Table of Contents

## 0.1 FunctionDef add(a, b)

# 1 Function: add(a, b)

## 1.1 Overview

The add function calculates 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

The add function takes two parameters, a and b. It uses the standard addition operator (+) to compute the sum of these two values. The result of this operation is then returned by the function.

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

## 1.4 Usage Notes

* This function is designed for numeric inputs. If strings are provided, JavaScript’s + operator will perform string concatenation instead of addition (e.g., add(5, "10") will return "510").
* The function works correctly with both integer and floating-point numbers.

**Output Example**: A numeric value representing the sum of the inputs. For example, if a is 5 and b is 10, the output will be 15.

## 1.5 Example

// Example usage  
const num1 = 5;  
const num2 = 10;  
const result = add(num1, num2);  
  
console.log(result);

**Output:**

15

## 1.6 FunctionDef factorial(n)

# 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 for which to calculate the factorial.

## 2.3 Description

This function implements the factorial operation, which is the product of all positive integers up to a given number n (denoted as n!). It uses a recursive approach to achieve this.

The core logic is based on a conditional (ternary) operator: n <= 1 ? 1 : n \* factorial(n - 1).

1. **Base Case**: The recursion terminates when the input n is less than or equal to 1. According to the definition of factorials, the factorial of 1 (1!) is 1, and the factorial of 0 (0!) is also 1. The condition n <= 1 correctly handles both of these base cases by returning 1.
2. **Recursive Step**: If n is greater than 1, the function returns the product of n and the result of calling itself with the argument n - 1. This process unwinds as follows for an input like 4:
   * factorial(4) returns 4 \* factorial(3)
   * factorial(3) returns 3 \* factorial(2)
   * factorial(2) returns 2 \* factorial(1)
   * factorial(1) hits the base case and returns 1.

The results are then multiplied back up the call stack: 2 \* 1 = 2, then 3 \* 2 = 6, and finally 4 \* 6 = 24.

## 2.4 Usage Notes

* This function is intended for non-negative integers. Providing a negative number will cause infinite recursion, leading to a “Maximum call stack size exceeded” error.
* For very large values of n, this recursive implementation may exceed the JavaScript engine’s call stack limit. An iterative approach is recommended for calculating factorials of very large numbers to avoid stack overflow errors.
* The function does not perform type or value checking on the input n. Passing non-integer values may lead to unexpected behavior.

**Output Example**: The function returns a single Number representing the calculated factorial. For factorial(5), the output would be:

120

## 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(name)

# 3 Function: greet(name)

## 3.1 Overview

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

## 3.2 parameters

* name (String): The name to include in the greeting message.

## 3.3 Description

This function provides a simple way to display a standardized greeting to the console. It takes a single parameter, name, which is expected to be a string.

The core logic uses the console.log method to print a formatted string. The string is created using a template literal (enclosed in backticks `), which allows for the direct embedding of variables. The value of the name parameter is interpolated into the string Hello, ${name}! to generate the final output message.

## 3.4 Usage Notes

* This function does not return a value (undefined). Its primary purpose is to produce a side effect by printing output to the console.
* While the function is designed to work with a string name, JavaScript’s type coercion will convert non-string arguments into their string representation. For example, calling greet(123) will output “Hello, 123!”.

## 3.5 Example

// Example usage:  
greet("World");  
greet("Developer");

**Output:**

Hello, World!  
Hello, Developer!