Unit Tests with Jest

Unit tests are the smallest tests in software development in terms of granularity. Smaller than regression, system, alpha, beta, or pilot testing. The most popular testing framework is currently Jest. You can install it into your JavaScript project with the command:

```
npm i jest
```

They have a nice getting started section in their documentation. It navigates you through installing Jest, writing a function that you wish to test, then creating a test file to write your unit test, then lastly, executing the test.

Activity

For day 1 of writing unit tests, let's keep it simple 😳

Write a function called `addTwoNumbers`. This seems overly simple, but think of all the problems that could happen within a mathematical function that receives (hopefully) two parameters. *Hint:* Think about perhaps including default values for parameters.

Test Happy Day Scenarios:

- Any two valid numbers return the sum of those numbers.
- Two positive numbers should return a positive number.
- Two negative numbers should return a negative number.

Test Erroneous Scenarios:

- An array / object is provided
- The first parameter is not of type `Number`
- The second parameter is not of type `Number`
- The second parameter is missing
- Both parameters are missing
- The first parameter is `NaN`, `undefined`, `null`
- The second parameter is `NaN`, `undefined`, `null`
- The first parameter is a JavaScript `function`
- The second parameter is a JavaScript `function`
- Too many parameters are provided

Examples:

```
addTwoNumbers(1, 2)// Outputs: 3
addTwoNumbers(1, -2)// Outputs: -1
addTwoNumbers(3.1, 2.2)// Outputs: ?
addTwoNumbers([1, 2])// Outputs: ?
addTwoNumbers({one: 1, two: 2})// Outputs: ?
addTwoNumbers(3, 2, 1)// Outputs: ?
addTwoNumbers()// Outputs: ?
addTwoNumbers(1)// Outputs: ?
addTwoNumbers(undefined)// Outputs: ?
addTwoNumbers(null, null)// Outputs: ?
addTwoNumbers(1, true)// Outputs: ?
addTwoNumbers(NaN, -2)// Outputs: ?
addTwoNumbers("one", -2)// Outputs: ?
```

Bonus

Extend the function to use a custom error classes that you create. To create a custom error class, create a new file called "customErrors.js", and export a handful of appropriate custom errors that extend the `Error` class, for example, `UndefinedParametersError`, `MissingParametersError`, `IncorrectParamTypeError`, etc.

Good Practices

Document the function correctly using code comments. See the example:

```
/**
 * Reverses an input string.
 * @param {String} sentence is the string that will be reversed.
 * @returns a string, reversed.
 * @throws {WrongInputTypeError} if the argument is not a string
 * @example
 * reverseSentence("I love robots") // returns "robots love I"
 */
function reverseSentence(sentence) {
    if (!sentence) // empty, undefined, or null
        throw new Error("Input cannot be empty, null, or undefined")

    if (typeof sentence !== "string")
        throw new WrongInputTypeError("Input must be a string")

    return sentence.split(" ").filter(item => item !== "").reverse().join(" ")
}
```