Functions are reusable blocks of code:

- Declared using `function` keyword: `function myFunc() {}` .
- Arrow functions: `const myFunc = () => {}` .
- Parameters and return values.
- **Higher-order functions**: Functions that take other functions as arguments.
- **Callback functions**: Functions passed as parameters to other functions.

Function Statement / Function Declaration

```
function greet(name) {
    return `Hello, ${name}!`;
}
console.log(greet('Alice'));  // Output: Hello, Alice!
```

Function Expression - Anonymous Function Assigned to a Variable

```
const greet = function(name) {
    return `Hello, ${name}!`;
};
console.log(greet('Bob'));
```

Anonymous Function

An anonymous function is a function without a name.
It is often used as a value for a variable or passed as an argument to another function.

```
function(parameters) {
    // function body
}
```

First-Class Functions

In JavaScript, functions are first-class citizens, meaning they can:

- Be assigned to variables

- Be passed as arguments to other functions
- Be returned from functions
- Be assigned to object properties

```
// Assigning a function to a variable
const greet = function(name) {
    return `Hello, ${name}!`;
};

// Passing a function as an argument
function executeFunction(func, value) {
    return func(value);
}
console.log(executeFunction(greet, 'Dave'));

// Returning a function from another function
function createGreeting(greeting) {
    return function(name) {
        return `${greeting}, ${name}!`;
    };
}
const sayHi = createGreeting('Hi');
console.log(sayHi('Eve'));  // Output: Hi, Eve!
```

Arrow Functions

Arrow functions provide a shorter syntax for writing function expressions and lexically bind the `this` value.

```
const functionName = (parameters) => {
    // function body
};
```

IIFE (Immediately Invoked Function Expression)

Variables declared inside an IIFE are not accessible outside its scope.

```
(function () {
    const privateVar = "I am private";
    console.log(privateVar);
})();
// console.log(privateVar); // Error: privateVar is not defined
```

Callback Function

A function passed as an argument to another function and executed later. Enables asynchronous programming, used in events, timers, and handling results of operations like API requests.

```
// Define a function that takes a callback
function processUserInput(callback) {
    const name = prompt('Please enter your name.');
    callback(name);
}

// Define a callback function
function greet(name) {
    alert(`Hello, ${name}!`);
}

// Pass the callback function as an argument
processUserInput(greet);
```

Higher-Order Function

A higher-order function is a function that can take other functions as arguments and/or return a function as a result.

```
function addition(callback) {
    console.log("Adding two numbers");
    callback();
}

function add() {
    console.log(2 + 2);
}
addition(add);  // Higher-order function taking a function as an argument

// Returning a function
function addition() {
    return function add() {
        console.log(2 + 2);
    };
}
const sum = addition(); // Higher-order function returning another function
sum();
```

Summary:

- **Function Declaration vs. Function Expression**: Function declarations are hoisted, while function expressions are not.
- **First-Class Functions**: Functions can be assigned to variables, passed as arguments, and returned from other functions.
- **Arrow Functions**: Shorter syntax, lexically bind `this`.
- **IIFE**: Self-executing function useful for avoiding global scope pollution.
- **Callback Functions**: Used in asynchronous programming and event handling.
- **Higher-Order Functions**: Functions that take or return other functions, enabling functional programming.