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 given arguments.

## 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 provides a straightforward way to perform addition. It accepts two parameters, a and b. Internally, it uses the + operator to compute the sum of these two values. The resulting sum is then returned by the function.

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

## 1.4 Usage Notes

* The + operator in JavaScript is used for both numeric addition and string concatenation. If either a or b is a string, the function will perform string concatenation instead of addition. For example, add(5, "5") would result in the string "55".
* To ensure correct mathematical addition, always provide numeric inputs for both parameters.

**Output Example**: A numeric value representing the sum.

## 1.5 Example

// Example usage with two numbers  
let result = add(10, 5);  
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 the factorial will be computed.

## 2.3 Description

This function calculates the factorial of a number n, which is the product of all positive integers up to n (n!). It employs a recursive approach to achieve this.

The function’s logic is based on two main parts:

1. **Base Case**: The function first checks if the input n is less than or equal to 1. The factorial of 1 (1!) is 1, and the factorial of 0 (0!) is also defined as 1. This condition serves as the termination point for the recursion, preventing an infinite loop. When this condition is met, the function returns 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 breaks the problem down into smaller, identical subproblems until the base case is reached.

For example, calculating factorial(4) would unfold as follows:

factorial(4) = 4 \* factorial(3)  
 = 4 \* (3 \* factorial(2))  
 = 4 \* (3 \* (2 \* factorial(1)))  
 = 4 \* (3 \* (2 \* 1))  
 = 24

## 2.4 Usage Notes

* The 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 the recursive implementation, very large values for n can also exhaust the call stack and cause a stack overflow error.
* Factorial values grow very rapidly. For inputs larger than a certain threshold (e.g., 170), the result may exceed JavaScript’s standard number representation and return Infinity.

**Output Example**: A typical return value for a valid input like 5 would be the number 120.

## 2.5 Example

// Example usage  
let result = factorial(5);  
console.log(result);

**Output:**

120

## 2.6 FunctionDef greet(name)

# 3 Function: greet

## 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 mechanism for displaying a standardized greeting. It accepts a single argument, name. The core logic utilizes the console.log method to print a formatted string to the standard output, such as a web browser’s developer console or a Node.js terminal.

The output string is constructed using a template literal: `Hello, ${name}!`. The placeholder ${name} within this string is dynamically substituted with the value passed to the name parameter upon function execution. This results in a personalized message for each call.

## 3.4 Usage Notes

* This function does not return a value (undefined). Its primary purpose is to produce a side effect by logging output to the console.
* While the name parameter is intended to be a string, JavaScript’s type coercion will attempt to convert any passed argument into its string representation. For example, calling greet(123) will output “Hello, 123!”.

## 3.5 Example

// Example usage  
greet('Alice');  
greet('Developer');

**Output:**

Hello, Alice!  
Hello, Developer!