JAVASCRIPT EXERCISE

Problems for exercising Javascript concepts

Tier One - Simple

1. Check Whether a Number Is Odd or Even

Create a function that takes a single integer as input and determines whether the number is **odd** or **even**.

Accepted Inputs and Validation Logics:

- A single integer number.
- If input is not an integer then return an error asking for a single integer as an input.

Expected Output:

- If the number is even, return **"The number X is even".**
- If the number is odd, return **"The number X is odd".**
- **"Error: Input must be an integer"** if the input is not an integer.

*(Here **X** is the given number)*

2. Count the number of vowels in a string

Create a function that counts the number of vowels in a given string.

Accepted Inputs and Validation Logics:

- A single string containing **one word only.**
- If a sentence (multiple words) or any other type of data is detected, return an error asking for a single string.

Expected Output:

- "**Total Vowel on the sentence \"Given String\" is: X**".
- "**Error: Please provide a single string**" if the input is a sentence or a non string data.

*(Here **X** is the number of vowels found)*

3. Remove duplicate values from an array

Create a function that removes duplicate values from an array.

The function should examine each element in the array and ensure that every value appears **only once** in the result. The original array must not be modified; instead, return a **new array** containing only unique values.

Accepted Inputs and Validation Logics:

- An array containing numbers, strings, or mixed data types.
- Any input that is not an array should be treated as invalid.

Expected Output:

- Return a new array with all duplicate values removed while preserving the original order.
- **"Error: Input must be an Array"** if the provided input is not an array.

4. Reverse a string

Create a function that reverses a given string.

The function should take a string as input and return a **new string** with the characters in reverse order. The original string should remain unchanged.

Accepted Inputs and Validation Logics:

- A single string containing **one word only.**
- If a sentence (multiple words) or any other type of data is detected, return an error asking for a single string.

Expected Output:

- Return the reversed version of the given string
- "**Error: Please provide a single string**" if the input is a sentence or a non string data.

5. Find the longest word in a sentence.

Create a function that finds the largest (longest) word in a given sentence.

The function should take a sentence as input and return the word that has the highest number of characters. If multiple words share the same maximum length, return the first one found. The original sentence should remain unchanged.

Accepted Inputs and Validation Logics:

- A sentence containing two or more words separated by spaces.
- If the input is not a string type then return an error and ask for valid input.

Expected Output:

- Return the longest word in the sentence.
- "**Error: Please provide a valid sentence.**" if the input does not contain valid sentence.

6. Count how many numbers are greater than the average.

Create a function that identifies all numbers in an array that are greater than the average value of the array.

The function should calculate the average using only numeric values, ignore non-numeric data, and return all unique numbers that are greater than the calculated average. The original array should remain unchanged.

Accepted Inputs and Validation Logics:

- An array containing any type of data.
- Only numeric values should be considered for calculations. Non-numeric values must be ignored.

Expected Output:

- Return a string in the following format containing no duplicates: "**The numbers those are greater than the average are: X, Y, Z.....**"
- If no numbers are greater than the average, return: "**There are no numbers greater than the average.**"

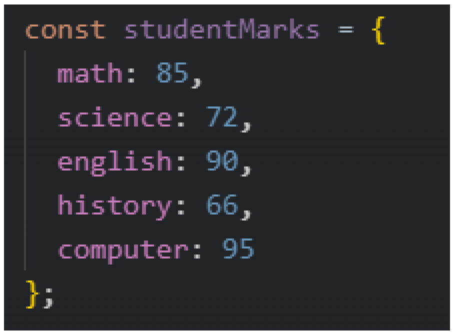
*(Here **X, Y, Z** are the numbers those are greater than the average)*

7. Grading System Based on Subject Marks.

The goal is to convert the marks of all subjects stored in an object into their corresponding grades. Each subject's numeric score should be replaced with a grade based on the predefined grading rules.

The grading process must be done by **directly working with the object below**, without creating or using any function. The logic should iterate through the object's keys and assign grades according to the value of each subject.

Sample Object:



```
const studentMarks = {
  math: 85,
  science: 72,
  english: 90,
  history: 66,
  computer: 95
};
```

Grading Rules:

A+ → 80 – 100	**A** → 70 – 79	**A-** → 60 – 69	**B** → 50 – 59
C → 40 – 49	**D** → 33 – 39	**F** → Below 33	

Workflow:

- Apply the grading rules to the given sample object to replace marks with grades.

Expected Output:

- Return a new object where The keys remain the same (subject names) and the values are replaced with their corresponding grades.

8. Capitalize the First Letter of Each Word

Create a function that converts a sentence so that the **first character of every word is uppercase**.

Accepted Inputs and Validation Logics:

- A sentence containing two or more words separated by spaces.
- If the input is not a string type then return an error and ask for valid input.

Expected Output:

- Return a new string where the first character of each word is uppercase.
- "**Error: Please provide a valid sentence.**" if the input does not contain valid sentence.

9. Decode a Morse Code Message.   (Simple But Not Simple)

A message is provided in **Morse code format**, consisting of dots (.), dashes (-), spaces, and word separators.

Your task is to **decode the message into a meaningful English text** using JavaScript. The decoding process should correctly interpret individual characters and word boundaries.

Suggestion: The decoding rules for Morse code are **publicly available**. Standard online references may be used to understand how Morse code maps to English characters before implementing the decoding logic.

Decoding Rules:

- Each Morse sequence represents a single English character.
- Characters within a word are separated by a **single space**.
- Words are separated by a **forward slash (/)**.
- The decoded output should preserve correct spacing between words.

Expected Output:

- Return the decoded English message as a string.

Example:

> Input: **.... . .-.. .-.. --- / .-- --- .-. .-.. -..**
> Output: **HELLO WORLD**

10. Convert Length Units.

Create a function that converts a length value from one unit to another.

Accepted Units:

mm (millimeter)	**cm** (centimeter)	**m** (meter)	**km** (kilometer)
mile (mile)	**ft** (foot)	**in** (inch)	

Accepted Inputs and Validation Logics:

- The function must accept **three inputs**:
    a. **Unit value** – the numeric value to be converted
    b. **Current unit type** – the unit of the given value
    c. **Target unit type** – the unit to convert the value into
- The function should correctly convert the given value from the current unit to the target unit.
- The returned value should be a numeric result of the conversion.
- The conversion logic must handle all valid unit combinations.
- If the unit value is **not a number**, throw an error clearly indicating that the **unit value is invalid**.
- If the current unit type is not one of the accepted unit names, throw an error stating that the **current unit is invalid or not found**. The error message must also **suggest the list of valid unit names**.
- If the conversion (target) unit type is not one of the accepted unit names, throw an error stating that the **conversion unit is invalid or not found**. The error message must also **suggest the list of valid unit names**.

Expected Output:

- If all inputs are valid, return the converted numeric value.
- If any validation fails, throw an appropriate error as defined above.