**Lab: Unit Testing with Mocha**

Problems for in-class lab for the [“JavaScript Advanced” course @ SoftUni](https://softuni.bg/courses/javascript-advanced). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/307/Unit-Testing-with-Mocha>.

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.

**Unit Testing**

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**





* **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
* **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**
* **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**
* **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**