**Lab: Unit Testing and Error Handling**

Problems for exercises and homework for the [" HYPERLINK "https://softuni.bg/modules/76"JavaScript HYPERLINK "https://softuni.bg/modules/76"Advanced" course @ HYPERLINK "https://softuni.bg/modules/76"SoftUni](https://softuni.bg/modules/76).

Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/2766/Unit-Testing-Lab>.

**Error Handling**

* **Sub Sum**

Write a function to sum a **range** of **numeric elements** from an array.

The function takes **three parameters** - the first is an **array**, the second is the **start index** and the third is the **end index**. Both indexes are **inclusive**. Have in mind that the 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, consider its value to be a **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 sum**.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| [10, 20, 30, 40, 50, 60], 3, 300 | 150 |
| [1.1, 2.2, 3.3, 4.4, 5.5], -3, 1 | 3.3000000000000003 |
| [10, 'twenty', 30, 40], 0, 2 | NaN |
| [], 1, 2 | 0 |
| 'text', 0, 2 | NaN |

* **Playing Cards**

Create a JS **factory function** that returns a **Card object** holding the card’s **face** and **suit**. **Throw an error** if the card is **initialized** with an **invalid** **face**.

* 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 object 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 takes **two string parameters**. The **toString()** method of the returned object must **return a string**.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 'A', 'S' | A♠ |
| '10', 'H' | 10♥ |
| '1', 'C' | Error |

* **Deck of Cards**

Write a function that takes **a deck of cards** as an **array of strings** and **prints** them as a **sequence** of cards (**space separated**). Use the solution from the **previous task** to generate the cards.

Print **`Invalid card: ${card}`** when an **invalid card** definition is passed as input.

**Input / Output**

The function takes an **array of strings** as a parameter. **Print** the list of cards as **string**, **separated by space**.

|  |
| --- |
| **deckOfCards.js** |
| **function printDeckOfCards(cards) {**  **function createCard (){**  ***// Use the solution from the previous task***  **}**  ***// TODO***  **}** |

**Examples**

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

**Unit Testing**

You are required to **submit only the unit tests** for the **object/function** you are testing.

* **Sum of Numbers**

Write tests to check the functionality of the following code:

|  |
| --- |
| **sumNumbers.js** |
| **function** *sum*(arr) {  **let** sum = 0;  **for** (let num **of** arr){  sum += Number(num);  }  **return** sum; } |

Your tests will be supplied with a function named **'sum()'**. It should meet the following requirements:

* Take an **array of numbers** as an argument
* **Return** the **sum** of the values of **all elements** inside the array
* **Check for Symmetry**

Write tests to check the functionality of the following code:

|  |
| --- |
| **checkForSymmetry.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 with a function named **'isSymmetric()'**. It should meet the following requirements:

* Take an **array** as an argument
* **Return false** for any input that isn’t of the **correct type**
* **Return true** if the input array is **symmetric**
* Otherwise, **return false**
* **RGB to Hex**

Write tests to check the functionality of the following 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 with a function named **'rgbToHexColor()'**, which takes **three arguments**. It should meet the following requirements:

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

Write tests to check the functionality of the following code:

|  |
| --- |
| **addSubtract.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 with a function named **'createCalculator()'**. It should meet the following requirements:

* **Return** **a** **module** (**object**), containing the functions **add()**, **subtract()** and **get()** as **properties**
* Keep an **internal sum** that **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**