# JavaScript

## Functions

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

In JavaScript, functions are one of the core building blocks and are essential to structuring code. They allow you to organize your code into reusable pieces, making it modular, readable, and efficient. Here's a breakdown of JavaScript functions and their primary purposes:

### 1. Defining Functions

- Functions are defined with the `function` keyword, followed by a name and a set of parameters. For example:

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

### 2. Function Expressions

- Functions can also be assigned to variables, known as function expressions. This approach doesn't require a name, making it an "anonymous function."

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

### 3. Arrow Functions

- A shorter syntax for functions introduced in ES6, arrow functions are especially useful for concise code.

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

### 4. Function Parameters and Arguments

- Functions can accept parameters (inputs) and arguments (values passed when calling the function).

```
function add(a, b) {
  return a + b;
}
console.log(add(3, 5)); // Output: 8
```

## 5. Return Statements

- Functions can return a value using the `return` statement, allowing the function's output to be stored or used elsewhere in the code.

```
function multiply(a, b) {
  return a * b;
}
let result = multiply(4, 5); // result is 20
```

## 6. Anonymous Functions

- Functions without a name, useful in callbacks or situations where the function is only used once.

```
setTimeout(function() {
  console.log("This runs after 2 seconds");
}, 2000);
```

## 7. IIFE (Immediately Invoked Function Expression)

- A function that runs as soon as it's defined. IIFEs are useful for creating a private scope.

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

## 8. Higher-Order Functions

- Functions that take other functions as parameters or return functions as results. Higher-order functions are essential for callbacks, event handling, and functional programming.

```
function applyOperation(a, b, operation) {
  return operation(a, b);
}
console.log(applyOperation(3, 4, add)); // Using the add function
```

## 9. Callback Functions

- A function passed into another function to be executed later. Commonly used in asynchronous programming.

```
function fetchData(callback) {
  setTimeout(() => {
    callback("Data received");
  }, 1000);
}

fetchData((message) => {
  console.log(message); // Output after 1 second: "Data received"
```

```
});
```

## 10. Recursion

- A function that calls itself, useful for tasks like traversing trees or complex data structures.

```
function factorial(n) {
  if (n <= 1) return 1;
  return n * factorial(n - 1);
}
console.log(factorial(5)); // Output: 120
```

## Summary

Functions in JavaScript help modularize code, improve reusability, and enable asynchronous operations, making them crucial for efficient programming.