## **Types of Functions in JavaScript**

JavaScript provides different ways to declare and use functions. Below are the main types, along with explanations, code examples, and interview questions.

### **1. Function Declaration (Named Function)**

A function declaration defines a function using the function keyword and a name. It is hoisted, meaning it can be called before its definition.

function greet(name) {

return `Hello, ${name}!`;

}

console.log(greet("Alice")); // Output: Hello, Alice!

#### **Interview Question**

✅ *What is function hoisting? How does it affect function declarations?***Answer:** Function declarations are hoisted, meaning they can be called before they are defined.

#### **Hoisting Example**

console.log(sum(5, 3)); // Works fine due to hoisting

function sum(a, b) {

return a + b;

}

**2. Function Expression (Anonymous Function)**

A function expression assigns a function to a variable. Unlike function declarations, function expressions are not hoisted.

const greet = function(name) {

return `Hello, ${name}!`;

};

console.log(greet("Bob")); // Output: Hello, Bob!

#### **Interview Question**

✅ *What is the difference between a function declaration and a function expression?***Answer:** Function declarations are hoisted, while function expressions are not.

#### **Hoisting Example**

console.log(add(2, 3)); // Error: Cannot access 'add' before initialization

const add = function(a, b) {

return a + b;

};

### **3. Arrow Function**

A shorter syntax introduced in ES6. Arrow functions do not have their own this binding.

const multiply = (a, b) => a \* b;

console.log(multiply(4, 2)); // Output: 8

#### **Interview Question**

✅ *How do arrow functions differ from regular functions?***Answer:** Arrow functions do not have their own this, and they cannot be used as constructors.

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

A function that runs immediately after being defined. Often used to create private scopes.

(function() {

console.log("IIFE executed!");

})();

#### **Interview Question**

✅ *Why use an IIFE?***Answer:** To create a private scope and avoid polluting the global namespace.

### **5. Higher-Order Function**

A function that takes another function as an argument or returns a function.

function operate(num, func) {

return func(num);

}

const double = (n) => n \* 2;

console.log(operate(5, double)); // Output: 10

#### **Interview Question**

✅ *What is a higher-order function? Give an example.***Answer:** A function that takes another function as an argument or returns a function.

## **Closures in JavaScript**

A closure is a function that "remembers" the variables from its parent scope even after the parent function has finished executing.

function outerFunction(outerVar) {

return function innerFunction(innerVar) {

return `Outer: ${outerVar}, Inner: ${innerVar}`;

};

}

const closureFunc = outerFunction("Hello");

console.log(closureFunc("World")); // Output: Outer: Hello, Inner: World

#### **Interview Question**

✅ *What is a closure in JavaScript?***Answer:** A function that retains access to its parent scope even after the parent function has executed.

## **Hoisting in JavaScript**

Hoisting is JavaScript's behavior of moving variable and function declarations to the top of their scope before code execution.

| **Variable Type** | **Hoisted?** | **Initialized?** | **Accessible Before Declaration?** |
| --- | --- | --- | --- |
| var | ✅ Yes | ❌ No | ✅ Undefined |
| let | ✅ Yes | ❌ No | ❌ ReferenceError |
| const | ✅ Yes | ❌ No | ❌ ReferenceError |
| Function Declaration | ✅ Yes | ✅ Yes | ✅ Works |
| Function Expression | ❌ No | ❌ No | ❌ ReferenceError |

console.log(x); // Undefined (var is hoisted but not initialized)

var x = 10;

console.log(y); // ReferenceError (let is hoisted but not initialized)

let y = 20;

## 

## 

## 

## 

## 

## 

## **Summary Table**

| **Feature** | **Function Declaration** | **Function Expression** | **Arrow Function** | **IIFE** | **Higher-Order Function** |
| --- | --- | --- | --- | --- | --- |
| **Hoisted?** | ✅ Yes | ❌ No | ❌ No | ❌ No | ❌ No |
| **Has this Binding?** | ✅ Yes | ✅ Yes | ❌ No | ✅ Yes | ✅ Yes |
| **Can Be Used as Constructor?** | ✅ Yes | ✅ Yes | ❌ No | ❌ No | ✅ Yes |
| **Common Use Cases** | General functions | Callback functions | Short functions | Encapsulation | Functional programming |

#### **Closure & Hoisting with var, let, and const**

| **Feature** | **var** | **let** | **const** |
| --- | --- | --- | --- |
| **Hoisted?** | ✅ Yes | ✅ Yes | ✅ Yes |
| **Initialized?** | ❌ No (undefined) | ❌ No | ❌ No |
| **Can Be Reassigned?** | ✅ Yes | ✅ Yes | ❌ No |
| **Block Scoped?** | ❌ No | ✅ Yes | ✅ Yes |