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 and returns the result.

## 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 basic arithmetic operation. It takes two parameters, a and b, which are intended to be numeric values. The core logic of the function uses the standard JavaScript addition operator (+) to sum the two input values. The result of this calculation is then returned by the function.

// The function returns the sum of a and b  
return a + b;

## 1.4 Usage Notes

* This function is designed for numeric addition. While the JavaScript + operator can also perform string concatenation, passing string arguments to this function will result in concatenation rather than mathematical addition (e.g., add("2", "3") will return "23").
* The function works correctly with both integer and floating-point numbers.

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

## 1.5 Example

// Example usage with two integers  
const result1 = add(5, 10);  
console.log(result1);  
  
// Example usage with floating-point numbers  
const result2 = add(7.5, 2.25);  
console.log(result2);

**Output:**

15  
9.75

### 1.5.1 FunctionDef add

# 2 Function: add(a, b)

## 2.1 Overview

The add function is a placeholder intended to calculate the sum of two numbers.

## 2.2 parameters

| Parameter | Type | Description |
| --- | --- | --- |
| a | Number | The first number to be added. |
| b | Number | The second number to be added. |

## 2.3 Description

This function is designed to accept two numerical arguments, a and b. The intended purpose is to compute their sum and return the result.

However, the current implementation is an empty stub. It does not contain any operational logic to perform the addition. Consequently, when called, it does not perform any calculation and implicitly returns undefined.

## 2.4 Usage Notes

* This function is a placeholder and is not yet implemented.
* To use this function as intended, you must add the addition logic (e.g., return a + b;) to the function body.
* Calling the function in its current state will always result in undefined.

## 2.5 Example

The following example demonstrates the actual behavior of the current stub and the intended behavior of a fully implemented function.

**Actual Behavior (Current Stub):**

// Calling the empty function as provided  
const result = add(5, 10);  
console.log(result);

**Output:**

undefined

**Intended Behavior (Example Implementation):**

// To make the function work, it must be implemented like this:  
function add(a, b) {  
 return a + b;  
}  
  
const result = add(5, 10);  
console.log(result);

**Output:**

15

## 2.6 FunctionDef factorial(n)

# 3 Function: factorial(n)

## 3.1 Overview

The factorial function recursively calculates the factorial of a given non-negative integer.

## 3.2 parameters

* n (Number): The non-negative integer for which the factorial will be calculated.

## 3.3 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 conditional statement:

1. **Base Case**: The recursion terminates when the input n is less than or equal to 1. In this scenario, the function returns 1. This handles two fundamental cases: the factorial of 1 (1!) is 1, and by mathematical convention, the factorial of 0 (0!) is also 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 continuously breaks down the calculation into smaller sub-problems until the base case is reached.

For instance, a call to factorial(4) is resolved as follows:

factorial(4) // returns 4 \* factorial(3)  
 // which is 4 \* (3 \* factorial(2))  
 // which is 4 \* (3 \* (2 \* factorial(1)))  
 // which is 4 \* (3 \* (2 \* 1))  
 // resulting in 24

## 3.4 Usage Notes

* This function is intended for non-negative integers. Passing a negative number as an argument will lead to infinite recursion, ultimately causing a “Maximum call stack size exceeded” error.
* Be mindful of JavaScript’s limitations with deep recursion. Calculating the factorial of very large numbers can exhaust the call stack. For such cases, an iterative approach is generally safer and more memory-efficient.
* The function correctly handles factorial(0) and factorial(1), both of which return 1.

**Output Example**: A single number representing the calculated factorial value.

## 3.5 Example

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

**Output:**

120

### 3.5.1 FunctionDef factorial

# 4 Function: factorial(n)

## 4.1 Overview

The factorial function calculates the factorial of a given non-negative integer.

## 4.2 parameters

* n: The non-negative integer for which to calculate the factorial.

## 4.3 Description

This function computes the factorial of a number n, which is the product of all positive integers up to n. The factorial of n is denoted as n!. By mathematical convention, the factorial of 0 is 1.

The function is implemented using recursion. It has two main parts:

1. **Base Case**: The recursion terminates when the input n is 0 or 1. In this case, the function returns 1, as 0! = 1 and 1! = 1. This prevents an infinite loop of recursive calls.
2. **Recursive Step**: If n is greater than 1, the function calls itself with the argument n - 1 and multiplies the result by the current n. This process continues until the base case is reached.

For example, calculating factorial(4) would unfold as follows: factorial(4) = 4 \* factorial(3) factorial(3) = 3 \* factorial(2) factorial(2) = 2 \* factorial(1) factorial(1) = 1 (base case)

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

A common implementation would look like this:

function factorial(n) {  
 // Base case: if n is 0 or 1, factorial is 1.  
 if (n === 0 || n === 1) {  
 return 1;  
 }  
 // Recursive step: n! = n \* (n-1)!  
 return n \* factorial(n - 1);  
}

## 4.4 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.
* Factorial values grow very rapidly. Using large numbers for n can result in values that exceed JavaScript’s Number.MAX\_SAFE\_INTEGER, which may lead to a loss of precision or return Infinity.
* The input n should be an integer. Passing a non-integer value may produce unexpected results.

## 4.5 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

## 4.6 FunctionDef greet(name)

# 5 Function: greet

## 5.1 Overview

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

## 5.2 parameters

* name (string): The name of the person or entity to be greeted. This value will be embedded in the output message.

## 5.3 Description

This function provides a simple way to display a standard greeting. It accepts a single argument, name. The core logic uses the console.log() method to print a message to the standard output, which is typically the web browser’s developer console or the terminal in a Node.js environment.

The message is constructed using a JavaScript template literal: `Hello, ${name}!`. This allows the value of the name parameter to be directly interpolated into the string, creating a personalized message like “Hello, Alice!”. The function does not return any value; its sole purpose is to produce this console output.

## 5.4 Usage Notes

* This function does not return a value (it implicitly returns undefined). Its primary effect is 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-in value to its string representation. For example, passing the number 123 will result in the output “Hello, 123!”.

## 5.5 Example

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

**Output:**

The following lines will be printed to the console:

Hello, Alice!  
Hello, World!

### 5.5.1 FunctionDef greet

# 6 Function: greet

## 6.1 Overview

The greet function is a placeholder intended to perform a greeting action using the provided name.

## 6.2 parameters

* name (any): The name of the person or entity to be greeted.

## 6.3 Description

The greet function is defined to accept a single parameter, name. The intended purpose is to use this name to construct and display or return a greeting message.

However, in its current state, the function body is empty. Consequently, when greet is called, it executes no operations and does not explicitly return a value. In JavaScript, a function that does not have a return statement implicitly returns undefined.

function greet(name) {  
 // The function body is empty.  
 // No action is performed with the 'name' parameter.  
 // The function will implicitly return 'undefined'.  
}

## 6.4 Usage Notes

* This function currently serves as a stub or placeholder and lacks any implementation.
* To make this function useful, you must add logic to its body. For example, you could use console.log to print a greeting to the console or use a return statement to send back a greeting string.

## 6.5 Example

// Example usage  
// Calling the function will execute but produce no visible output.  
greet("World");  
  
// To see the return value, you can log it to the console.  
const returnValue = greet("Alice");  
console.log(returnValue);

**Output:**

undefined