Objectives

- Comparison and Logical Operators
- Working with Conditional Statements
- Working with Loops

Comparison and Logical Operators

Introduction

The code and scripts we have seen so far are linear, but we can add more control to our script's behavior by using **comparison operators** and **logical operators**. These allow us to implement conditions based on values and combine multiple conditions for more complex decision-making.

Comparison Operators

With comparison operators, we can compare two values and determine if the first operand is: Greater than, Less than, Equal to or Not equal tothe second operand.

Common Comparison Operators:

- `==` : Checks equality but ignores type

```
let a = 4;
let b = 6;
console.log(a == b); // false
let a = 4;
let b = "4";
console.log(a == b); // true
```

- `!=` : Checks non-equality but ignores type

```
let a = 4;
let b = 6;
console.log(a != b); // true
let a = 4;
let b = "4";
console.log(a != b); // false
```

- `===` : Checks **strict equality** both value **and** type must match.

```
let a = 4;
let b = "4";
console.log(a === b); // false
```

- `!==` : Checks **strict non-equality** true if value **or** type differ.

```
let a = 4;
let b = "4";
console.log(a !== b); // true
```

- `>` : Checks if greater **than**

```
let a = 4;
let b = 6;
console.log(a > b); // false
```

- `<` : Checks if smaller **than**

```
let a = 4;
let b = 6;
console.log(a < b); // true
```

- `>=` : Checks if greater than or equal to

```
let a = 4;
let b = 6;
console.log(a >= b); // false
```

- `<=` : Checks if less than or equal to

```
let a = 4;
let b = 6;
console.log(a <= b); // true
```

Logical Operators

Logical operators allow us to combine or invert conditions using boolean logic (`true` / `false`).to create more complex conditions..

Common Logical Operators:

- `&&` **(AND)**: Returns `true` if **both** operands are true.

```
let isSunny = true;
let isWarm = true;
console.log(isSunny && isWarm); // true
```

- `||` **(OR)**: Returns `true` if **at least one** operand is true.

```
let hasCoffee = false;
let hasTea = true;
console.log(hasCoffee || hasTea); // true
```

- `!` **(NOT)**: Inverts the boolean value of a single operand.

```
let isDark = false;
console.log(!isDark); // true
```

Truth Table for Logical Operators:

A	B	A && B	A | B	!A
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Conditional Statements

Introduction

JavaScript provides us with conditional statements, which add precise control over our code's behavior and make it more dynamic. These statements let us decide how our code executes based on conditions like values, user input, or other variables, making our programs adaptable and responsive.

If Else Statements

The first conditional statement in JavaScript is the **`if...else` statement**. It allows us to execute code based on conditions. There are three basic ways to use it:

Single Condition

This block runs **only** when the condition evaluates to `true`:

```
if (condition) {
  // Code to run if condition is true
}
```

Condition with Alternative

- Run one block if true, another if false:

```
if (condition) {
  // Code to run if true
} else {
  // Code to run if false
}
```

Multiple Conditions

- Chain conditions with `else if` for alternative paths:

```
if (condition1) {
  // Code for condition1
} else if (condition2) {
  // Code for condition2
} else {
  // Default code if all conditions fail
}
```

Switch Case Statements

The `switch` statement allows **us** to execute different code blocks based on the value of an expression. It’s ideal for scenarios with **multiple fixed conditions** (e.g., menu options, status codes).

```
switch (expression) {
  case value1:
    // Code for value1
    break;
  case value2:
    // Code for value2
    break;
  default:
    // Code if no case matches
}
```

Example

```
let day = "Monday";
switch (day) {
  case "Monday":
    console.log("Weekday 1");
    break;
  case "Tuesday":
    console.log("Weekday 2");
    break;
  default:
    console.log("Not Monday/Tuesday");
}
// It will display "Weekday 1"
```

Loops

Introduction

Loops are programming constructs that allow us to **repeat blocks of code** efficiently. They automate repetitive tasks and can execute instructions based on:

1. **Conditions** (e.g., `while` loops).
2. **Number of iterations** (e.g., `for` loops).
3. **Arrays/objects** (e.g., `for...of` or `for...in` loops).

While Loop

We use a `while` **loop** to repeat code **as long as a condition remains true**. The loop executes its instructions repeatedly until the condition becomes false.

Example

Imagine pouring water into a cup until it’s full. Here’s how this translates to a `while` loop:

```
let isNotFull = true;

while (isNotFull) {
  console.log("Adding water...");
  // Eventually, set isNotFull = false when the cup is full
}
```

- **Condition**: `isNotFull` keeps the loop running.
- **Action**: The loop keeps "adding water" until the condition fails.

Do While Loop

The `do...while` **loop** works similarly to a `while` loop, repeating code based on a condition. However, there’s a key difference:

- A `do...while` loop always runs its code block at least once, even if the condition is initially false.
- A `while` loop skips execution entirely if the condition is false at the start.

Example

The previous example of pouring water into a cup can be rewritten using a `do...while` loop as follows

```
let isNotFull = false;

do {
  console.log("Adding water..."); // This runs ONCE, even though the condition is
false
} while (isNotFull);
```

For Loop

The **`for` loop** is used when we need precise control over iterations. Unlike while loops, for loops combine initialization, condition checking, and iteration into one compact structure.

Repeat Code N Times

We use a `for` loop when we need to execute code a specific number of times. The loop maintains our counter and stops after reaching our target number.

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

Using Number Ranges

We can work with number ranges efficiently using `for` loops. We initialize our counter, set an end condition, and define our increment.

```
for (let num = 10; num <= 15; num++) {
  console.log(num);
}
```

Array Iteration (Traditional)

When processing arrays, we can loop from index 0 to `array.length` to access each element:

```
const colors = ['red', 'green', 'blue'];
for (let i = 0; i < colors.length; i++) {
  console.log(colors[i]);
}
```

Array Iteration (for...of)

We can use `for...of` for cleaner array iteration, giving us direct access to each value:

```
const colors = ['red', 'green', 'blue'];
for (const color of colors) {
  console.log(color);
}
```

Object Iteration (for...in)

When working with objects, we use `for...in` to loop through our properties:

```
const person = {name: 'Alex', age: 30};
for (const key in person) {
  console.log(`${key}: ${person[key]}`);
}
```

Task

Task 1

Create a script that simulates a decision-making process based on weather conditions. The script should:

- Declare variables `isRaining` and `isWarm` (both Boolean values).
- Use a combination of **logical operators** to decide whether you need to bring an umbrella and wear a jacket:
  - If it is raining and not warm, print "Bring an umbrella and wear a jacket."
  - If it is raining but warm, print "Bring an umbrella but no jacket needed."
  - If it is not raining but warm, print "No umbrella needed, but wear light clothes."
  - If it is neither raining nor warm, print "No umbrella and no jacket needed."

Task 2

Write a script that prints numbers from 1 to 100, but with a twist:

- For numbers divisible by 3, print "Fizz".
- For numbers divisible by 5, print "Buzz".
- For numbers divisible by both 3 and 5, print "FizzBuzz".
- For all other numbers, print the number itself.

Task 3

Write a script that uses a **`while` loop** to find the first prime number greater than 100. The script should:

- Start from the number n and check each number to see if it is prime.
- To check if a number is prime, use another **`while` loop** inside the main loop to check divisibility of the number by any number less than it.

- Once a prime number is found, print it and exit the loop.