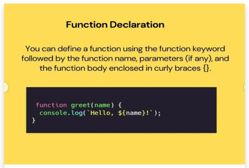
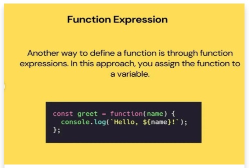


### Function Declaration

You can define a function using the function keyword followed by the function name, parameters (if any), and the function body enclosed in curly braces {}.

```
function greet(name) {
 console.log(`Hello, ${name}!`);
}
```

Function Declaration

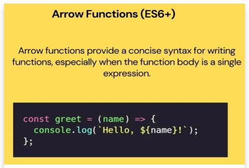


### Function Expression

Another way to define a function is through function expressions. In this approach, you assign the function to a variable.

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

Function Expression

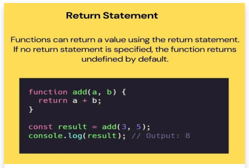


### Arrow Functions (ES6+)

Arrow functions provide a concise syntax for writing functions, especially when the function body is a single expression.

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

Arrow Function ES6+



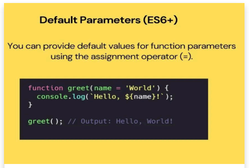
### Return Statement

Functions can return a value using the return statement. If no return statement is specified, the function returns undefined by default.

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

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

Function - Return



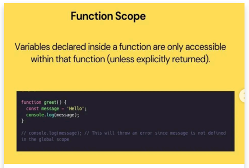
### Default Parameters (ES6+)

You can provide default values for function parameters using the assignment operator (=).

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

greet(); // Output: Hello, World!
```

Function - Default Parameter



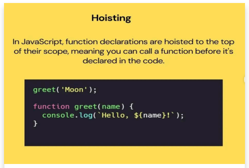
### Function Scope

Variables declared inside a function are only accessible within that function (unless explicitly returned).

```
function greet() {
  const message = 'Hello';
  console.log(message);
}

// console.log(message); // This will throw an error since message is not defined
in the global scope
```

Arrow Function ES6+



### Hoisting

In JavaScript, function declarations are hoisted to the top of their scope, meaning you can call a function before it's declared in the code.

```
greet('Moon');

function greet(name) {
  console.log(`Hello, ${name}!`);
}
```

Function - Return