddOrEven("pesho")).to.equal("odd", "Function did not return correct result!"); |
|  | }); |
|  | it('with multiple consecutive checks, should return correct values', function () { |
|  | expect(isOddOrEven("cat")).to.equal('odd', "Function did not return correct result!"); |
|  | expect(isOddOrEven("alabala")).to.equal("odd", "Function did not return correct result!"); |
|  | expect(isOddOrEven("is it even")).to.equal("even", "Function did not return correct result!"); |
|  | }); |
|  | }); |

|  |
| --- |
| function isOddOrEven(string) { |
|  | if (typeof (string) !== 'string') { |
|  | return undefined; |
|  | } |
|  | if (string.length % 2 === 0) { |
|  | return "even"; |
|  | } |
|  |  |
|  | return "odd"; |
|  | } |
|  |  |
|  | module.exports = { |
|  | isOddOrEven |
|  | }; |

|  |
| --- |
| const expect = require("chai").expect; |
|  | const isOddOrEven = require("../2\_EvenOrOdd").isOddOrEven; |
|  |  |
|  | describe("isOddOrEven", function () { |
|  | it("should return undefined for []", function () { |
|  | expect(isOddOrEven([])).to.be.undefined; |
|  | }); |
|  | it("should return undefined for {}}", function () { |
|  | expect(isOddOrEven({})).to.be.undefined; |
|  | }); |
|  | it("should return undefined for 10", function () { |
|  | expect(isOddOrEven(10)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for new Date(2007,10,10)", function () { |
|  | expect(isOddOrEven(new Date(2007, 10, 10))).to.be.undefined; |
|  | }); |
|  | it("should return undefined for null", function () { |
|  | expect(isOddOrEven(null)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for undefined", function () { |
|  | expect(isOddOrEven(null)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for NaN", function () { |
|  | expect(isOddOrEven(NaN)).to.be.undefined; |
|  | }); |
|  |  |
|  | it("should return odd for abc", function () { |
|  | expect(isOddOrEven("abc")).to.be.equal("odd"); |
|  | }); |
|  | it("should return odd for a", function () { |
|  | expect(isOddOrEven("a")).to.be.equal("odd"); |
|  | }); |
|  | it("should return odd for abcdefg", function () { |
|  | expect(isOddOrEven("abcdefg")).to.be.equal("odd"); |
|  | }); |
|  |  |
|  | it("should return even for ", function () { |
|  | expect(isOddOrEven("")).to.be.equal("even"); |
|  | }); |
|  | it("should return even for aa", function () { |
|  | expect(isOddOrEven("aa")).to.be.equal("even"); |
|  | }); |
|  | it("should return even for aaaaaa", function () { |
|  | expect(isOddOrEven("aaaaaa")).to.be.equal("even"); |
|  | }); |
|  | it("should return even for 123456", function () { |
|  | expect(isOddOrEven("123456")).to.be.equal("even"); |
|  | }); |
|  | }) |

## 3.Char Lookup

function lookupChar(string, index) {

if (typeof(string) !== 'string' || !Number.isInteger(index)) {

return undefined;

}

if (string.length <= index || index < 0) {

return "Incorrect index";

}

return string.charAt(index);

}

module.exports = {lookupChar};

let expect = require('chai').expect;

let lookupChar = require('../03CharLookup').lookupChar;

//In Judge must be paste without this above

describe('lookupChar - returns a character by a given index in the string', function(){

it('with a non-string first parameter, should return undefined', function(){

expect(lookupChar(13, 0)).to.equal(undefined, 'The function did not return the correct result!');

});

it('with a non-number secnd parameter, should return undefined', function(){

expect(lookupChar('pesho', 'gosho')).to.equal(undefined, 'The function did not return the correct result!');

});

it('with a floating point number second parameter, should return undefined', function(){

expect(lookupChar('pesho', 3.14)).to.equal(undefined, 'The function did not return the correct message!');

});

it('with an incorrect index value, should return Incorrect index', function(){

expect(lookupChar('pesho', 10)).to.equal('Incorrect index', 'The function did not return the correct value!');

});

it('with a negative index value, should return Incorrect index', function(){

expect(lookupChar('pesho', -1)).to.equal('Incorrect index', 'The function did not return the correct value!');

});

it('with an index value equal to string length, should return Incorrect index', function(){

expect(lookupChar('pesho', 5)).to.equal('Incorrect index', 'The function did not return the correct value!');

});

it('with correct parameters, should return correct value', function(){

expect(lookupChar('pesho', 0)).to.equal('p', 'The function did not return the correct result!');

});

it('with correct parameters, should return correct value', function(){

expect(lookupChar('pesho', 3)).to.equal('h', 'The function did not return the correct result!');

});

it('with correct parameters, should return correct value', function(){

expect(lookupChar(' 123456789', 6)).to.equal('6', 'The function did not return the correct result!');

});

});

|  |
| --- |
| function lookupChar(string, index) { |
|  | if (typeof(string) !== 'string' || !Number.isInteger(index)) { |
|  | return undefined; |
|  | } |
|  | if (string.length <= index || index < 0) { |
|  | return "Incorrect index"; |
|  | } |
|  |  |
|  | return string.charAt(index); |
|  | } |
|  |  |
|  | module.exports = {lookupChar}; |

|  |
| --- |
| let expect = require('chai').expect; |
|  | let should = require('chai').should(); |
|  | let lookupChar = require('../02. Char Lookup').lookupChar; |
|  |  |
|  | describe("lookupChar", function () { |
|  | it('with a non-string first parameter should return undefined', function () { |
|  | expect(lookupChar(13, 0)).to.equal(undefined, "The function did not return correct result"); |
|  | }); |
|  | it('with a non-string second parameter should return undefined', function () { |
|  | expect(lookupChar("pesho", "gosho")).to.equal(undefined, "The function did not return correct result"); |
|  | }); |
|  | it('with a floating point number second parameter should return undefined', function () { |
|  | expect(lookupChar("pesho", 3.12)).to.equal(undefined, "The function did not return correct result"); |
|  | }); |
|  | it('with an incorrect index value should return incorrect index', function () { |
|  | expect(lookupChar("gosho", 13)).to.equal("Incorrect index", "The function did not return correct result"); |
|  | }); |
|  | it('with a negative index value should return incorrect index', function () { |
|  | expect(lookupChar("stamat", -1)).to.equal("Incorrect index", "The function did not return correct result"); |
|  | }); |
|  | it('with an index value equal to string length, should return incorrect index', function () { |
|  | expect(lookupChar("pesho", 5)).to.equal("Incorrect index", "The function did not return correct result"); |
|  | }); |
|  | it('with correct parameters, should return correct value', function () { |
|  | expect(lookupChar("pesho", 0)).to.equal("p", "The function did not return correct result"); |
|  | }); |
|  | it('with correct parameters should return correct value', function () { |
|  | expect(lookupChar("stamat", 3)).to.equal("m", "The function did not return correct result"); |
|  | }); |
|  | }); |

|  |
| --- |
|  |
| function lookupChar(string, index) { |
|  | if (typeof (string) !== 'string' || !Number.isInteger(index)) { |
|  | return undefined; |
|  | } |
|  | if (string.length <= index || index < 0) { |
|  | return "Incorrect index"; |
|  | } |
|  |  |
|  | return string.charAt(index); |
|  | } |
|  |  |

|  |
| --- |
| const expect = require("chai").expect; |
|  | const lookupChar = require("../3\_CharLookup").lookupChar; |
|  |  |
|  | describe("lookupChar", function () { |
|  | it("should return undefined for {{},1}", function () { |
|  | expect(lookupChar({}, 1)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for {[],1}", function () { |
|  | expect(lookupChar([], 1)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for {10,1}", function () { |
|  | expect(lookupChar(10, 1)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for {NaN,1}", function () { |
|  | expect(lookupChar(NaN, 1)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for {null,1}", function () { |
|  | expect(lookupChar(null, 1)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for {3.5,1}", function () { |
|  | expect(lookupChar(3.5, 1)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for {undefined,1}", function () { |
|  | expect(lookupChar(undefined, 1)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for {asd,1.5}", function () { |
|  | expect(lookupChar("asd", 1.5)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for {asd,null}", function () { |
|  | expect(lookupChar("asd", null)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for {asd,undefined}", function () { |
|  | expect(lookupChar("asd", undefined)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for {asd,null}", function () { |
|  | expect(lookupChar("asd", null)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for {asd,NaN}", function () { |
|  | expect(lookupChar("asd", NaN)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for {asd,[]}", function () { |
|  | expect(lookupChar("asd", [])).to.be.undefined; |
|  | }); |
|  |  |
|  | it("should return Incorrect index for {asd,100}", function () { |
|  | expect(lookupChar("asd", 100)).to.be.equal("Incorrect index"); |
|  | }); |
|  | it("should return Incorrect index for {a,2}", function () { |
|  | expect(lookupChar("a", 2)).to.be.equal("Incorrect index"); |
|  | }); |
|  | it("should return Incorrect index for {a,-1}", function () { |
|  | expect(lookupChar("a", -1)).to.be.equal("Incorrect index"); |
|  | }); |
|  | it("should return Incorrect index for {a,1}", function () { |
|  | expect(lookupChar("a", 1)).to.be.equal("Incorrect index"); |
|  | }); |
|  | it("should return Incorrect index for {abcdefg,7}", function () { |
|  | expect(lookupChar("abcdefg", 7)).to.be.equal("Incorrect index"); |
|  | }); |
|  |  |
|  | it("should return correct char for {a,0}", function () { |
|  | expect(lookupChar("a", 0)).to.be.equal("a"); |
|  | }); |
|  | it("should return correct char for {abcdefg,6}", function () { |
|  | expect(lookupChar("abcdefg", 6)).to.be.equal("g"); |
|  | }); |
|  | it("should return correct char for {abc,1}", function () { |
|  | expect(lookupChar("abc", 1)).to.be.equal("b"); |
|  | }); |
|  | }) |

## 4.Math Enforcer

let mathEnforcer = {

addFive: function (num) {

if (typeof(num) !== 'number') {

return undefined;

}

return num + 5;

},

subtractTen: function (num) {

if (typeof(num) !== 'number') {

return undefined;

}

return num - 10;

},

sum: function (num1, num2) {

if (typeof(num1) !== 'number' || typeof(num2) !== 'number') {

return undefined;

}

return num1 + num2;

}

};

module.exports = {mathEnforcer};

let expect = require('chai').expect;

let assert = require('chai').assert;

let mathEnforcer = require('../04MathEnforcer').mathEnforcer;

//In Judge must be paste without this above

describe('mathEnforcer - adds 5, subtracts 10 or sums', function(){

it('should have addFive, subtractTen, sum', function(){

let mathEnforcerKeys = Object.keys(mathEnforcer).join(', ');

let expectedMathEnforcerKeys = 'addFive, subtractTen, sum';

assert.equal(mathEnforcerKeys, expectedMathEnforcerKeys)

});

describe('addFive - adds 5 to a given number', function(){

it('should return 5 for 0', function(){

expect(mathEnforcer.addFive(0)).to.be.equal(5);

});

it('should return 10 for 5', function(){

expect(mathEnforcer.addFive(5)).to.be.equal(10);

});

it('should return 1000005 for 1000000', function(){

expect(mathEnforcer.addFive(1000000)).to.be.equal(1000005);

});

it('should return -5 for -10', function(){

expect(mathEnforcer.addFive(-10)).to.be.equal(-5);

});

it('should return 0 for -5', function(){

expect(mathEnforcer.addFive(-5)).to.be.equal(0);

});

it('should return 10.18 for 5.18', function(){

expect(mathEnforcer.addFive(5.18)).to.be.closeTo(10.18, 0.01);

});

it('should return undefined for abc', function(){

expect(mathEnforcer.addFive("abc")).to.be.undefined;

});

it('should return undefined for "1"', function(){

expect(mathEnforcer.addFive("1")).to.be.undefined;

});

it('should return undefined for []', function(){

expect(mathEnforcer.addFive([])).to.be.undefined;

});

it('should return undefined for {}', function(){

expect(mathEnforcer.addFive({})).to.be.undefined;

});

it('should return undefined for new Date(2019, 10, 10)', function(){

expect(mathEnforcer.addFive(new Date(2019, 10, 10))).to.be.undefined;

});

it('should return undefined for null', function(){

expect(mathEnforcer.addFive(null)).to.be.undefined;

});

it('should return undefined for undefined', function(){

expect(mathEnforcer.addFive(undefined)).to.be.undefined;

});

it('should return undefined for true', function(){

expect(mathEnforcer.addFive(true)).to.be.undefined;

});

it('should return NaN for NaN', function(){

expect(mathEnforcer.addFive(NaN)).to.be.NaN;

});

});

describe('subtractTen - subtracts 10 from a given number', function(){

it('should return -10 for 0', function(){

expect(mathEnforcer.subtractTen(0)).to.be.equal(-10);

});

it('should return 0 for 10', function(){

expect(mathEnforcer.subtractTen(10)).to.be.equal(0);

});

it('should return -5 for 5', function(){

expect(mathEnforcer.subtractTen(5)).to.be.equal(-5);

});

it('should return 5 for 15', function(){

expect(mathEnforcer.subtractTen(15)).to.be.equal(5);

});

it('should return -15 for -5', function(){

expect(mathEnforcer.subtractTen(-5)).to.be.equal(-15);

});

it('should return 1.7 for 11.7', function(){

expect(mathEnforcer.subtractTen(11.7)).to.be.closeTo(1.7, 0.01);

});

it('should return undefined for abc', function(){

expect(mathEnforcer.subtractTen("abc")).to.be.undefined;

});

it('should return undefined for "1"', function(){

expect(mathEnforcer.subtractTen("1")).to.be.undefined;

});

it('should return undefined for []', function(){

expect(mathEnforcer.subtractTen([])).to.be.undefined;

});

it('should return undefined for {}', function(){

expect(mathEnforcer.subtractTen({})).to.be.undefined;

});

it('should return undefined for new Date(2019, 10, 10)', function(){

expect(mathEnforcer.subtractTen(new Date(2019, 10, 10))).to.be.undefined;

});

it('should return undefined for null', function(){

expect(mathEnforcer.subtractTen(null)).to.be.undefined;

});

it('should return undefined for undefined', function(){

expect(mathEnforcer.subtractTen(undefined)).to.be.undefined;

});

it('should return undefined for true', function(){

expect(mathEnforcer.subtractTen(true)).to.be.undefined;

});

it('should return NaN for NaN', function(){

expect(mathEnforcer.subtractTen(NaN)).to.be.NaN;

});

});

describe('sum - sums two given numbers', function(){

it('should return 0 for (0, 0)', function(){

expect(mathEnforcer.sum(0, 0)).to.be.equal(0);

});

it('should return 10 for (0, 10)', function(){

expect(mathEnforcer.sum(0, 10)).to.be.equal(10);

});

it('should return -10 for (-10, 0)', function(){

expect(mathEnforcer.sum(-10, 0)).to.be.equal(-10);

});

it('should return 100015 for (100000, 15)', function(){

expect(mathEnforcer.sum(100000, 15)).to.be.equal(100015);

});

it('should return 19 for (30, -11)', function(){

expect(mathEnforcer.sum(30, -11)).to.be.equal(19);

});

it('should return -41 for (-30, -11)', function(){

expect(mathEnforcer.sum(-30, -11)).to.be.equal(-41);

});

it('should return 1.9 for (0.8, 1.1)', function(){

expect(mathEnforcer.sum(0.8, 1.1)).to.be.closeTo(1.9, 0.01);

});

it('should return -0.1 for (-0.01, -0.09)', function(){

expect(mathEnforcer.sum(-0.01, -0.09)).to.be.closeTo(-0.1, 0.01);

});

it('should return undefined for ("abc", 10)', function(){

expect(mathEnforcer.sum("abc", 10)).to.be.undefined;

});

it('should return undefined for (10, "abc")', function(){

expect(mathEnforcer.sum(10, "abc")).to.be.undefined;

});

it('should return undefined for ("1", 10)', function(){

expect(mathEnforcer.sum("1", 10)).to.be.undefined;

});

it('should return undefined for (10, "1")', function(){

expect(mathEnforcer.sum(10, "1")).to.be.undefined;

});

it('should return undefined for ([], 10)', function(){

expect(mathEnforcer.sum([], 10)).to.be.undefined;

});

it('should return undefined for (10, [])', function(){

expect(mathEnforcer.sum(10, [])).to.be.undefined;

});

it('should return undefined for ({}, 10)', function(){

expect(mathEnforcer.sum({}, 10)).to.be.undefined;

});

it('should return undefined for (10, {})', function(){

expect(mathEnforcer.sum(10, {})).to.be.undefined;

});

it('should return undefined for (new Date(2019, 10, 10), 10)', function(){

expect(mathEnforcer.sum(new Date(2019, 10, 10), 10)).to.be.undefined;

});

it('should return undefined for (10, new Date(2019, 10, 10))', function(){

expect(mathEnforcer.sum(10, new Date(2019, 10, 10))).to.be.undefined;

});

it('should return undefined for (null, 10)', function(){

expect(mathEnforcer.sum(null, 10)).to.be.undefined;

});

it('should return undefined for (10, null)', function(){

expect(mathEnforcer.sum(10, null)).to.be.undefined;

});

it('should return undefined for (undefined, 10)', function(){

expect(mathEnforcer.sum(undefined, 10)).to.be.undefined;

});

it('should return undefined for (10, undefined)', function(){

expect(mathEnforcer.sum(10, undefined)).to.be.undefined;

});

it('should return undefined for (true, 10)', function(){

expect(mathEnforcer.sum(true, 10)).to.be.undefined;

});

it('should return undefined for (10, true)', function(){

expect(mathEnforcer.sum(10, true)).to.be.undefined;

});

it('should return NaN for (NaN, 10)', function(){

expect(mathEnforcer.sum(NaN, 10)).to.be.NaN;

});

it('should return NaN for (10, NaN)', function(){

expect(mathEnforcer.sum(10, NaN)).to.be.NaN;

});

});

});

|  |
| --- |
| let mathEnforcer = { |
|  | addFive: function (num) { |
|  | if (typeof(num) !== 'number') { |
|  | return undefined; |
|  | } |
|  | return num + 5; |
|  | }, |
|  | subtractTen: function (num) { |
|  | if (typeof(num) !== 'number') { |
|  | return undefined; |
|  | } |
|  | return num - 10; |
|  | }, |
|  | sum: function (num1, num2) { |
|  | if (typeof(num1) !== 'number' || typeof(num2) !== 'number') { |
|  | return undefined; |
|  | } |
|  | return num1 + num2; |
|  | } |
|  | }; |
|  |  |
|  | module.exports = {mathEnforcer}; |

|  |
| --- |
| let expect = require('chai').expect; |
|  | let mathEnforcer = require('../03. Math Enforcer').mathEnforcer; |
|  |  |
|  | describe("mathEnforcer", function () { |
|  | describe('addFive', function () { |
|  | it("should return undefined for non-number parameter",function () { |
|  | expect(mathEnforcer.addFive("5")).to.be.equal(undefined); |
|  | }); |
|  | it("should return correct result for positive integer parameter", function () { |
|  | expect(mathEnforcer.addFive(10)).to.be.equal(15); |
|  | }); |
|  | it("should return correct result for negative integer parameter", function () { |
|  | expect(mathEnforcer.addFive(-5)).to.be.equal(0); |
|  | }); |
|  | it("should return correct result for floating point parameter", function () { |
|  | expect(mathEnforcer.addFive(3.14)).to.be.closeTo(8.14, 0.01); |
|  | }); |
|  | }); |
|  |  |
|  | describe('subtractTen', function () { |
|  | it("should return undefined for non-number parameter",function () { |
|  | expect(mathEnforcer.subtractTen("5")).to.be.equal(undefined); |
|  | }); |
|  | it("should return correct result for positive integer parameter", function () { |
|  | expect(mathEnforcer.subtractTen(10)).to.be.equal(0); |
|  | }); |
|  | it("should return correct result for negative integer parameter", function () { |
|  | expect(mathEnforcer.subtractTen(-5)).to.be.equal(-15); |
|  | }); |
|  | it("should return correct result for floating point parameter", function () { |
|  | expect(mathEnforcer.subtractTen(3.14)).to.be.closeTo(-6.86, 0.01); |
|  | }); |
|  | }); |
|  |  |
|  | describe('sum', function () { |
|  | it("should return undefined for non-number first parameter", function () { |
|  | expect(mathEnforcer.sum("5", 5)).to.be.equal(undefined); |
|  | }); |
|  | it("should return undefined for non-number second parameter", function () { |
|  | expect(mathEnforcer.sum(5, "5")).to.be.equal(undefined); |
|  | }); |
|  | it("should return correct result for integer parameters", function () { |
|  | expect(mathEnforcer.sum(5, -3)).to.be.equal(2); |
|  | }); |
|  | it("should return correct result for floating point parameters", function () { |
|  | expect(mathEnforcer.sum(2.7, 3.4)).to.be.closeTo(6.1, 0.01); |
|  | }) |
|  | }) |
|  | }); |

|  |
| --- |
| let mathEnforcer = { |
|  | addFive: function (num) { |
|  | if (typeof (num) !== 'number') { |
|  | return undefined; |
|  | } |
|  | return num + 5; |
|  | }, |
|  | subtractTen: function (num) { |
|  | if (typeof (num) !== 'number') { |
|  | return undefined; |
|  | } |
|  | return num - 10; |
|  | }, |
|  | sum: function (num1, num2) { |
|  | if (typeof (num1) !== 'number' || typeof (num2) !== 'number') { |
|  | return undefined; |
|  | } |
|  | return num1 + num2; |
|  | } |
|  | }; |
|  |  |
|  | module.exports = { |
|  | mathEnforcer |
|  | }; |

|  |
| --- |
| const expect = require("chai").expect; |
|  | const assert = require("chai").assert; |
|  | const mathEnforcer = require("../4\_MathEnforcer").mathEnforcer; |
|  |  |
|  | describe("mathEnforcer", function () { |
|  | describe("addFive", function () { |
|  | it("should have addFive, subtractTen, sum", function () { |
|  | let keysJoined = Object.keys(mathEnforcer).join(", "); |
|  | let expectedKeysJoined = "addFive, subtractTen, sum"; |
|  | assert.equal(keysJoined, expectedKeysJoined); |
|  | }); |
|  |  |
|  | it("should return 15 for addFive(10)", function () { |
|  | expect(mathEnforcer.addFive(10)).to.be.equal(15); |
|  | }); |
|  | it("should return 5 for addFive(0)", function () { |
|  | expect(mathEnforcer.addFive(0)).to.be.equal(5); |
|  | }); |
|  | it("should return 1005 for addFive(1000)", function () { |
|  | expect(mathEnforcer.addFive(1000)).to.be.equal(1005); |
|  | }); |
|  | it("should return 0 for addFive(-5)", function () { |
|  | expect(mathEnforcer.addFive(-5)).to.be.equal(0); |
|  | }); |
|  | it("should return -5 for addFive(-10)", function () { |
|  | expect(mathEnforcer.addFive(-10)).to.be.equal(-5); |
|  | }); |
|  | it("should return 5.5 for addFive(0.5)", function () { |
|  | expect(mathEnforcer.addFive(0.5)).to.be.equal(5.5); |
|  | }); |
|  |  |
|  | it("should return undefined for addFive(asd)", function () { |
|  | expect(mathEnforcer.addFive("asd")).to.be.undefined; |
|  | }); |
|  | it("should return undefined for addFive([])", function () { |
|  | expect(mathEnforcer.addFive([])).to.be.undefined; |
|  | }); |
|  | it("should return undefined for addFive({})", function () { |
|  | expect(mathEnforcer.addFive({})).to.be.undefined; |
|  | }); |
|  | it("should return undefined for addFive(null)", function () { |
|  | expect(mathEnforcer.addFive(null)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for addFive(NaN)", function () { |
|  | expect(mathEnforcer.addFive(NaN)).to.be.NaN; |
|  | }); |
|  | it("should return undefined for addFive(new Date(2007,10,10))", function () { |
|  | expect(mathEnforcer.addFive(new Date(2007, 10, 10))).to.be.undefined; |
|  | }); |
|  | }); |
|  |  |
|  | describe("subtractTen", function () { |
|  | it("should return 0 for subtractTen(10)", function () { |
|  | expect(mathEnforcer.subtractTen(10)).to.be.equal(0); |
|  | }); |
|  | it("should return -10 for subtractTen(0)", function () { |
|  | expect(mathEnforcer.subtractTen(0)).to.be.equal(-10); |
|  | }); |
|  | it("should return 1005 for subtractTen(1015)", function () { |
|  | expect(mathEnforcer.subtractTen(1015)).to.be.equal(1005); |
|  | }); |
|  | it("should return -15 for subtractTen(-5)", function () { |
|  | expect(mathEnforcer.subtractTen(-5)).to.be.equal(-15); |
|  | }); |
|  | it("should return -5 for subtractTen(5)", function () { |
|  | expect(mathEnforcer.subtractTen(5)).to.be.equal(-5); |
|  | }); |
|  | it("should return 3.14 for subtractTen(13.4)", function () { |
|  | expect(mathEnforcer.subtractTen(13.14)).to.be.closeTo(3.14, 0.01); |
|  | }); |
|  | it("should return -6.86 for subtractTen(13.4)", function () { |
|  | expect(mathEnforcer.subtractTen(3.14)).to.be.closeTo(-6.86, 0.01); |
|  | }); |
|  |  |
|  |  |
|  | it("should return undefined for subtractTen(asd)", function () { |
|  | expect(mathEnforcer.subtractTen("asd")).to.be.undefined; |
|  | }); |
|  | it("should return undefined for subtractTen([])", function () { |
|  | expect(mathEnforcer.subtractTen([])).to.be.undefined; |
|  | }); |
|  | it("should return undefined for subtractTen({})", function () { |
|  | expect(mathEnforcer.subtractTen({})).to.be.undefined; |
|  | }); |
|  | it("should return undefined for subtractTen(null)", function () { |
|  | expect(mathEnforcer.subtractTen(null)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for subtractTen(NaN)", function () { |
|  | expect(mathEnforcer.subtractTen(NaN)).to.be.NaN; |
|  | }); |
|  | it("should return undefined for subtractTen(new Date(2007,10,10))", function () { |
|  | expect(mathEnforcer.subtractTen(new Date(2007, 10, 10))).to.be.undefined; |
|  | }); |
|  | }); |
|  |  |
|  | describe("sum", function () { |
|  | it("should return undefined for sum(asd, 1)", function () { |
|  | expect(mathEnforcer.sum("asd", 1)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for sum([], 1)", function () { |
|  | expect(mathEnforcer.sum([], 1)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for sum({}, 1)", function () { |
|  | expect(mathEnforcer.sum({}, 1)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for sum(null, 1)", function () { |
|  | expect(mathEnforcer.sum(null, 1)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for sum(NaN, 1)", function () { |
|  | expect(mathEnforcer.sum(NaN, 1)).to.be.NaN; |
|  | }); |
|  | it("should return undefined for sum(new Date(2007,10,10), 1)", function () { |
|  | expect(mathEnforcer.sum(new Date(2007, 10, 10), 1)).to.be.undefined; |
|  | }); |
|  |  |
|  | it("should return undefined for sum(1, asd)", function () { |
|  | expect(mathEnforcer.sum(1, "asd")).to.be.undefined; |
|  | }); |
|  | it("should return undefined for sum(1, [])", function () { |
|  | expect(mathEnforcer.sum(1, [])).to.be.undefined; |
|  | }); |
|  | it("should return undefined for sum(1, {})", function () { |
|  | expect(mathEnforcer.sum(1, {})).to.be.undefined; |
|  | }); |
|  | it("should return undefined for sum(1, null)", function () { |
|  | expect(mathEnforcer.sum(1, null)).to.be.undefined; |
|  | }); |
|  | it("should return undefined for sum(1, NaN)", function () { |
|  | expect(mathEnforcer.sum(1, NaN)).to.be.NaN; |
|  | }); |
|  | it("should return undefined for sum(1, new Date(2007,10,10))", function () { |
|  | expect(mathEnforcer.sum(1), new Date(2007, 10, 10)).to.be.undefined; |
|  | }); |
|  |  |
|  | it("should return 0 for sum(0,0)", function () { |
|  | expect(mathEnforcer.sum(0, 0)).to.be.equal(0); |
|  | }); |
|  | it("should return 1000 for sum(995,5)", function () { |
|  | expect(mathEnforcer.sum(995, 5)).to.be.equal(1000); |
|  | }); |
|  | it("should return -100 for sum(0,-100)", function () { |
|  | expect(mathEnforcer.sum(0, -100)).to.be.equal(-100); |
|  | }); |
|  | it("should return 1 for sum(1,0)", function () { |
|  | expect(mathEnforcer.sum(1, 0)).to.be.equal(1); |
|  | }); |
|  | it("should return 2 for sum(0,2)", function () { |
|  | expect(mathEnforcer.sum(0, 2)).to.be.equal(2); |
|  | }); |
|  | it("should return -50 for sum(-25,-25)", function () { |
|  | expect(mathEnforcer.sum(-25, -25)).to.be.equal(-50); |
|  | }); |
|  | it("should return -30 for sum(50,-80)", function () { |
|  | expect(mathEnforcer.sum(50, -80)).to.be.equal(-30); |
|  | }); |
|  | it("should return 0 for sum(2.5,2.5)", function () { |
|  | expect(mathEnforcer.sum(2.5, 2.5)).to.be.equal(5); |
|  | }); |
|  | it("should return -0.6 for sum(2.5,-3.1)", function () { |
|  | expect(mathEnforcer.sum(2.5, -3.1)).to.be.equal(2.5 + (-3.1)); |
|  | }); |
|  | it("should return 0.6 for sum(2.5,-3.1)", function () { |
|  | expect(mathEnforcer.sum(-3.1, 2.5)).to.be.equal((-3.1) + 2.5); |
|  | }); |
|  | it("should return 0.0005 for sum(0.0004,0.0001)", function () { |
|  | expect(mathEnforcer.sum(0.0004, 0.0001)).to.be.equal(0.0004 + 0.0001); |
|  | }); |
|  | it("should return 0.0004 for sum(0.0005,-0.0001)", function () { |
|  | expect(mathEnforcer.sum(0.0005, -0.0001)).to.be.equal(0.0005 + (-0.0001)); |
|  | }); |
|  | it("should return -0.0005 for sum(0.0004,0.0001)", function () { |
|  | expect(mathEnforcer.sum(-0.0004, -0.0001)).to.be.equal(-0.0004 + -0.0001); |
|  | }); |
|  | it("should return -0.0004 for sum(0.0005,-0.0001)", function () { |
|  | expect(mathEnforcer.sum(-0.0005, 0.0001)).to.be.equal(-0.0005 + 0.0001); |
|  | }); |
|  |  |
|  | it("should return 3.14 for sum(4.10,-0.96)", function () { |
|  | let result = mathEnforcer.sum(4.10, -0.96); |
|  | expect(result).to.be.closeTo(3.14, 0.1); |
|  | }); |
|  | it("should return -3.14 for sum(-4.10,0.96)", function () { |
|  | let result = mathEnforcer.sum(-4.10, 0.96); |
|  | expect(result).to.be.closeTo(-3.14, 0.1); |
|  | }); |
|  | it("should return correct result for floating point parameters", function () { |
|  | expect(mathEnforcer.sum(2.7, 3.4)).to.be.closeTo(6.1, 0.01); |
|  | }) |
|  | }); |
|  | }); |