# Lab: Unit Testing and Error Handling

Problems for in-class lab for the "JavaScript Advanced" course @ SoftUni. Submit your solutions in the SoftUni judge system at https://judge.softuni.bg/Contests/307/.

# Error Handling

## 1. Sub Sum

Write a JS function to sum a **range** of numeric elements from array. The function takes three parameters – the first is an **array**, the second is **start index** and the third is an **end index**. Both indexes are **inclusive**. Assume array elements may not be of type Number and cast everything. Implement the following error handling:

- if the **first element** is not an array, return **NaN**
- if the **start index** is less than zero, assume it is zero
- if the **end index** is outside the bounds of the array, assume it points to the last index of the array

### Input / Output

Your function must take three **parameters**. As output, **return** the resulting **sum** as instructed.

### Examples

| Sample Input | Sample Output |
|---|---|
| `subsum([10, 20, 30, 40, 50, 60], 3, 300)` | `150` |
| `subsum([1.1, 2.2, 3.3, 4.4, 5.5], -3, 1)` | `3.3` |
| `subsum([10, 'twenty', 30, 40], 0, 2)` | `NaN` |
| `subsum([], 1, 2)` | `0` |
| `subsum('text', 0, 2)` | `NaN` |

## 2. Playing Cards

Create a JS **factory function** that returns a **Card** object to hold a card's **`face`** and **`suit`**, both set trough the constructor. **Throw** an error if the card is initialized with invalid **face** or **suit** or if an attempt is made to change the **face** or **suit** of an existing instance to an invalid value.

- Valid card faces are: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A
- Valid card suits are: S (♠), H (♥), D (♦), C (♣)

Both face and suit are expected as an uppercase string. The class also needs to have a **`toString()`** method that prints the card's face and suit as a string. Use the following UTF code literals to represent the suits:

- \u2660 – Spades (♠)
- \u2665 – Hearts (♥)
- \u2666 – Diamonds (♦)
- \u2663 – Clubs (♣)

## Input / Output

The factory function must take two string parameters. The **`toString()`** method of the returned object must return a string.

Submit the factory function.

## Examples

| Sample Input | Sample Output |
|---|---|
| `console.log('' + makeCard('A', 'S'));` | A♠ |
| `console.log('' + makeCard('10', 'H'));` | 10♥ |
| `console.log('' + makeCard('1', 'C'));` | Error |

# 3. Deck of Cards

Write a JS function that takes a deck of cards as a string array and prints them as a sequence of cards (space separated). Print "Invalid card: [card]" when an invalid card definition is passed as input. Use the solution from the previous task to generate the cards.

## Input / Output

The function must take an array of strings as parameter. As output, print on the console the list of cards as strings, separated by space.

Submit a function that contains the **`makeCard`** factory function and other logic.

**deck.js**

```
function printDeckOfCards(cards) {
  function makeCard {
    // TODO use function definition from previous task
  }

  // TODO process input
}
```

## Examples

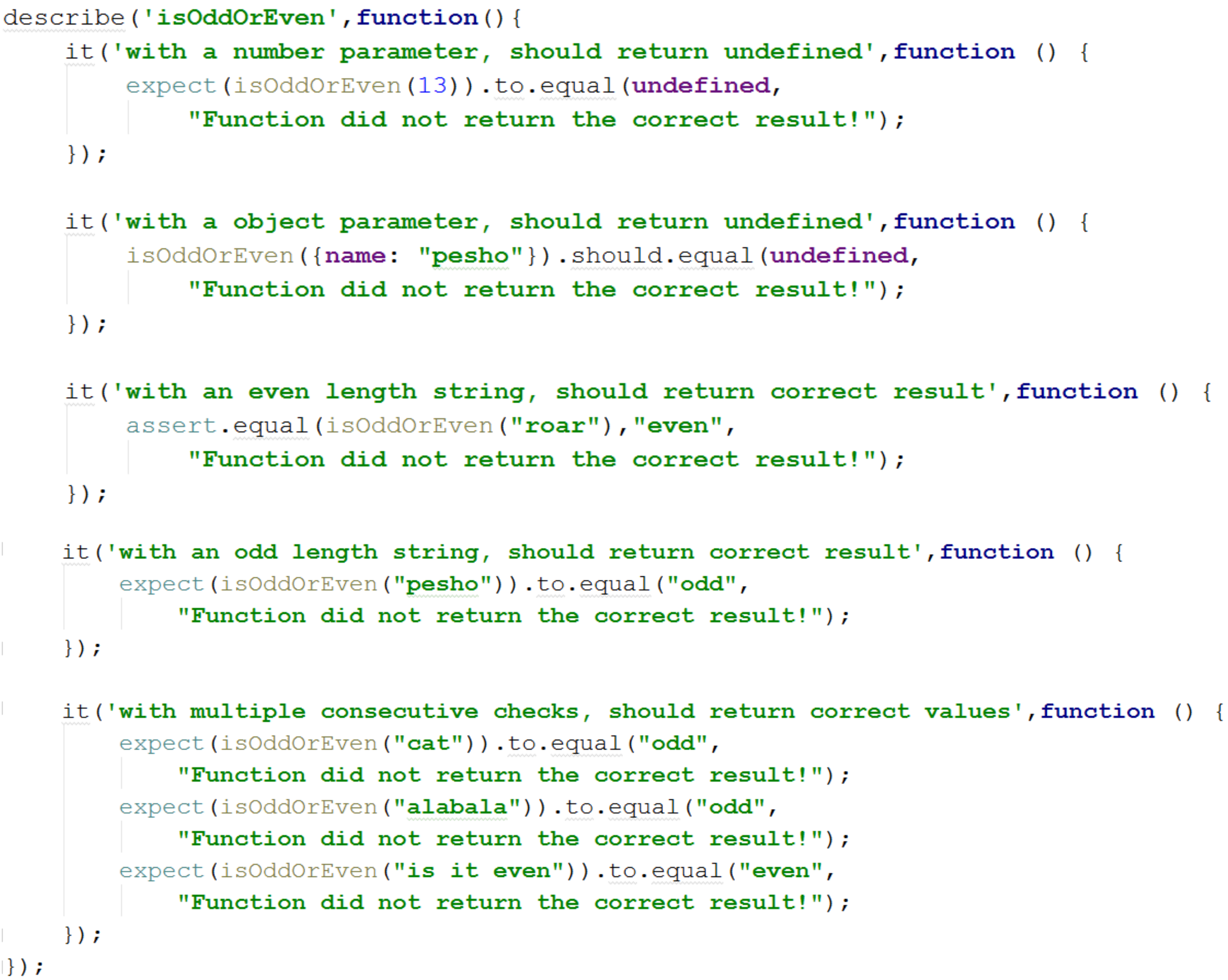
| Sample Input | Sample Output |
|---|---|
| `printDeckOfCards(['AS', '10D', 'KH', '2C']);` | A♠ 10♦ K♥ 2♣ |
| `printDeckOfCards(['5S', '3D', 'QD', '1C']);` | Invalid card: 1C |

# Unit Testing

The Unit Tests with Sinon and Mocha strategy gives you access to the following libraries to help you test your code - Mocha, Sinon, Chai, Sinon-Chai and jQuery.

You are required to **only submit the unit tests** for the object/function you are testing. The strategy provides access to Chai's **expect**, **assert** and **should** methods and jQuery.

## Example Submission



```
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 correct result',function () {
        assert.equal(isOddOrEven("roar"),"even",
            "Function did not return the correct result!");
    });

    it('with an odd length string, should return correct result',function () {
        expect(isOddOrEven("pesho")).to.equal("odd",
            "Function did not return the correct result!");
    });

    it('with multiple consecutive checks, should return correct values',function () {
        expect(isOddOrEven("cat")).to.equal("odd",
            "Function did not return the correct result!");
        expect(isOddOrEven("alabala")).to.equal("odd",
            "Function did not return the correct result!");
        expect(isOddOrEven("is it even")).to.equal("even",
            "Function did not return the correct result!");
    });
});
```

## 4. Sum of Numbers

Write Mocha tests to check the functionality of the following JS code:

**rgb-to-hex.js**

```
function sum(arr) {
    let sum = 0;
    for (num of arr)
        sum += Number(num);
    return sum;
}
```

Your tests will be supplied a function named **'sum'**. It needs to meet the following requirements:

- Takes and **array** of **numbers** as argument
- **Returns** the **sum** of the values of all elements inside the array

## 5. Check for Symmetry

Write Mocha tests to check the functionality of the following JS code:

**rgb-to-hex.js**

```
function isSymmetric(arr) {
    if (!Array.isArray(arr))
        return false; // Non-arrays are non-symmetric
    let reversed = arr.slice(0).reverse(); // Clone and reverse
    let equal = (JSON.stringify(arr) == JSON.stringify(reversed));
    return equal;
}
```

Your tests will be supplied a function named **'isSymmetric'**. It needs to meet the following requirements:

- Takes and **array** as argument
- **Returns false** for any input that isn't of the **correct type**
- **Returns true** if the input array is **symmetric** (first half is the same as the second half mirrored)
- Otherwise, returns **false**

## 6. RGB to Hex

Write Mocha tests to check the functionality of the following JS code:

**rgb-to-hex.js**

```
function rgbToHexColor(red, green, blue) {
    if (!Number.isInteger(red) || (red < 0) || (red > 255))
        return undefined; // Red value is invalid
    if (!Number.isInteger(green) || (green < 0) || (green > 255))
        return undefined; // Green value is invalid
    if (!Number.isInteger(blue) || (blue < 0) || (blue > 255))
        return undefined; // Blue value is invalid
    return "#" +
        ("0" + red.toString(16).toUpperCase()).slice(-2) +
        ("0" + green.toString(16).toUpperCase()).slice(-2) +
        ("0" + blue.toString(16).toUpperCase()).slice(-2);
}
```

Your tests will be supplied a function named **'rgbToHexColor'**, which takes three arguments. It needs to meet the following requirements:

- Takes three **integer numbers**, representing the red, green and blue values of an RGB color, each **within range [0…255]**
- **Returns** the same color in hexadecimal format as a **string** (e.g. '#FF9EAA')
- **Returns 'undefined'** if **any** of the input parameters are of **invalid type** or not in the expected **range**

## 7. Add / Subtract

Write Mocha tests to check the functionality of the following JS code:

**rgb-to-hex.js**

```
function createCalculator() {
    let value = 0;
    return {
        add: function(num) { value += Number(num); },
        subtract: function(num) { value -= Number(num); },
        get: function() { return value; }
    }
}
```

Your tests will be supplied a function named **'createCalculator'**. It needs to meet the following requirements:

- **Returns** a **module** (object), containing the functions **add**, **subtract** and **get** as **properties**
- Keeps an **internal sum** which **can't be modified** from the outside
- The functions **add** and **subtract** take a parameter that can be parsed as a number (either a number or a string containing a number) that is added or subtracted from the **internal sum**
- The function **get returns** the value of the **internal sum**