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 primary purpose is to encapsulate the addition operation for reuse and clarity.

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

## 1.4 Usage Notes

* This function is intended for numeric inputs. If strings are provided, the + operator will perform string concatenation instead of numeric addition. For example, add("2", "3") would result in "23".
* The function does not perform any type checking on its inputs. It is the caller’s responsibility to ensure that the arguments are numbers.

**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);

**Output:**

15

## 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 for which the factorial will be calculated.

## 2.3 Description

This function implements the factorial calculation using recursion. The logic is divided into two main parts:

1. **Base Case**: The function first checks if the input number 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. This process continues until the base case is reached.

For example, calling factorial(4) will execute as follows: - 4 \* factorial(3) - 4 \* (3 \* factorial(2)) - 4 \* (3 \* (2 \* factorial(1))) - 4 \* (3 \* (2 \* 1)) - The final result is 24.

// Ternary operator implementation  
return n <= 1 ? 1 : n \* factorial(n - 1);

## 2.4 Usage Notes

* This function is designed for non-negative integers. Providing a negative number will cause an infinite recursion, leading to a stack overflow error.
* Due to the nature of recursion, calculating the factorial of very large numbers can also lead to a stack overflow error, as each recursive call adds a new frame to the call stack.
* The input n should be an integer. While JavaScript might handle floating-point numbers, the mathematical concept of a factorial is typically defined only for non-negative integers.

**Output Example**: 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(name)

## 3.1 Overview

The greet function prints 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 personalized greeting. When called, it takes a single argument, name, which is expected to be a string.

The function utilizes a template literal (`) to construct a new string, embedding the provided name into the phrase “Hello, …!”. The resulting string, for example, Hello, World!, is then passed to the console.log() method, which outputs it to the standard console. The function does not return any value; its sole purpose is to produce this console output.

// The function uses a template literal for string formatting.  
console.log(`Hello, ${name}!`);

## 3.4 Usage Notes

* This function does not return a value. Its output is sent directly to the console and cannot be assigned to a variable.
* If a non-string value (like a number or an object) is passed as the name, JavaScript will attempt to convert it to its string representation, which may result in unexpected output.
* The function’s primary effect is logging to the console, making it useful for debugging or displaying information to the user in a development environment.

## 3.5 Example

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

**Output:**

Hello, Alice!  
Hello, World!