# Day-6

## Task-1 : JavaScript Functions and Scope

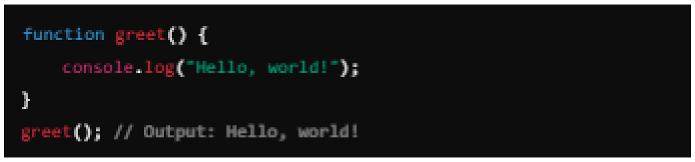
### *Functions in JavaScript*

A **function** is a block of code designed to perform a specific task. It can be executed whenever it's called or invoked.

### *Defining a Function*

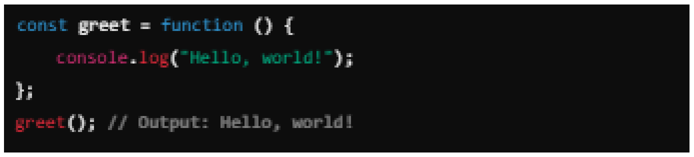
Functions in JavaScript can be defined in different ways:

1. **Function Declaration**



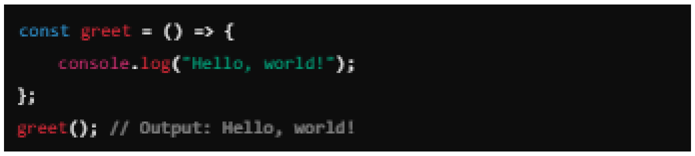
```
function greet() {
    console.log("Hello, world!");
}
greet(); // Output: Hello, world!
```

2. **Function Expression**



```
const greet = function () {
    console.log("Hello, world!");
};
greet(); // Output: Hello, world!
```

3. **Arrow Function (ES6)**



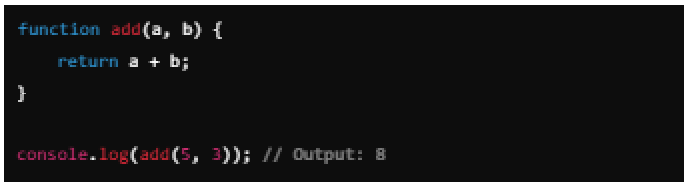
```
const greet = () => {
    console.log("Hello, world!");
};
greet(); // Output: Hello, world!
```

### *Parameters and Arguments*

Functions can take **parameters** as inputs and return outputs.

- **Parameters** are placeholders defined in the function.
- **Arguments** are actual values passed when the function is called.

Example:

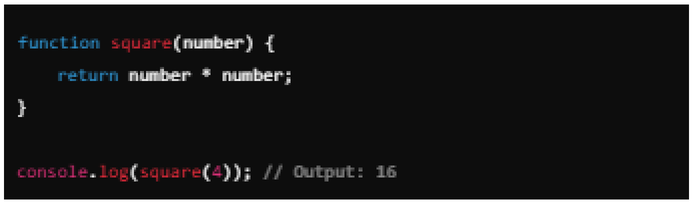


```
function add(a, b) {
    return a + b;
}

console.log(add(5, 3)); // Output: 8
```

### *Function Return Values*

A function can return a value using the `return` statement.

Example:

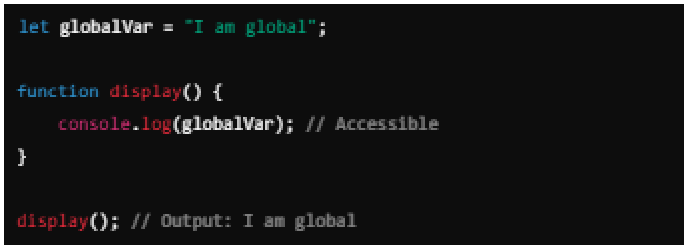


```
function square(number) {
    return number * number;
}

console.log(square(4)); // Output: 16
```

### *Scope in JavaScript*

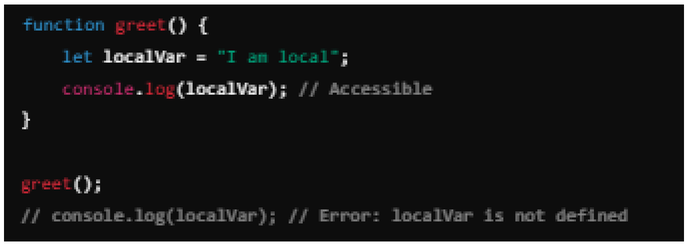
**Scope** determines the accessibility of variables in different parts of the code.

1. **Global Scope**
   - Variables declared outside of any function or block are in the global scope.
   - Accessible anywhere in the script.



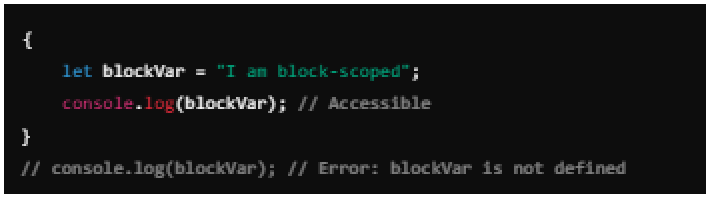
```
let globalVar = "I am global";

function display() {
    console.log(globalVar); // Accessible
}

display(); // Output: I am global
```

2. **Local Scope**
   - Variables declared inside a function are in local scope.
   - Accessible only within that function.



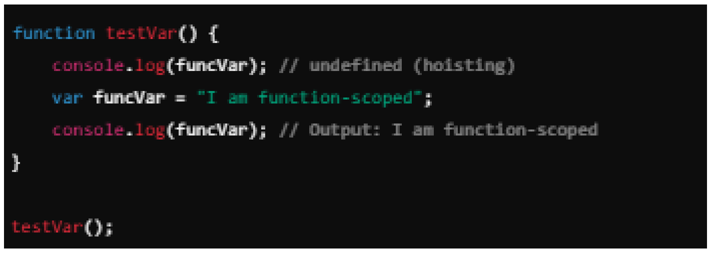
```
function greet() {
    let localVar = "I am local";
    console.log(localVar); // Accessible
}

greet();
// console.log(localVar); // Error: localVar is not defined
```

3. **Block Scope**
   - Variables declared with `let` or `const` inside a block `{}` are block-scoped.
   - Accessible only within that block.



```
{
    let blockVar = "I am block-scoped";
    console.log(blockVar); // Accessible
}
// console.log(blockVar); // Error: blockVar is not defined
```

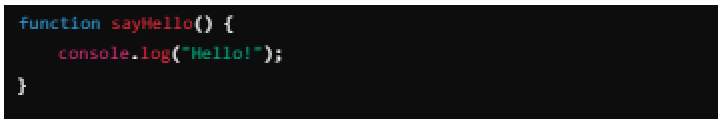
4. **Function Scope**
   - Variables declared with `var` are function-scoped.
   - Accessible throughout the entire function, even before declaration (due to **hoisting**).



```
function testVar() {
    console.log(funcVar); // undefined (hoisting)
    var funcVar = "I am function-scoped";
    console.log(funcVar); // Output: I am function-scoped
}

testVar();
```

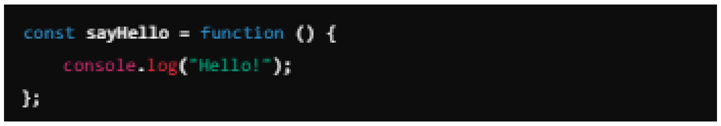
## *Types of Functions*

1. **Named Function**:



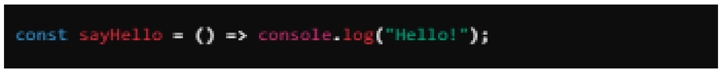
```
function sayHello() {
    console.log("Hello!");
}
```

2. **Anonymous Function**:



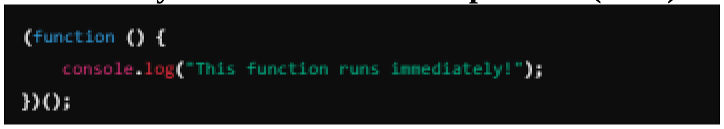
```
const sayHello = function () {
    console.log("Hello!");
};
```

3. **Arrow Function**:



```
const sayHello = () => console.log("Hello!");
```

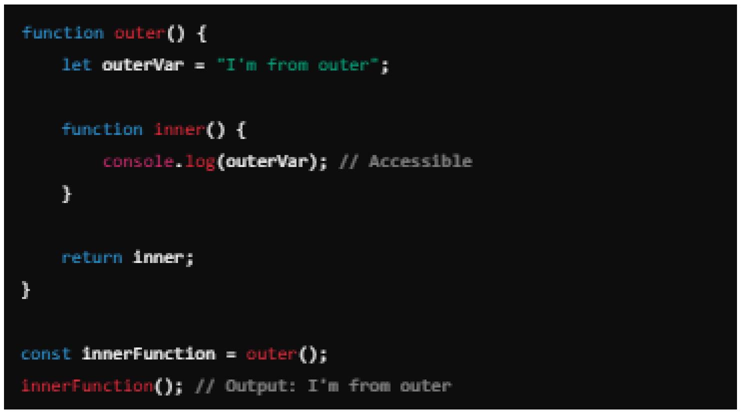
4. **Immediately Invoked Function Expression (IIFE)**:



```
(function () {
    console.log("This function runs immediately!");
})();
```

## Closures

A closure is a function that "remembers" its lexical scope even when executed outside that scope.



```
function outer() {
    let outerVar = "I'm from outer";

    function inner() {
        console.log(outerVar); // Accessible
    }

    return inner;
}

const innerFunction = outer();
innerFunction(); // Output: I'm from outer
```