DAY-6

JavaScript Functions and Scope

JavaScript Functions and Scope:

1. ***Functions in JavaScript:***
   – A function in JavaScript is a block of code designed to perform a particular task. Functions are reusable, so instead of repeating the same code multiple times, you can define it once in a function and call it whenever needed.

   Syntax of a Function:

```
        function functionName(parameters) {
          // code to be executed
}
```

   - **functionName:** The name of the function.
   - **parameters:** Input values the function can accept (optional).
   - **code block:** The code that runs when the function is called.

   Example:

```
        // Function to greet someone by their name
        function greet(name) {
          console.log("Hello, " + name + "!");
        }

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

   In this example:
   - greet is the function name.
   - name is the parameter, which gets the value when the function is called.
   - The function prints a greeting message to the console.

2. ***Function with Return Value***
   – Functions can also return a value. This means you can perform a calculation or any operation inside the function, and then return the result.

   Syntax of a Function with Return Value:

```
function functionName(parameters) {
  // code to perform an operation
  return value;  // Return a value
}
```

DAY-6

Example:

```
// Function to calculate the square of a number
function square(num) {
  return num * num;  // Returns the square of the number
}

// Calling the function and storing the result
let result = square(5);  // result is now 25
console.log(result);  // Output: 25
```

In this example:

- square is a function that returns the square of a number.
- The return statement sends the result back to the caller.
- The result is stored in the variable result, which is then printed.

3. ***Function Expressions (Anonymous Functions)***
   - In JavaScript, functions can also be defined as expressions, meaning they can be assigned to variables. These are often called anonymous functions because they don't have a name.

Example:

```
// Function expression assigned to a variable
let add = function(a, b) {
  return a + b;
};

// Calling the function
let sum = add(10, 5);
console.log(sum);  // Output: 15
```

In this case:

- The function is defined and assigned to the variable add.
- The function takes two parameters (a and b) and returns their sum.

DAY-6

4. ***Arrow Functions (ES6)***
   – Arrow functions provide a shorter syntax for writing functions. They are especially useful for simple functions and function expressions.

Syntax of Arrow Functions:

```
const functionName = (parameters) => {
  // code to be executed
}
```

Example:

```
// Arrow function to add two numbers
const add = (a, b) => {
  return a + b;
};

let sum = add(10, 5);
console.log(sum);  // Output: 15
```

For functions with a single expression, you can omit the return and curly braces:

```
const multiply = (a, b) => a * b;  // Single expression without curly braces
let product = multiply(4, 3);
console.log(product);  // Output: 12
```

5. ***Scope in JavaScript***
   – Scope refers to the context in which variables and functions are accessible. There are different types of scopes in JavaScript:
     1. **Global Scope:**
        - Variables or functions declared outside of any function are in the global scope, meaning they can be accessed anywhere in the program.
     2. **Local Scope:**
        - Variables declared inside a function are in the local scope of that function and can only be accessed inside that function.
     3. **Block Scope (introduced in ES6 with let and const):**
        - Variables declared with let or const inside a block (e.g., if, for) are only accessible within that block.

DAY-6

Example of Global Scope:

```
let globalVariable = "I am global!";  // Declared globally

function showGlobal() {
  console.log(globalVariable);  // Can access global variable inside the function
}

showGlobal();  // Output: I am global!
```

In this case, the globalVariable is accessible both outside and inside the showGlobal function because it's in the global scope.

Example of Local Scope:

```
function localScopeExample() {
  let localVariable = "I am local!";  // Declared inside the function
  console.log(localVariable);  // Can access inside the function
}

localScopeExample();  // Output: I am local!

// Trying to access localVariable outside the function will result in an error
// console.log(localVariable);  // Error: localVariable is not defined
```

Here, localVariable is only accessible inside the localScopeExample function, which means it has local scope.

Example of Block Scope:

```
if (true) {
  let blockScopedVariable = "I am block-scoped!";
  console.log(blockScopedVariable);  // Output: I am block-scoped!
}

// Trying to access blockScopedVariable outside the block will result in an error
// console.log(blockScopedVariable);  // Error: blockScopedVariable is not defined
```

DAY-6

6. Function Scope Example:

```
function greet() {
  let greeting = "Hello, world!";  // Local variable in function scope
  console.log(greeting);  // Can access greeting inside the function
}

greet();  // Output: Hello, world!

// Trying to access greeting outside the function will result in an error
// console.log(greeting);  // Error: greeting is not defined
```

In this case, the greeting variable is accessible only inside the greet function because it's declared in function scope.

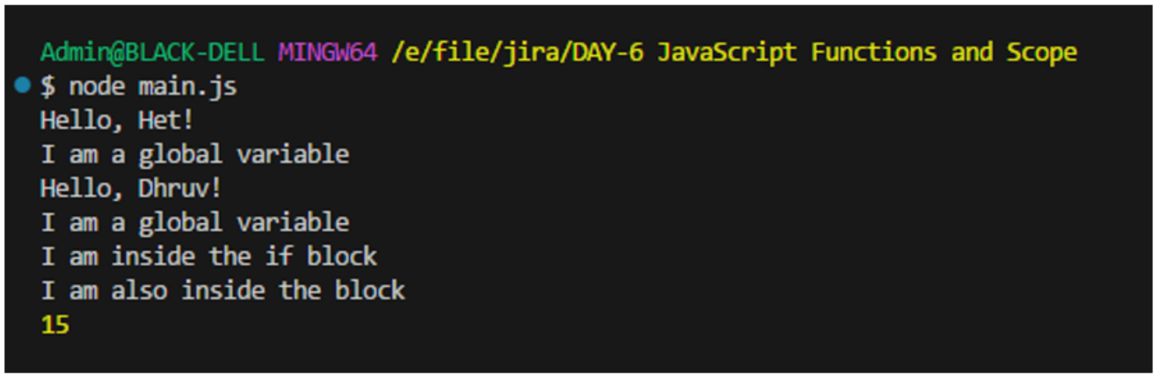
Example: Functions and Scope in JavaScript

```
// Global scope
let globalVar = "I am a global variable";  // Accessible anywhere in the
program

// Step 1: Function to greet a person
function greet(name) {
  // Local scope inside the function
  let greeting = "Hello, " + name + "!";  // This variable is only accessible
inside the greet function
  console.log(greeting);  // Prints the greeting message

  // Using global variable inside the function
  console.log(globalVar);  // Accessing global variable inside the function
}

// Call the greet function
greet("Het");  // Output: Hello, Het!
               // Output: I am a global variable

greet("Dhruv");  // Output: Hello, Dhruv!
                 // Output: I am a global variable

// Step 2: Trying to access function-scoped variable outside the function
```

DAY-6

```
// This will result in an error because "greeting" is scoped to the greet function
// console.log(greeting);  // Error: greeting is not defined

// Step 3: Block scope with let and const
if (true) {
  let blockScopedVar = "I am inside the if block";  // Block scope
  const blockScopedConst = "I am also inside the block";  // Block scope
  console.log(blockScopedVar);  // Output: I am inside the if block
  console.log(blockScopedConst);  // Output: I am also inside the block
}

// Trying to access the block-scoped variables outside the block will result in an error
// console.log(blockScopedVar);  // Error: blockScopedVar is not defined
// console.log(blockScopedConst);  // Error: blockScopedConst is not defined

// Step 4: Function with return value
function add(a, b) {
  return a + b;  // Return the sum of a and b
}

// Calling the add function and storing the result in a variable
let result = add(10, 5);
console.log(result);  // Output: 15
```

Output:



```
Admin@BLACK-DELL MINGW64 /e/file/jira/DAY-6 JavaScript Functions and Scope
$ node main.js
Hello, Het!
I am a global variable
Hello, Dhruv!
I am a global variable
I am inside the if block
I am also inside the block
15
```