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 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 simple mechanism for performing addition. It takes two parameters, a and b, which are expected to be numbers. The core logic of the function uses the standard addition operator (+) to calculate the sum of these two parameters. The result of this operation is then returned as the function’s output.

// The function returns the sum of its two arguments.  
return a + b;

## 1.4 Usage Notes

* This function is intended for numeric addition. If non-numeric types (like strings) are passed, JavaScript’s + operator will perform concatenation instead of mathematical addition, which may lead to unexpected behavior.
* The function correctly handles both integer and floating-point numbers.

**Output Example**: A successful call to add(10, 5) will return the number 15.

## 1.5 Example

// Example usage  
const num1 = 15;  
const num2 = 7;  
const result = add(num1, num2);  
  
console.log(result);

**Output:**

22

## 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 implements the mathematical concept of a factorial, denoted as n!, which is the product of all positive integers up to n. The implementation uses recursion to achieve this.

The core logic is contained within a single conditional (ternary) expression: n <= 1 ? 1 : n \* factorial(n - 1).

1. **Base Case**: The recursion terminates when the input n is less than or equal to 1. By definition, the factorial of 1 (1!) is 1, and the factorial of 0 (0!) is also 1. The function returns 1 in these cases, which provides the essential stopping condition for the recursive calls.
2. **Recursive Step**: If n is greater than 1, the function multiplies n by the result of calling itself with the argument n - 1. This creates a chain of calls, each with a decremented number, until the base case is reached.

For instance, calculating factorial(4) unfolds as follows:

factorial(4) // returns 4 \* factorial(3)  
factorial(3) // returns 3 \* factorial(2)  
factorial(2) // returns 2 \* factorial(1)  
factorial(1) // returns 1 (base case)

The final result is the product of the values from each step: 4 \* 3 \* 2 \* 1 = 24.

## 2.4 Usage Notes

* This function is recursive. Providing a very large number as input can lead to a “Maximum call stack size exceeded” error, as each recursive call consumes memory on the call stack.
* The function is intended for non-negative integers. Passing a negative number will cause an infinite recursion, ultimately resulting in a stack overflow.
* While the function will execute with floating-point numbers, the factorial concept is traditionally defined only for non-negative integers.

**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(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 be included in the greeting message.

## 3.3 Description

The greet function provides a simple way to display a formatted greeting. It accepts a single string argument, name. The function’s core logic uses console.log() to print a message to the standard output, typically the browser’s developer console or a terminal environment.

The message itself is constructed using a JavaScript template literal: `Hello, ${name}!`. This syntax allows the value of the name parameter to be directly embedded within the string, creating a personalized greeting.

// The function takes a 'name' and logs a greeting.  
function greet(name) {  
 // A template literal is used to embed the 'name' variable into the string.  
 console.log(`Hello, ${name}!`);  
}

## 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 expected to be a string, JavaScript’s type coercion will attempt to convert any non-string value passed to it into a string for the output. For instance, greet(123) will log “Hello, 123!”.

## 3.5 Example

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

**Output:**

Hello, World!  
Hello, Alice!