Table of Contents

## 0.1 FunctionDef add(a, b)

# 1 Function: add(a, b)

## 1.1 Overview

The add function computes the sum of two given 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 provides a basic arithmetic operation. It takes two parameters, a and b, which are expected to be numbers. The function uses the standard addition operator (+) to calculate the sum of these two values. The result of this operation is then returned by the function.

For example, if a is 10 and b is 5, the function will compute 10 + 5 and return the value 15.

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

## 1.4 Usage Notes

* This function is designed for numerical addition. If string values are passed as arguments, the + operator will perform string concatenation instead of addition. For example, add("Hello, ", "World!") would return "Hello, World!".
* To ensure correct arithmetic, always pass arguments of the Number type. Passing mixed types (e.g., a number and a string) will result in type coercion and string concatenation.

**Output Example**: A typical return value for numerical inputs.

15

## 1.5 Example

// Example usage with two numbers  
let result = add(7, 8);  
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 calculated.

## 2.3 Description

This function computes the factorial of a number n using a recursive approach. The factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n.

The function’s logic is based on a ternary operator which serves as a compact if-else statement:

1. **Base Case**: It first checks if the input n is less than or equal to 1. If it is, the function returns 1. This is the termination condition for the recursion, as the factorial of 1 (1!) and 0 (0!) is defined as 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.

For example, factorial(4) would be calculated as follows: 4 \* factorial(3) 4 \* (3 \* factorial(2)) 4 \* (3 \* (2 \* factorial(1))) 4 \* (3 \* (2 \* 1)) Which evaluates to 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.
* Be aware of JavaScript’s number limitations. Calculating the factorial of very large numbers can result in a stack overflow due to deep recursion or return Infinity if the result exceeds the maximum value for a standard number.
* The base cases factorial(0) and factorial(1) will both correctly return 1.

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

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

## 3.1 Overview

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

## 3.2 parameters

* name (string): The name to be included in the greeting message.

## 3.3 Description

This function provides a simple way to display a standardized greeting. It accepts a single argument, name. The core logic uses the console.log() method to print a formatted string to the web console. The message is constructed using a template literal: `Hello, ${name}!`. The ${name} expression within the string is a placeholder that is dynamically replaced by the value passed into the name parameter, creating a personalized output.

## 3.4 Usage Notes

* This function does not return a value; its primary effect is the side effect of printing output to the console.
* While the name parameter is expected to be a string for the intended output, JavaScript’s type coercion will convert non-string arguments into their string representation when embedded in the template literal.

## 3.5 Example

// Example usage of the greet function  
greet("World");  
greet("Alice");

**Output:**

Hello, World!  
Hello, Alice!