## Functions

In JavaScript, functions are blocks of reusable code that can be defined and invoked to perform a specific task or calculate a value. Functions are one of the fundamental building blocks of JavaScript programming, and they play a crucial role in organizing and structuring code, promoting code reuse, and encapsulating logic.

**Let's learn functions in a simple way:**

Imagine you're baking cookies. You have a recipe that tells you exactly what ingredients you need and what steps to follow. Now, think of a function as a mini-recipe within the bigger recipe.

**Here's how it works:**

**Ingredients (Parameters):**

Just like in a recipe, you have specific ingredients like flour, sugar, and eggs. In a function, you have parameters, which are like placeholders for values that you'll use inside the function.

**Instructions (Code):**

In your mini-recipe (function), you have a set of instructions to follow. These instructions tell you what to do with the ingredients. Similarly, in a function, you have a block of code that performs a specific task.

**Usage (Invocation):**

Once you have your mini-recipe (function) prepared, you can use it whenever you need it. You can call the function, just like you'd follow the steps in your mini-recipe to make cookies.

**Output (Return Value):**

After following the instructions in your mini-recipe, you get something in return, like delicious cookies. Similarly, a function can also give you something back after performing its task. This is called the return value.

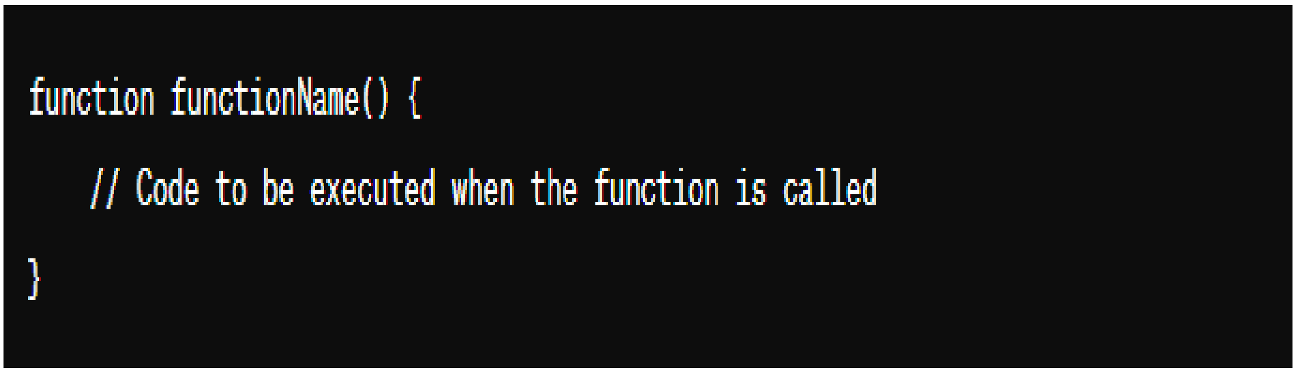
**Here's a simple analogy:**

Imagine you have a function called "MakeCookies". You give it ingredients like flour, sugar, and eggs (parameters). Inside the "MakeCookies" function, there are instructions on how to mix the ingredients and bake them to make delicious cookies (code). When you want to make cookies, you simply call the "MakeCookies" function and pass it the ingredients you have. Then, the function executes its instructions and gives you back the freshly baked cookies (return value).

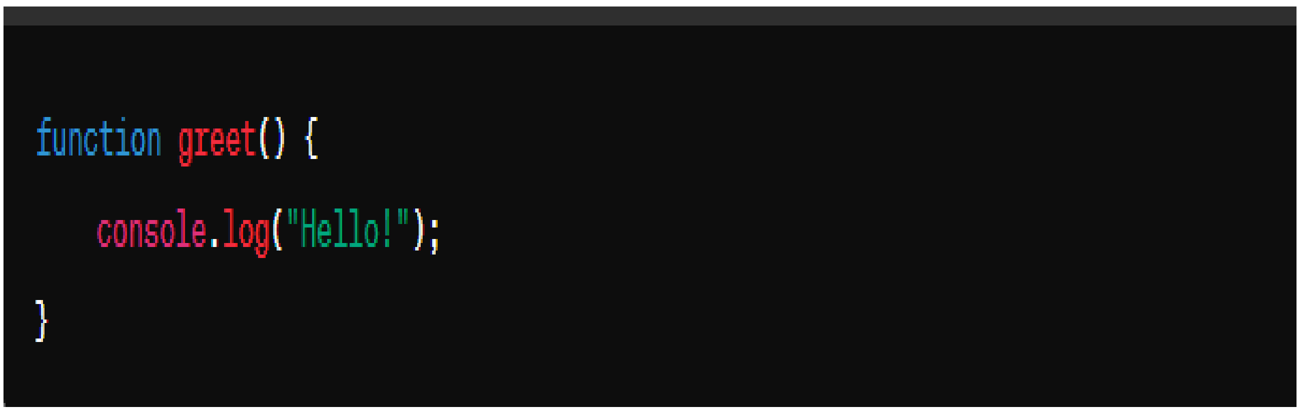
In summary, a function is like a mini-recipe that takes some inputs, performs a specific task, and gives you an output. It helps keep your code organized, reusable, and easy to understand, just like a recipe helps you bake delicious treats

**Function Declaration:**

You can define a function using the function keyword followed by the function name, parameters (optional), and the function body enclosed in curly braces {}.



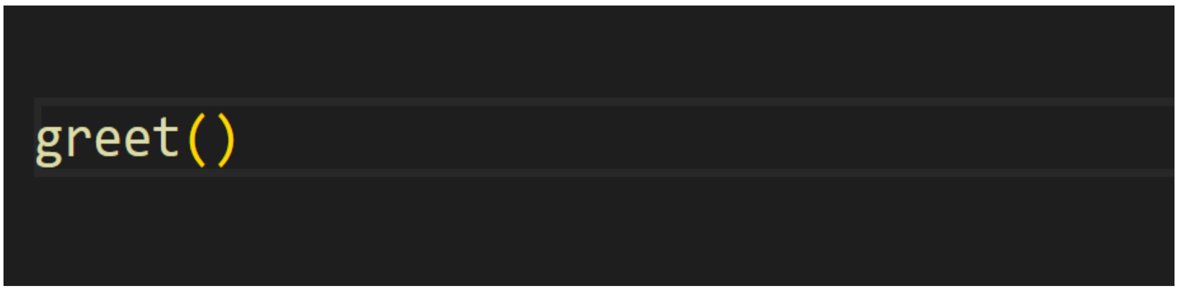
```
function functionName() {
    // Code to be executed when the function is called
}
```



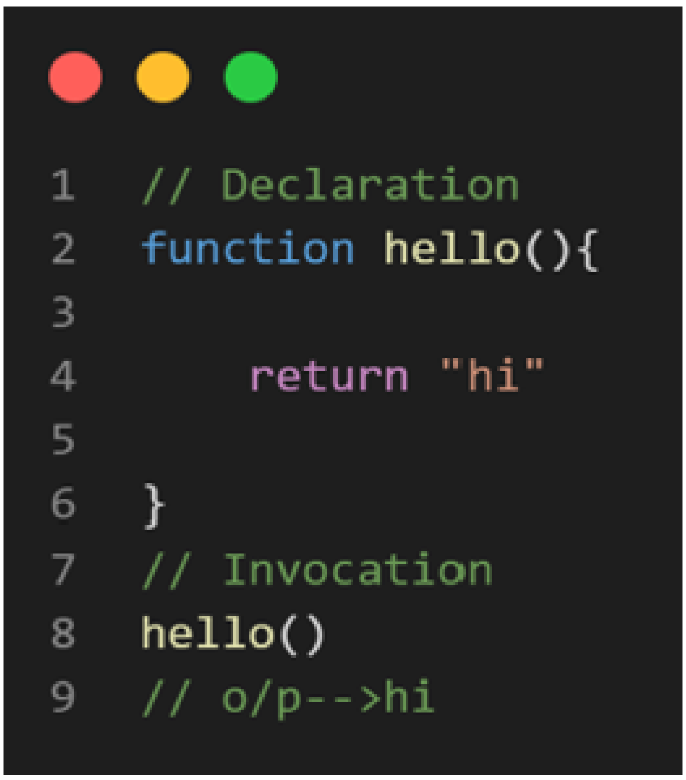
```javascript
function greet() {
    console.log("Hello!");
}
```

**Function Invocation:**

To execute or invoke a function, you simply use its name followed by parentheses (), optionally passing any required arguments.



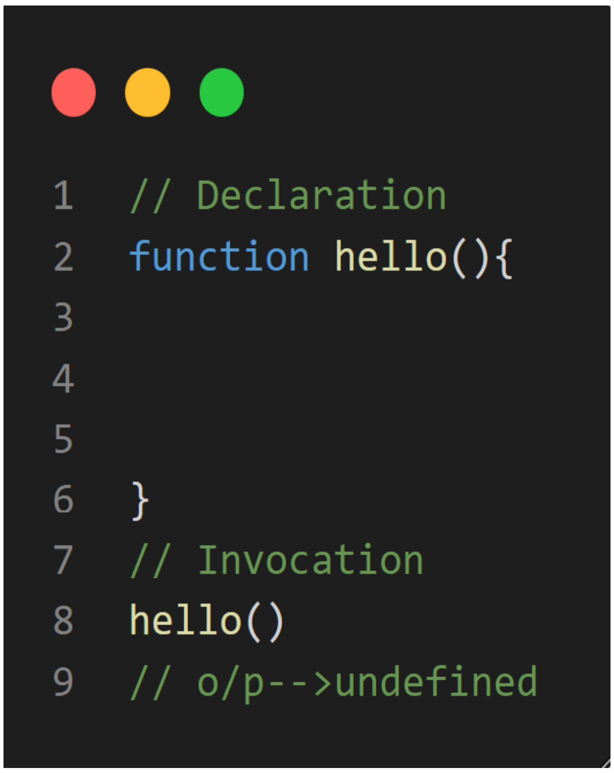
```javascript
greet()
```



```javascript
// Declaration
function hello(){

    return "hi"

}
// Invocation
hello()
// o/p-->hi
```



```javascript
// Declaration
function hello(){



}
// Invocation
hello()
// o/p-->undefined
```

In JavaScript, if a function doesn't explicitly return a value, it implicitly returns undefined. This means that when the function is called and executed, but no explicit return statement is encountered, the function returns undefined by default.

**"parameter" and "argument"**

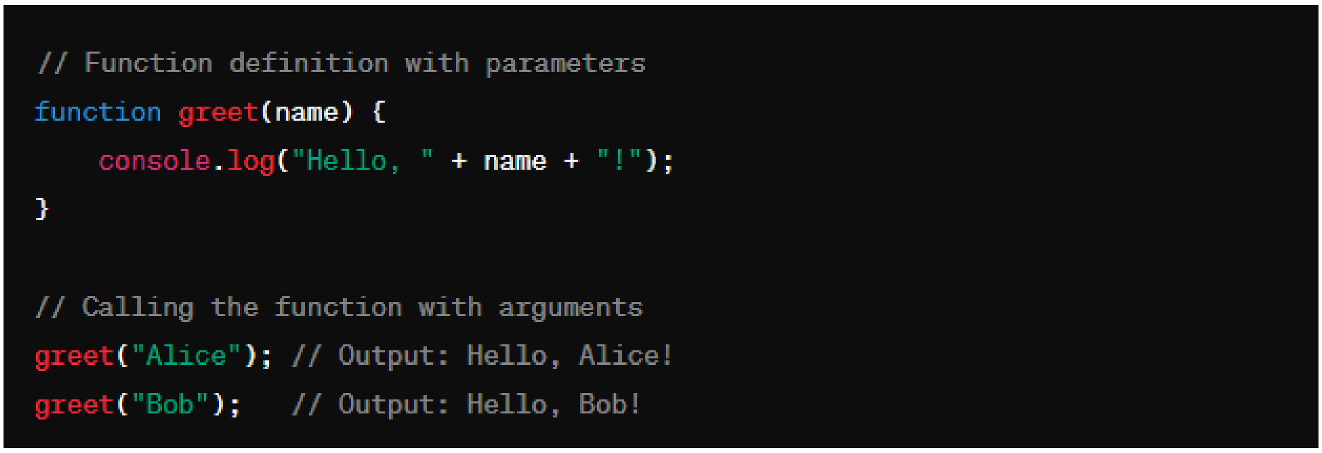
**1. Parameter:**

- In the context of functions, a parameter is a variable name listed in the function declaration. It represents a value that the function expects to receive when it is called.
- Parameters are essentially placeholders for values that will be provided to the function when it is invoked.
- Parameters are defined in the function declaration and act as local variables within the function body.
- Parameters define the function's interface, specifying what kind of input the function can accept.

**2. Argument:**

- An argument is a value that is passed to a function when the function is called or invoked.
- Arguments are the actual values supplied to the function during its execution.
- When a function is called, the arguments are passed to the corresponding parameters in the function definition.
- Arguments can be literals, variables, expressions, or even other functions.

In summary, parameters are variables declared in the function declaration, representing values that the function expects to receive, while arguments are the actual values passed to the function when it is called, matching the parameters specified in the function declaration.



```javascript
// Function definition with parameters
function greet(name) {
    console.log("Hello, " + name + "!");
}

// Calling the function with arguments
greet("Alice"); // Output: Hello, Alice!
greet("Bob");   // Output: Hello, Bob!
```

Here we define a function named greet that takes one parameter, name.

Inside the function, we use the parameter name to construct a greeting message.

When we call the greet function, we pass actual values, known as arguments, for the name parameter.

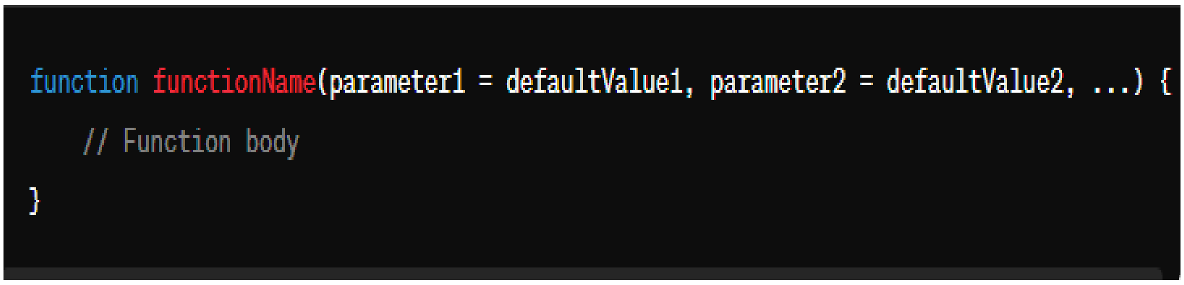
The function is called twice with different arguments: "Alice" and "Bob".

Each time the function is called, it receives the corresponding argument and logs a greeting message to the console.

Here, name is a parameter in the function definition, and "Alice" and "Bob" are the arguments passed to the function when it is called. The function uses these arguments to perform its task, which, in this case, is to greet the provided name.

**default parameters:**

Default parameters allow you to define parameters with default values that will be used if no argument is provided for that parameter when the function is called. This feature was introduced in ECMAScript 6 (ES6).

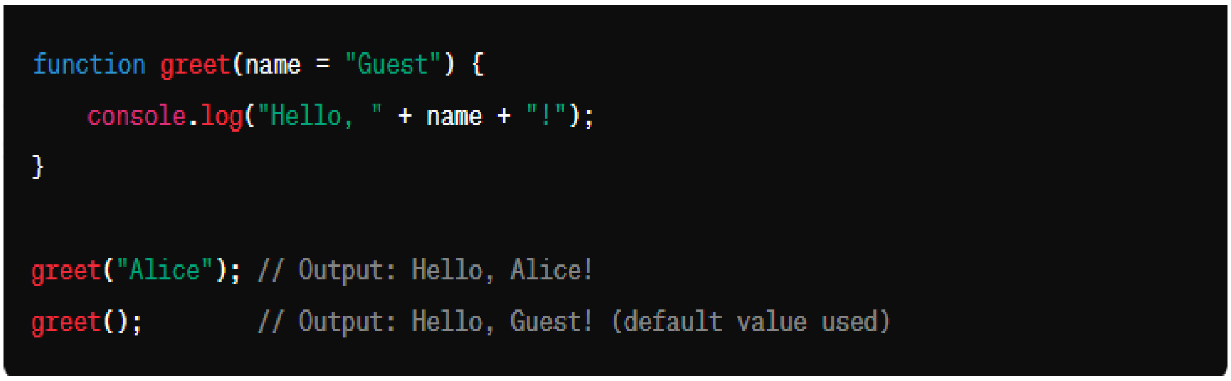


```
function functionName(parameter1 = defaultValue1, parameter2 = defaultValue2, ...) {
    // Function body
}
```

In this syntax:

- `parameter1`, `parameter2`, etc.: Parameters of the function.

- `defaultValue1`, `defaultValue2`, etc.: Default values assigned to the parameters if no argument is provided when the function is called.



```
function greet(name = "Guest") {
    console.log("Hello, " + name + "!");
}

greet("Alice"); // Output: Hello, Alice!
greet();        // Output: Hello, Guest! (default value used)
```

In this example, the `greet` function has a single parameter `name` with a default value of `"Guest"`. When the function is called with an argument (`greet("Alice")`), it uses the provided argument (`"Alice"`) as the `name`. However, when the function is called without any argument (`greet()`), it uses the default value `"Guest"` for the `name` parameter.

## Functions as first-class citizens

In JavaScript, functions are considered first-class citizens, which means they can be treated as values and used in the same way as other types of values, such as strings, numbers, or objects. This characteristic of functions as first-class citizens enables several powerful programming paradigms, including functional programming and higher-order functions. Here's what it means for functions to be first-class citizens in JavaScript

**Functions Can Be Assigned to Variables:**

You can assign a function to a variable, just like you would assign a string or a number.

```
const greet = function(name) {
    console.log("Hello, " + name + "!");
};
```

**Functions Can Be Passed as Arguments to Other Functions:**

You can pass a function as an argument to another function.

```
function callFunction(func) {
    func();
}

callFunction(function() {
    console.log("This is a function passed as an argument.");
});
```

**Functions Can Be Returned as Values from Other Functions**:

You can return a function from another function.

```
function createGreetingFunction() {
    return function(name) {
        console.log("Hello, " + name + "!");
    };
}

const greet = createGreetingFunction();
greet("John");
```

**Functions Can Be Stored in Data Structures:**

You can store functions in arrays, objects, or other data structures.

```
const functionsArray = [
    function() { console.log("Function 1"); },
    function() { console.log("Function 2"); }
];

functionsArray[0](); // Output: Function 1
```

**Functions Can Have Properties and Methods:**

Functions are objects in JavaScript and can have properties and methods like any other object.

```
function greet(name) {
    console.log("Hello, " + name + "!");
}

greet.customMessage = "Welcome!";
console.log(greet.customMessage); // Output: Welcome!
```

**Types of functions**

In JavaScript, there are several types of functions, each serving different purposes and offering unique features. Here are some common types of functions:

### 1. Named Function Declaration:

- This is the most common type of function declaration, where you specify a name for the function.

```
function greet(name) {
    console.log("Hello, " + name + "!");
}
```

### 2. Function Expression:

- A function expression is when you define a function as part of an expression. It can be anonymous or named.

```
const greet = function(name) {
    console.log("Hello, " + name + "!");
};
```

### 3. Arrow Function:

- Arrow functions provide a concise syntax for defining functions, especially for short, single-expression functions.

```
const greet = (name) => {
    console.log("Hello, " + name + "!");
};
```

### 4. Immediately Invoked Function Expression (IIFE):

- An IIFE is a function that is executed immediately after it's defined. It's often used to create a separate scope and prevent polluting the global namespace.

```
(function() {
    console.log("This is an IIFE");
})();
```

**5. Generator Function:**

- Generator functions allow you to define functions that can pause execution and yield intermediate results using the `yield` keyword.

```
function* generateSequence() {
    yield 1;
    yield 2;
    yield 3;
}
```

**6. Constructor Function:**

- Constructor functions are used to create objects with a specific structure and behavior. They are often used with the `new` keyword.

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}
```

**7. Higher-Order Function:**

- A higher-order function is a function that either takes another function as an argument or returns a function.

```
function map(array, callback) {
    let result = [];
    for (let item of array) {
        result.push(callback(item));
    }
    return result;
}
```