وزارة التعليم العالي والبحث العلمي

جامعة برج العرب التكنولوجية
BORG AL ARAB TECHNOLOGICAL UNIVERSITY

Lecture Six:

JavaScript Functions and Control Structures

COURSE TITLE: WEB PROGRAMMING 1

TOPIC: JAVASCRIPT FUNCTIONS AND CONTROL STRUCTURES

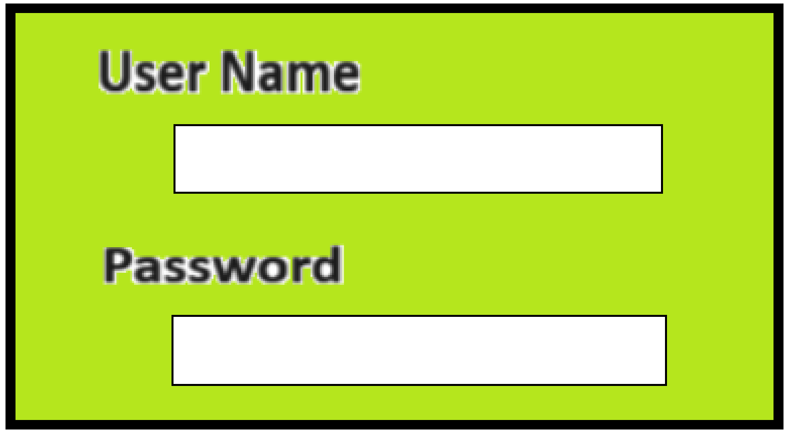
SEMESTER: 1 2025/2026 DURATION: 10 WEEKS

What is JavaScript?

- JavaScript as a high-level, interpreted programming language.
- JavaScript's role in web development to create interactive user experiences.
- Benefits of using JavaScript for client-side,

  reducing server load and enhancing

  user interactivity.



Variables and Data Types

- Introduction to variables as containers for storing data in JavaScript.
- How to declare variables using the var, let, and const keywords.
- Explanation of different data types: numbers, strings, booleans, arrays, and objects.
- Practical examples of variable declaration and data manipulation in JavaScript.

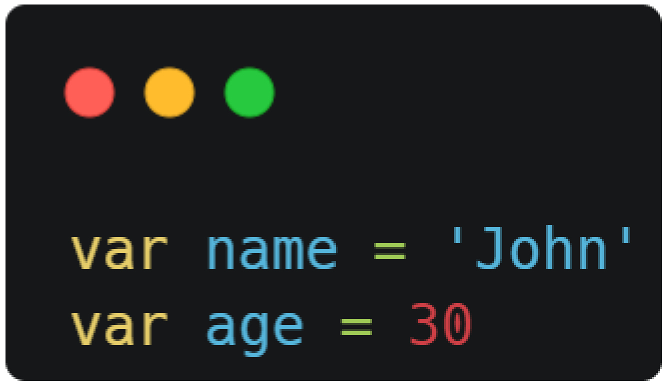
Variables and Data Types

You can declare variables using the var, let, and const keywords.

Each has its own characteristics and use cases.

1. var Keyword

Syntax:



```
var name = 'John'
var age = 30
```

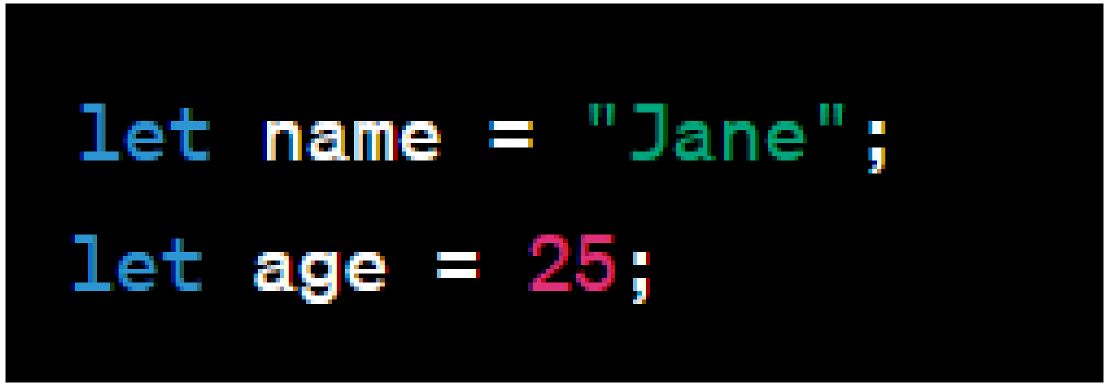
Variables and Data Types

2. let Keyword

The let keyword allows block-scoped variable declaration,

Especially in loops.

Syntax:



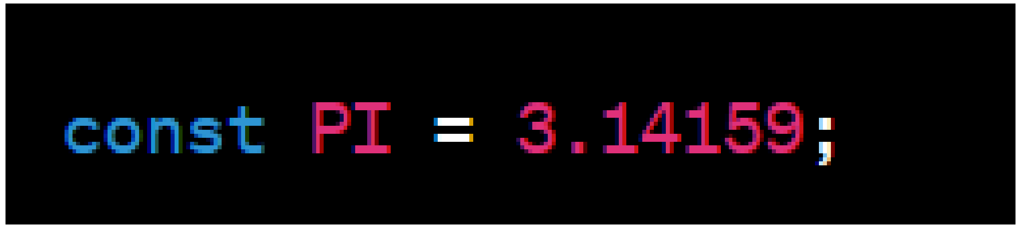
```
let name = "Jane";
let age = 25;
```

Variables and Data Types

3. const Keyword

The const keyword allows block-scoped declaration of variables that cannot be reassigned.

Syntax:



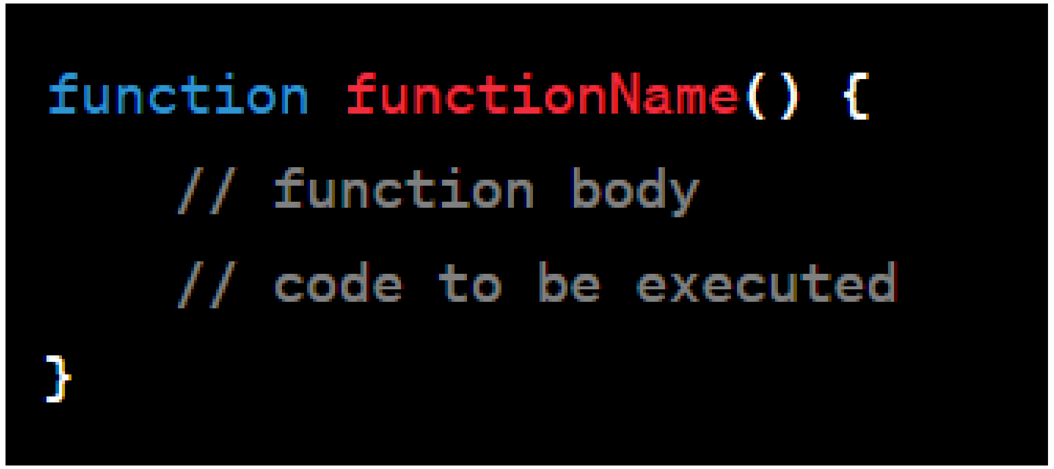
```
const PI = 3.14159;
```

Operators and Expressions

- Overview of arithmetic operators (+, -, *, /, %) and their use in mathematical calculations.
- Explanation of comparison operators (==, ===, !=, !==, >, <, >=, <=) for conditional expressions.
- Understanding logical operators (&&, ||, !) for combining multiple conditions.
- Practical examples of using operators and expressions to solve problems in JavaScript

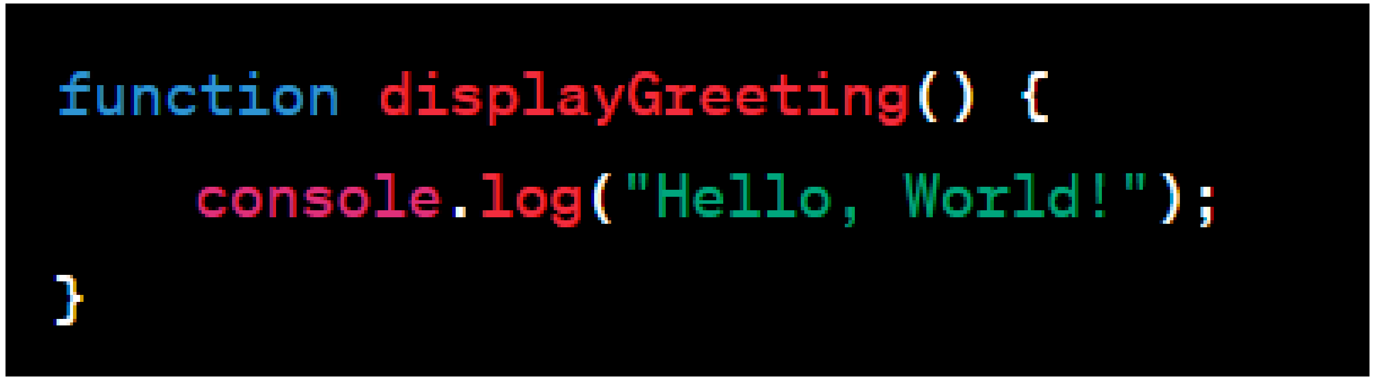
Introduction to Functions

- Functions is a block of code designed to perform a specific task.
- Calls functions to execute their code.



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

Example



```
function displayGreeting() {
    console.log("Hello, World!");
}
```

Function Parameters

- Passing parameters to functions.
- Exploring the use of arguments.
- Creating flexible functions with multiple parameters.

Basic Structure

```
function functionName(parameter1, parameter2, ...) {
    // function body
    // code to be executed
}
```

example

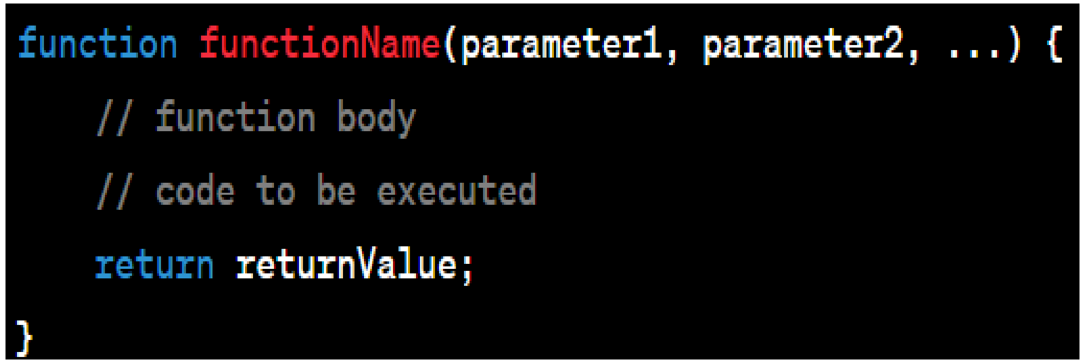
```
function greetUser(name, age) {
    console.log("Hello, " + name + "! You are " + age + " years old.");
}
```

Invoking

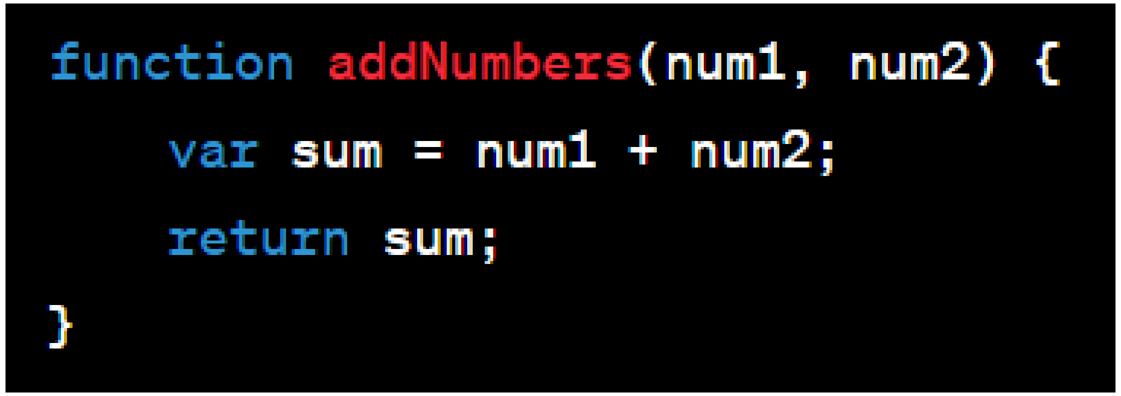
```
greetUser("Alice", 30);
greetUser("Bob", 25);
```

Return Values

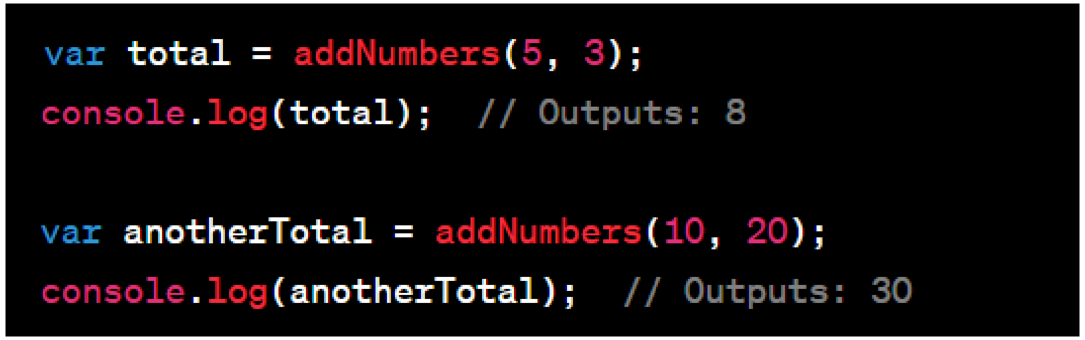
1. Understanding return statements in functions.
2. Using return values to obtain results from functions.
3. Examples of functions with return values.



```
function functionName(parameter1, parameter2, ...) {
    // function body
    // code to be executed
    return returnValue;
}
```



```
function addNumbers(num1, num2) {
    var sum = num1 + num2;
    return sum;
}
```



```
var total = addNumbers(5, 3);
console.log(total);  // Outputs: 8

var anotherTotal = addNumbers(10, 20);
console.log(anotherTotal);  // Outputs: 30
```

Higher-Order Functions

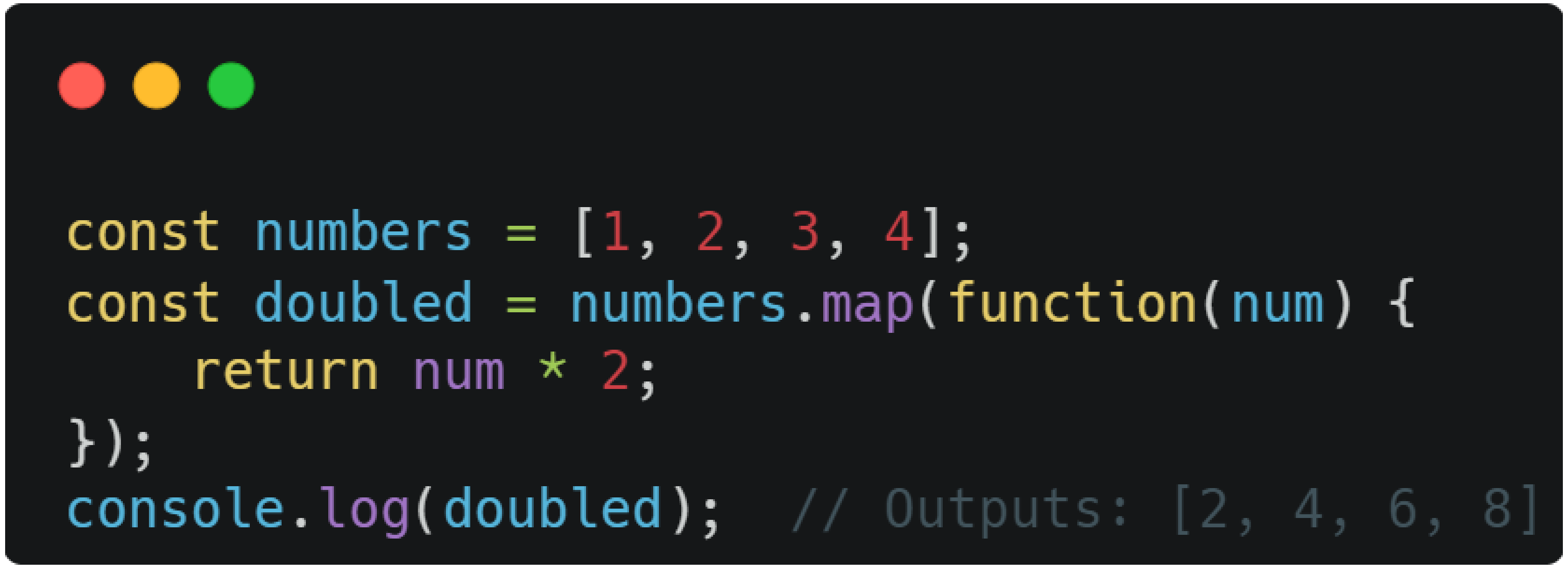
Definition of Higher-Order Functions:

A higher-order function is a function that does at least one of the following:

1. Takes one or more functions as arguments.
2. Returns a function as its result.

Examples and Common Uses:

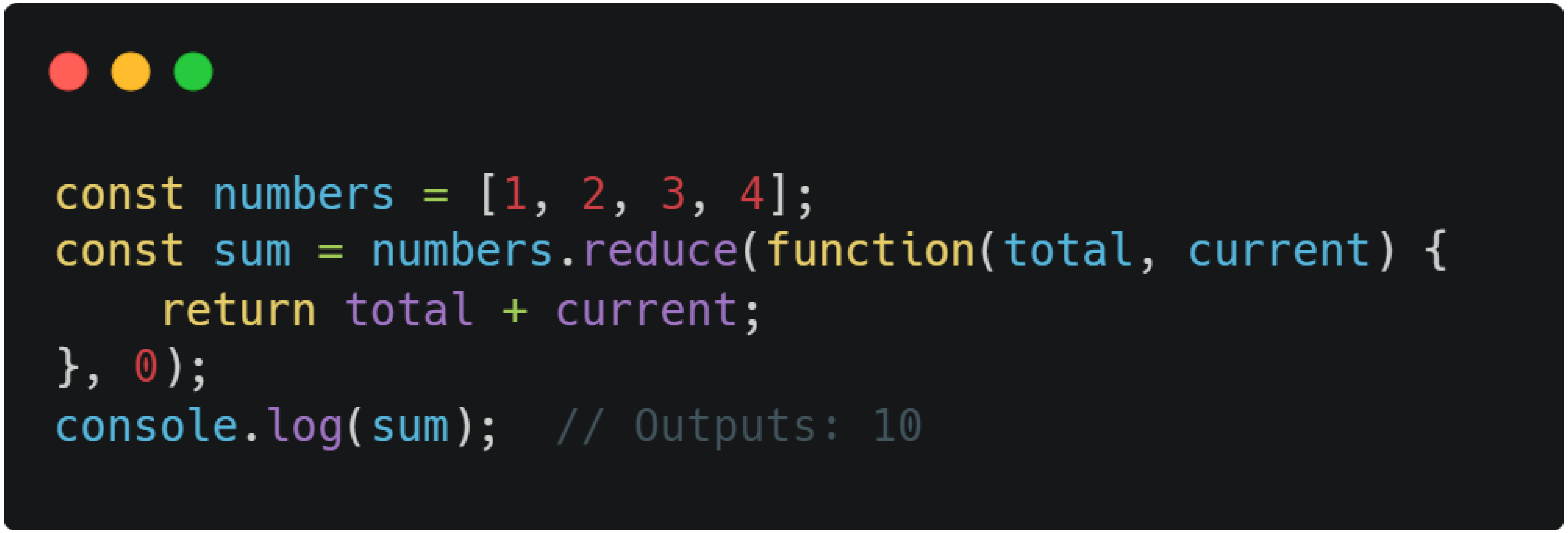
map: Transforms each element in an array using a function and returns a new array.



```
const numbers = [1, 2, 3, 4];
const doubled = numbers.map(function(num) {
    return num * 2;
});
console.log(doubled);  // Outputs: [2, 4, 6, 8]
```

Examples and Common Uses:

reduce: Accumulates values in an array based on a function, resulting in a single value.



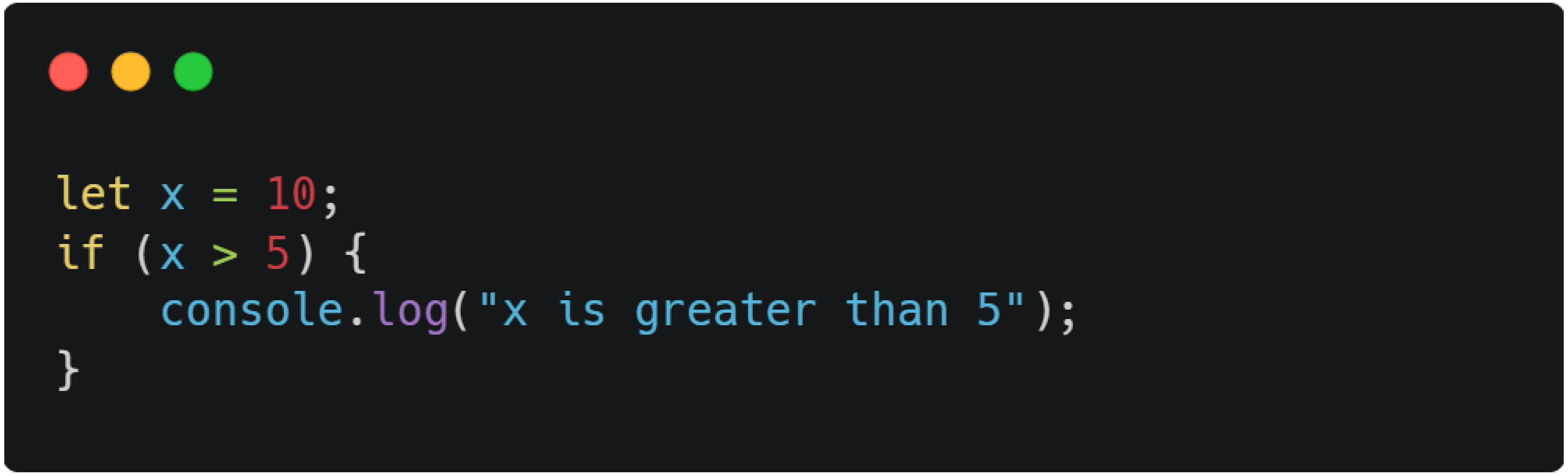
```
const numbers = [1, 2, 3, 4];
const sum = numbers.reduce(function(total, current) {
    return total + current;
}, 0);
console.log(sum);  // Outputs: 10
```

Introduction to Control Structures

- Understanding control structures in programming.
- Control structures as decision-making tools.
- Overview of conditionals and loops.

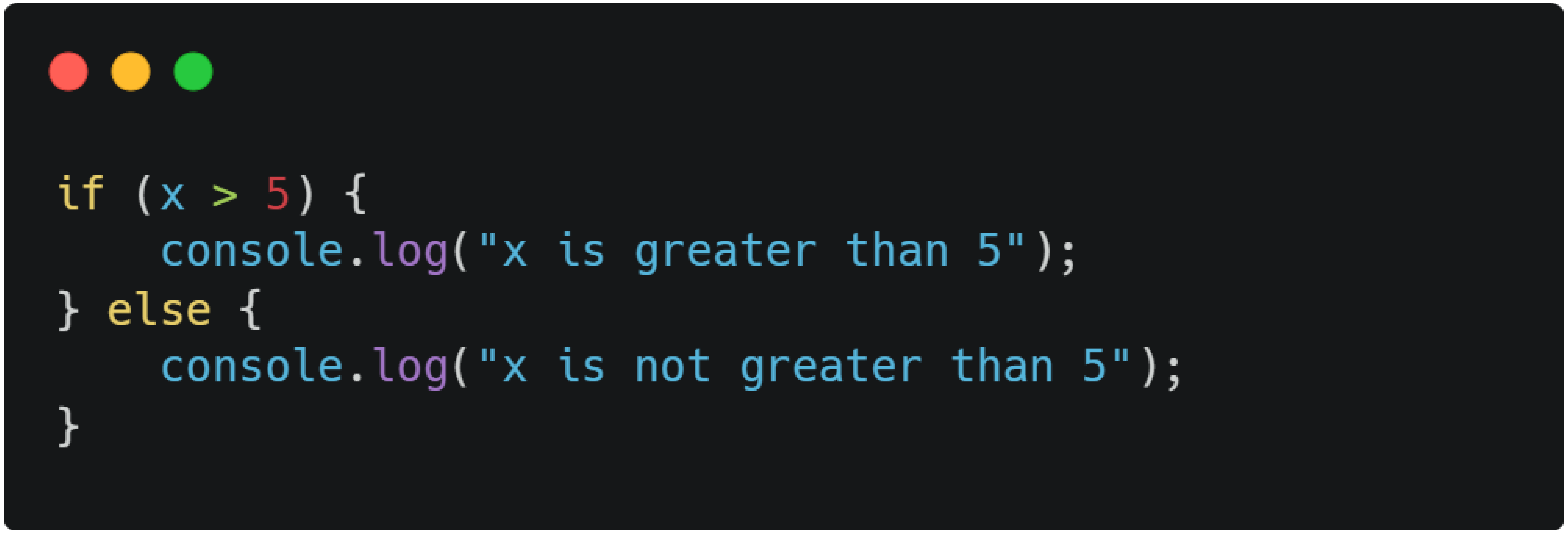
if Statement

- Conditional execution using the if statement.
- Handling multiple conditions with else if and else.
- Practical examples of if statements.



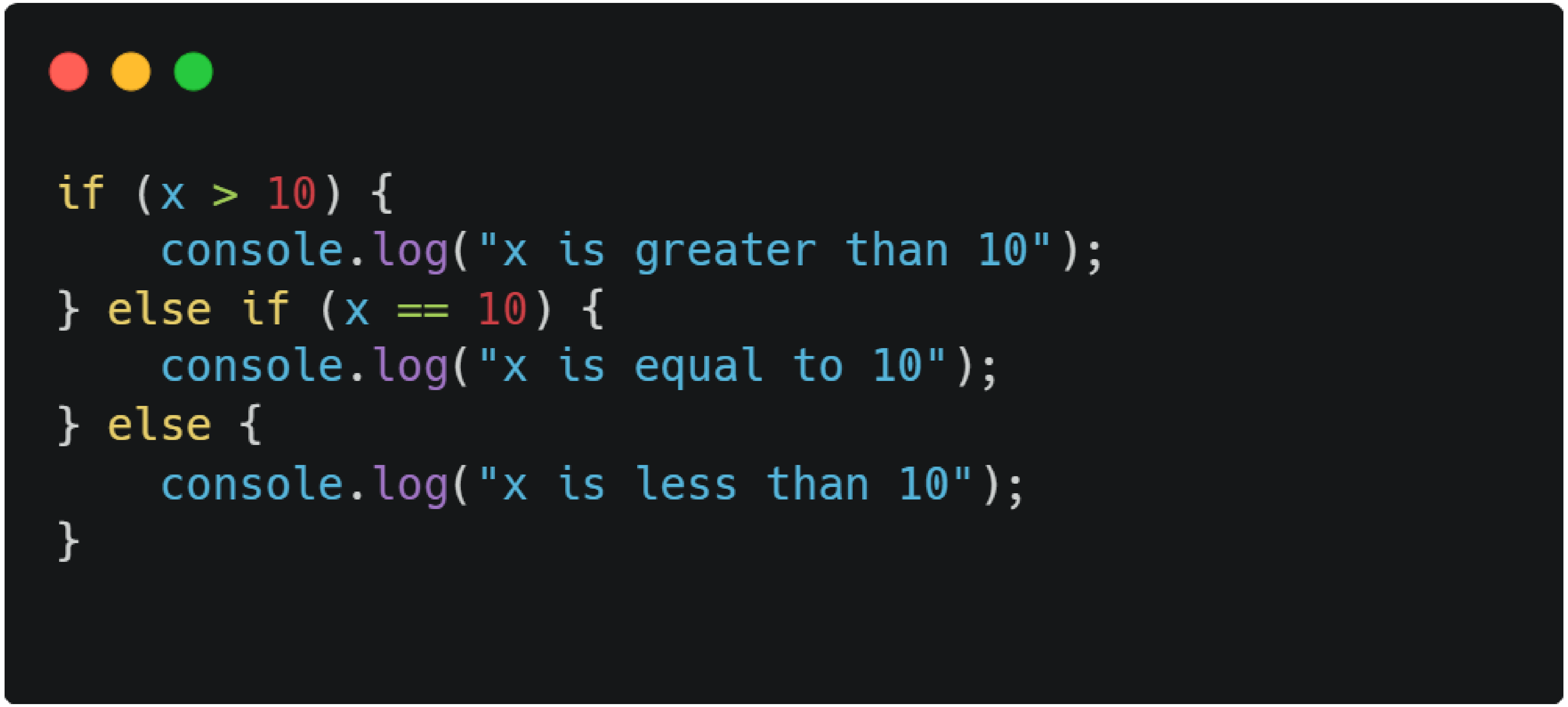
```
let x = 10;
if (x > 5) {
    console.log("x is greater than 5");
}
```

If else



```
if (x > 5) {
    console.log("x is greater than 5");
} else {
    console.log("x is not greater than 5");
}
```

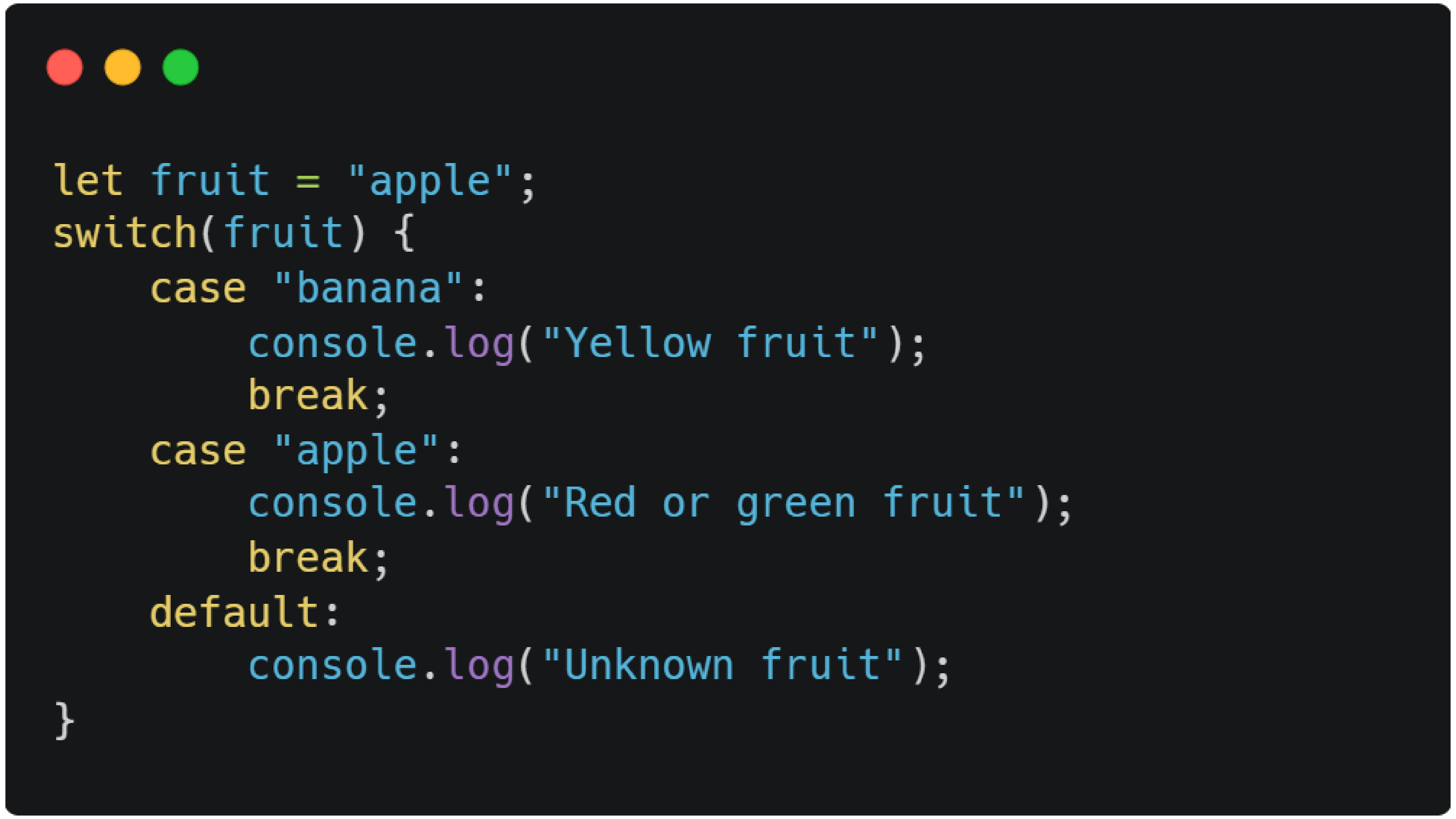
if-else if-else statement



```
if (x > 10) {
    console.log("x is greater than 10");
} else if (x == 10) {
    console.log("x is equal to 10");
} else {
    console.log("x is less than 10");
}
```

switch Statement

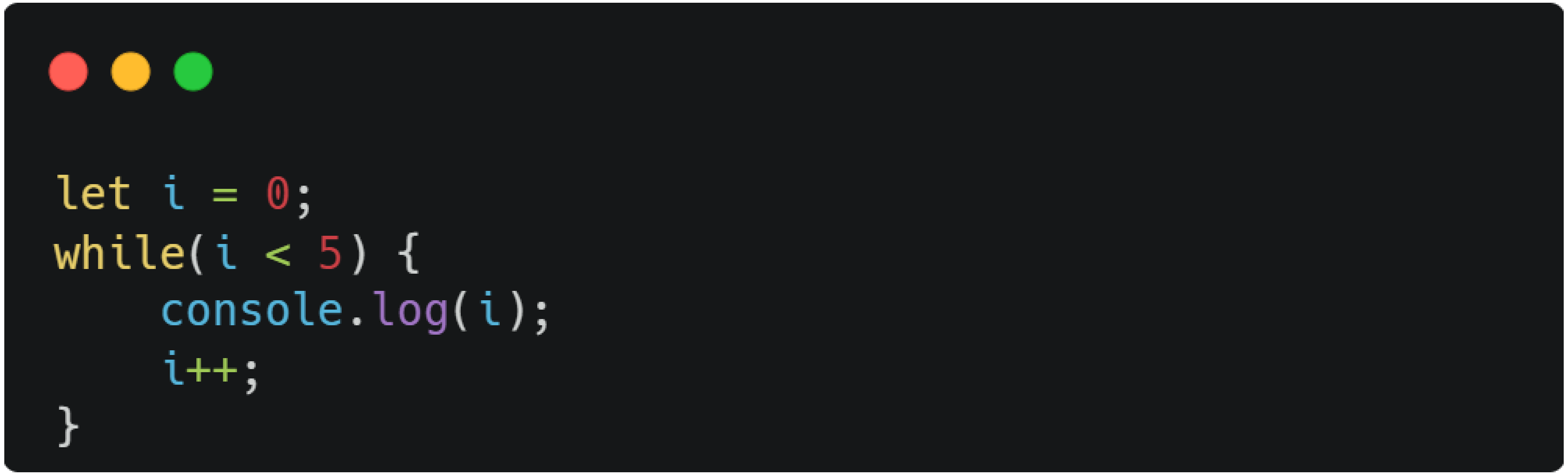
- Simplifying multiple conditions with the switch statement.
- When to use switch over if-else statements.
- Examples of switch statements in action.



```
let fruit = "apple";
switch(fruit) {
    case "banana":
        console.log("Yellow fruit");
        break;
    case "apple":
        console.log("Red or green fruit");
        break;
    default:
        console.log("Unknown fruit");
}
```

while Loop

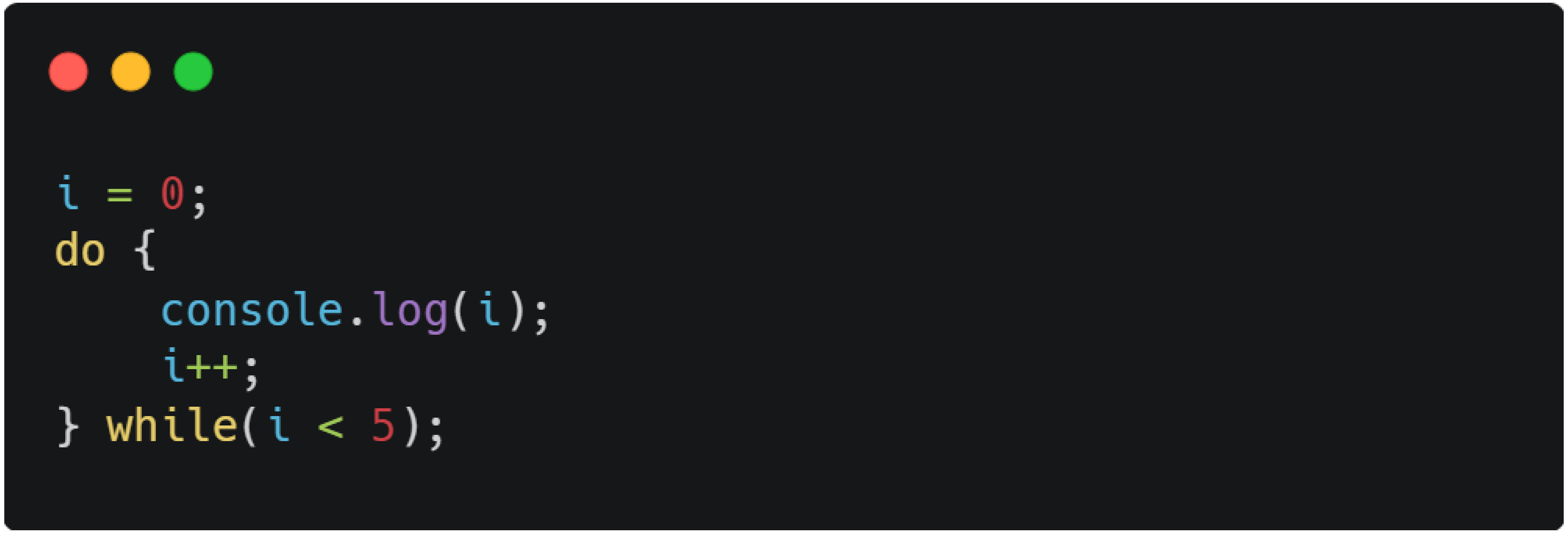
- Introduction to while loops.
- Using while loops for repetitive tasks.
- Practical examples of while loops.



```
let i = 0;
while(i < 5) {
    console.log(i);
    i++;
}
```

do...while Loop

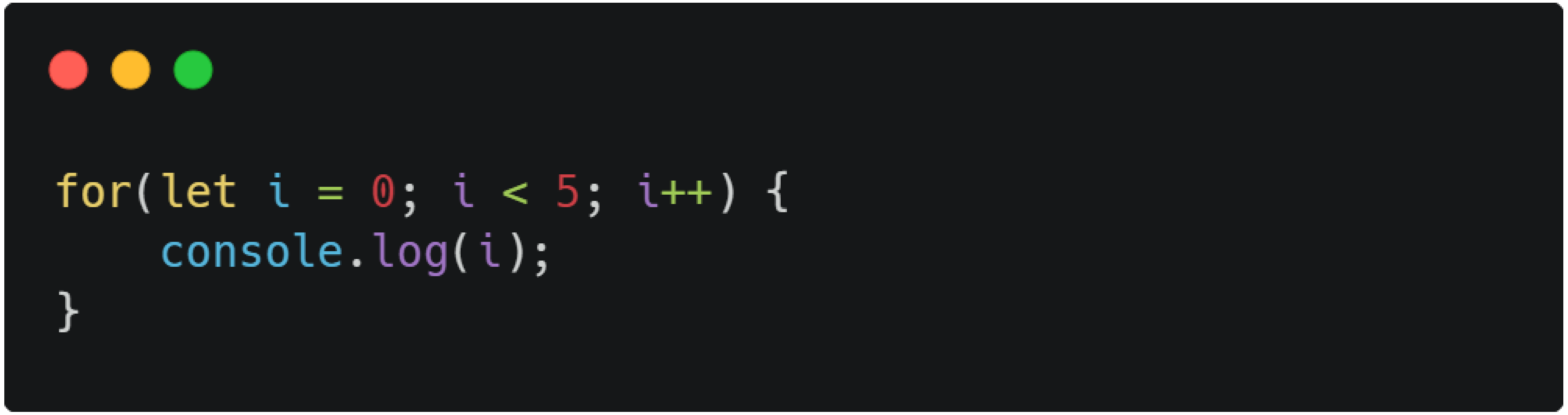
- Differences between do...while and while loops.
- Practical applications of do...while loops.



```
i = 0;
do {
    console.log(i);
    i++;
} while(i < 5);
```

for Loop

- Introduction to for loops.
- Controlling loop iterations with initialization, condition, and increment.
- Looping through arrays with for loops.



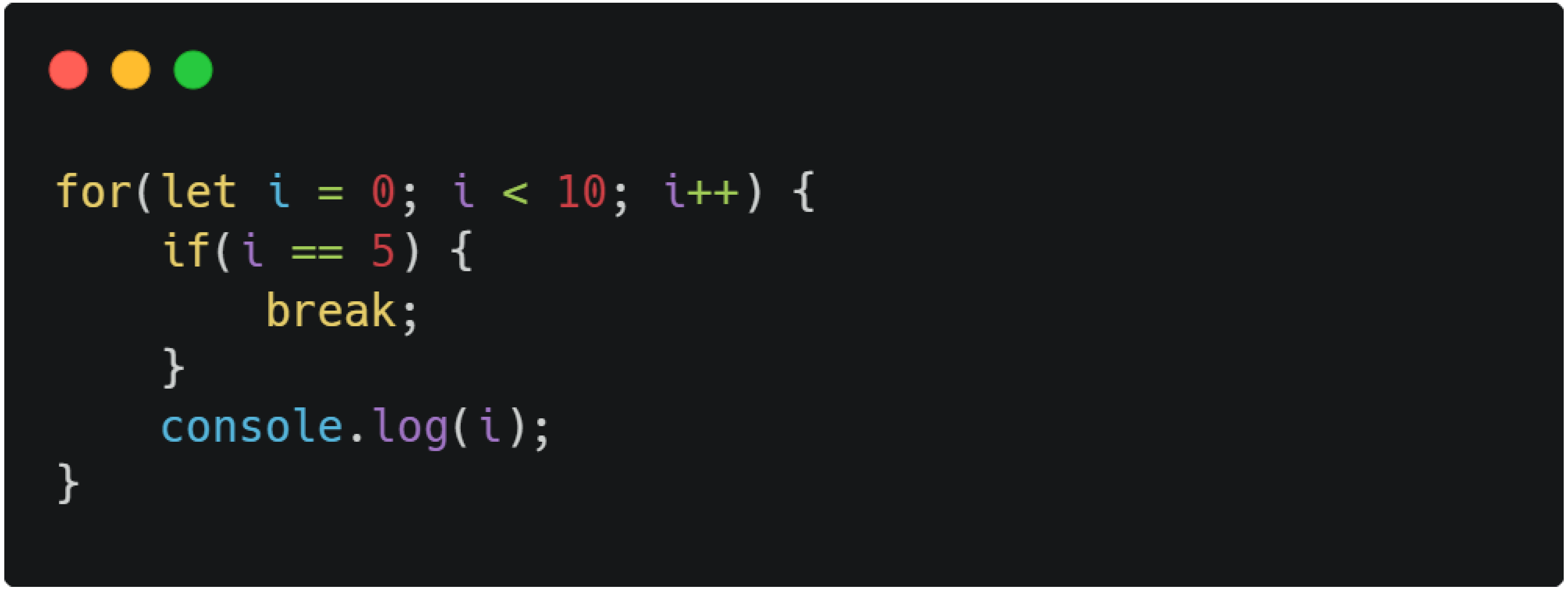
```
for(let i = 0; i < 5; i++) {
    console.log(i);
}
```

Jump Statements

- Using break to exit loops prematurely.
- Using continue to skip current iterations.
- When to apply break and continue in loops.

break

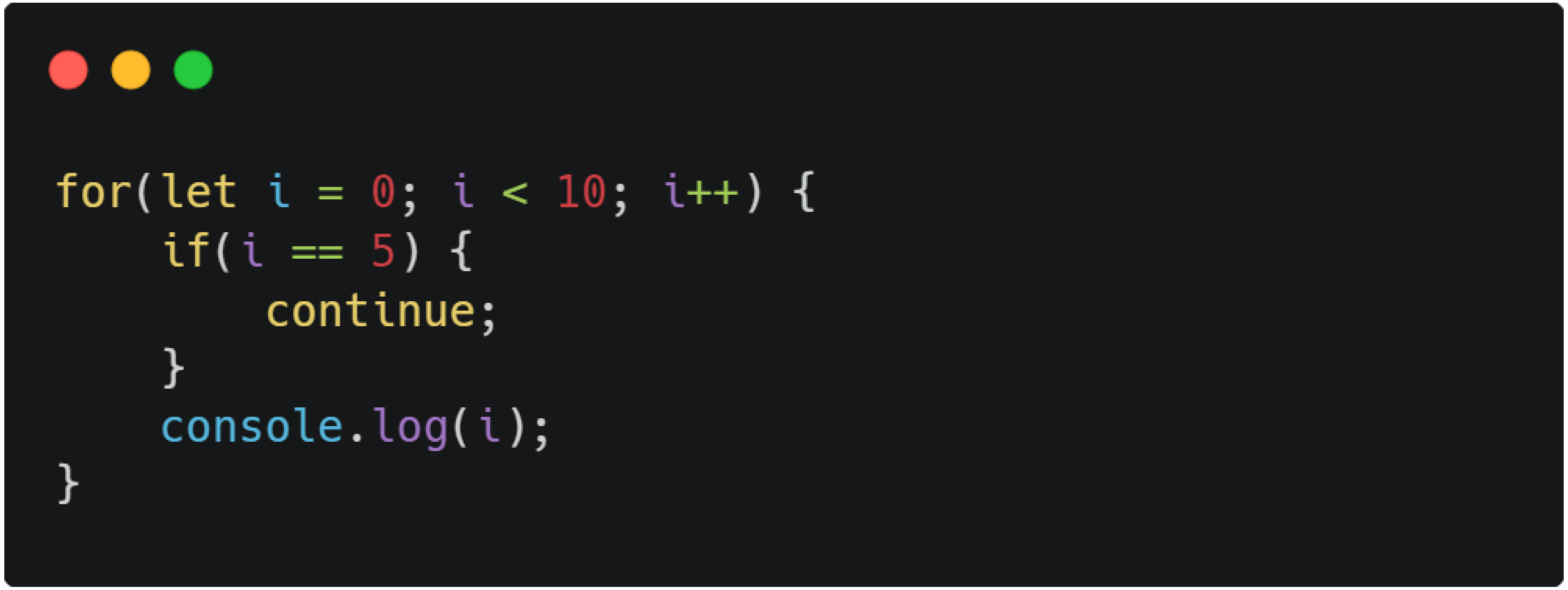
(used to exit from a loop or switch statement)



```
for(let i = 0; i < 10; i++) {
    if(i == 5) {
        break;
    }
    console.log(i);
}
```

continue

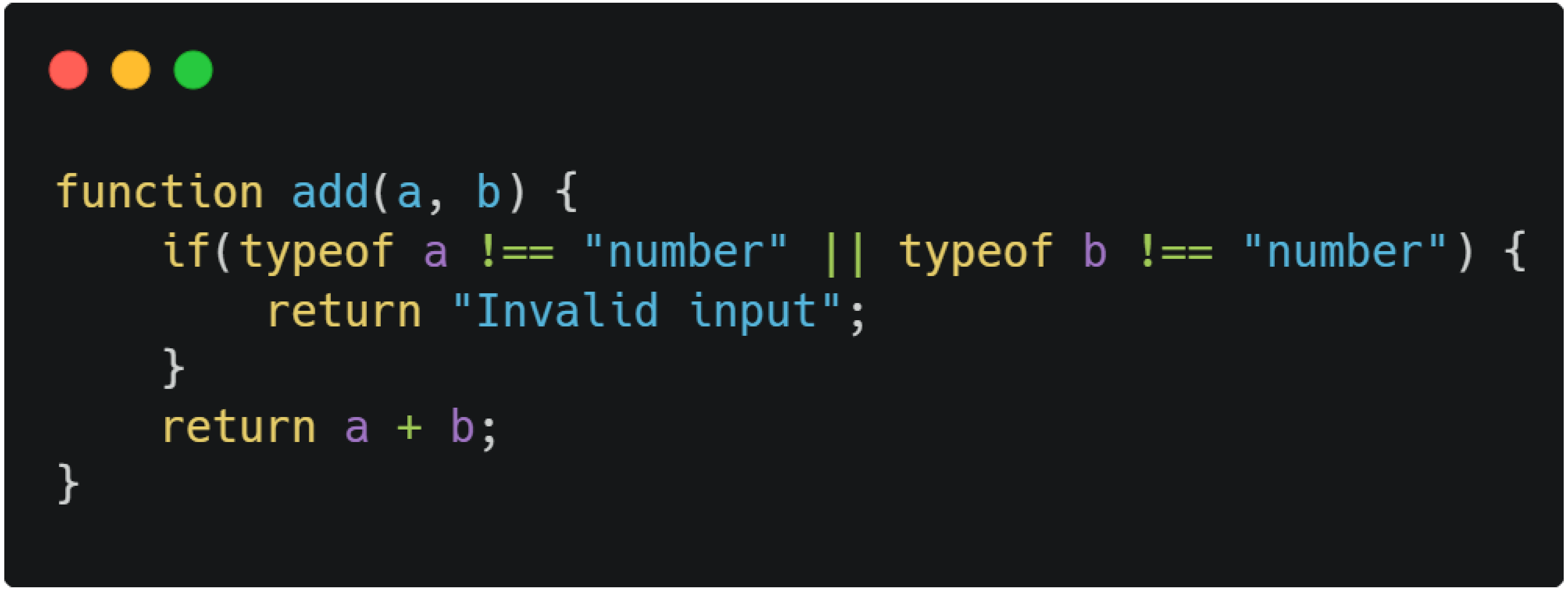
(used to skip the rest of the current loop iteration)



```
for(let i = 0; i < 10; i++) {
    if(i == 5) {
        continue;
    }
    console.log(i);
}
```

return

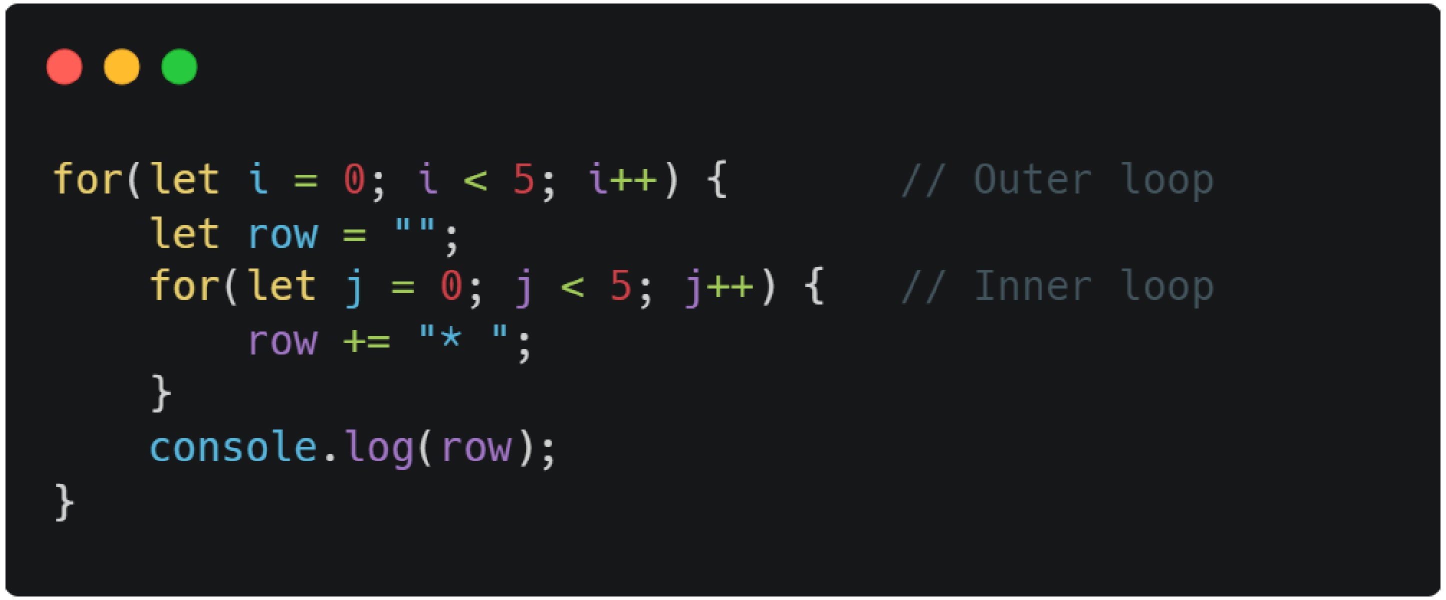
(used to exit from a function)



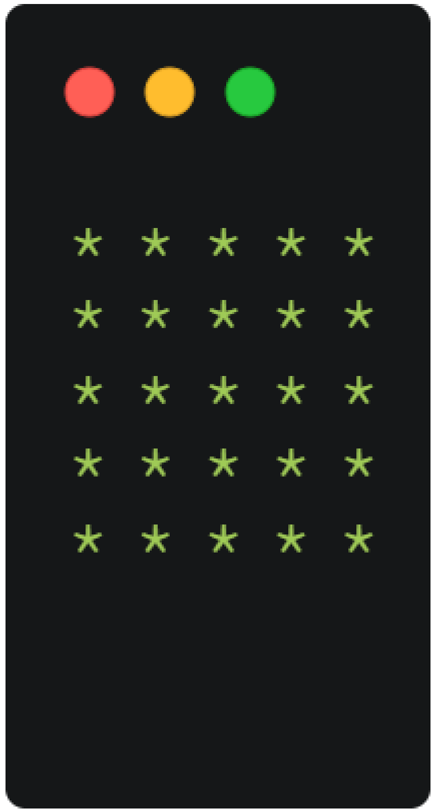
```
function add(a, b) {
    if(typeof a !== "number" || typeof b !== "number") {
        return "Invalid input";
    }
    return a + b;
}
```

Nested Loops

- Nesting loops for complex patterns and tasks.
- Practical examples of nested loops.
- Considerations for efficient use of nested loops.



```
for(let i = 0; i < 5; i++) {       // Outer loop
    let row = "";
    for(let j = 0; j < 5; j++) {   // Inner loop
        row += "* ";
    }
    console.log(row);
}
```



```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

References

- Mozilla Developer Network (MDN) - developer.mozilla.org
- W3Schools - www.w3schools.com

وزارة التعليم العالي والبحث العلمي

جامعة برج العرب التكنولوجية
BORG AL ARAB TECHNOLOGICAL UNIVERSITY