Table of Contents

## 0.1 FunctionDef add

# 1 Function: add(a, b)

## 1.1 Overview

The add function computes the sum of two numbers.

## 1.2 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| a | number | The first number to be added. |
| b | number | The second number to be added. |

## 1.3 Description

This function takes two arguments, a and b, and returns their sum. It uses the standard addition operator (+) to perform the calculation. The function is a straightforward implementation of arithmetic addition for numeric inputs.

// The function returns the result of a + b  
return a + b;

## 1.4 Usage Notes

* This function is designed to work with numeric types (integers or floating-point numbers).
* If strings are passed as arguments, the + operator will perform string concatenation instead of numeric addition due to JavaScript’s type coercion. For example, add("2", "3") would result in "23".
* To ensure proper addition, make sure both inputs are numbers before calling the function.

**Output Example**: The function returns a single number which is the sum of the two input parameters.

## 1.5 Example

// Example usage  
let result = add(5, 10);  
console.log(result);  
  
let anotherResult = add(-7.5, 2.5);  
console.log(anotherResult);

**Output:**

15  
-5

## 1.6 FunctionDef factorial

# 2 Function: factorial(n)

## 2.1 Overview

The factorial function recursively calculates the factorial of a given non-negative integer.

## 2.2 parameters

* n (Number): The non-negative integer whose factorial is to be calculated.

## 2.3 Description

This function implements the factorial operation using recursion. The logic is divided into two main parts: a base case and a recursive step.

The function first evaluates the base case using a ternary operator: n <= 1 ? 1 : .... If the input number n is less than or equal to 1, the function immediately returns 1. This is the termination condition for the recursion, as the factorial of 1 (1!) and 0 (0!) are both defined as 1.

If n is greater than 1, the function executes the recursive step: ... : n \* factorial(n - 1). It returns the product of the current number n and the result of calling itself with the argument n - 1. This process repeats, decrementing n by 1 in each subsequent call, until the base case (n <= 1) is reached.

For example, factorial(4) would be calculated as follows: 1. factorial(4) returns 4 \* factorial(3) 2. factorial(3) returns 3 \* factorial(2) 3. factorial(2) returns 2 \* factorial(1) 4. factorial(1) returns 1 (base case)

The results are then multiplied up the call stack: 4 \* 3 \* 2 \* 1, yielding 24.

## 2.4 Usage Notes

* This function is designed for non-negative integers. Providing a negative number will cause infinite recursion, leading to a “Maximum call stack size exceeded” error.
* Due to its recursive nature, calculating the factorial of very large numbers can also lead to a stack overflow error. For such cases, an iterative approach is recommended.
* The function does not validate if the input is an integer. While it may produce a result for floating-point numbers, the factorial is mathematically defined only for non-negative integers.

**Output Example**: The function returns a single Number representing the calculated factorial.

## 2.5 Example

// Example usage  
const number = 5;  
const result = factorial(number);  
console.log(`The factorial of ${number} is ${result}`);

**Output:**

The factorial of 5 is 120

## 2.6 FunctionDef greet

# 3 Function: greet

## 3.1 Overview

The greet function logs a personalized greeting message to the web console.

## 3.2 parameters

* name (string): The name of the person to greet. This value will be embedded in the output message.

## 3.3 Description

This function provides a simple way to display a standard greeting. It accepts a single argument, name. The core logic uses the console.log method to print a message to the browser’s developer console or a Node.js terminal.

The message is constructed using a JavaScript template literal (a string enclosed in backticks `). This allows for easy embedding of variables directly into the string using the ${name} syntax. When the function is called, the value passed as the name parameter replaces the ${name} placeholder, creating a personalized message like “Hello, Alice!”.

## 3.4 Usage Notes

* This function does not return any value; its sole purpose is to produce a side effect by logging output to the console.
* While the name parameter is intended to be a string, JavaScript will attempt to convert any passed data type to its string representation. For example, passing the number 123 will result in the output “Hello, 123!”.

## 3.5 Example

// Example usage  
greet("World");  
greet("Alice");

**Output:**

Hello, World!  
Hello, Alice!