Agenda

1. Comparison Operators
2. Logical Operators
3. if...else Statement
4. Switch Statement

JavaScript Comparison Operators

Comparison operators compare two values and give back a boolean val either `true` or `false`. Comparison operators are used in decision making and loops

Operator	Description	Example
`==`	**Equal to:** `true` if the operands are equal	`5==5; //true`
`!=`	**Not equal to:** `true` if the operands are not equal	`5!=5; //false`
`===`	**Strict equal to:** `true` if the operands are equal and of the same type	`5==='5'; //false`
`!==`	**Strict not equal to:** `true` if the operands are equal but of different type or not equal at all	`5!=='5'; //true`
`>`	**Greater than:** `true` if the left operand is greater than the right operand	`3>2; //true`
`>=`	**Greater than or equal to:** `true` if the left operand is greater than or equal to the right operand	`3>=3; //true`
`<`	**Less than:** `true` if the left operand is less than the right operand	`3<2; //false`
`<=`	**Less than or equal to:** `true` if the left operand is less than or equal to the right operand	`2<=2; //true`

Example 1: Equal to Operator

```
const a = 5, b = 2, c = 'hello';

// equal to operator
console.log(a == 5);     // true
console.log(b == '2');   // true
console.log(c == 'Hello');  // false
```

`==` evaluates to `true` if the operands are equal.

Note: In JavaScript, `==` is a comparison operator, whereas `=` is an assignm
operator. If you mistakenly use `=` instead of `==`, you might get an unwanted resu

---

Example 2: Not Equal to Operator

```
const a = 3, b = 'hello';

// not equal operator
console.log(a != 2); // true
console.log(b != 'Hello'); // true
```

`!=` evaluates to `true` if the operands are not equal.

Example 3: Strict Equal to Operator

```
const a = 2;

// strict equal operator
```

```
console.log(a === 2); // true
console.log(a === '2'); // false
```

`===` evaluates to `true` if the operands are equal and of the same ty
Here **2** and **'2'** are the same numbers but the data type is different. And `===` a
checks for the data type while comparing.

Note: The difference between `==` and `===` is that:

`==` evaluates to `true` if the operands are equal, however, `===` evalua
to `true` only if the operands are equal and of the same type

Example 4: Strict Not Equal to Operator

```
const a = 2, b = 'hello';

// strict not equal operator
console.log(a !== 2); // false
console.log(a !== '2'); // true
console.log(b !== 'Hello'); // true
```

`!==` evaluates to `true` if the operands are strictly not equal. It's the compl
opposite of strictly equal `===`.

In the above example, `2 != '2'` gives `true`. It's because their types are different ev
though they have the same value.

Example 5: Greater than Operator

```
const a = 3;
```

```
// greater than operator
console.log(a > 2); // true
```

`>` evaluates to `true` if the left operand is greater than the right operand.

Example 6: Greater than or Equal to Operator

```
const a = 3;

// greater than or equal operator
console.log(a >= 3); //true
```

`>=` evaluates to `true` if the left operand is greater than or equal to the right opera

Example 7: Less than Operator

```
const a = 3, b = 2;

// less than operator
console.log(a < 2); // false
console.log(b < 3); // true
```

`<` evaluates to `true` if the left operand is less than the right operand.

Example 8: Less than or Equal to Operator

```
const a = 2;

// less than or equal operator
```

```
console.log(a <= 3) // true
console.log(a <= 2); // true
```

`<=` evaluates to `true` if the left operand is less than or equal to the right operand.

JavaScript Logical Operators

Logical operators perform logical operations: **AND**, **OR** and **NOT**.

Operator	Description	Example
`&&`	**Logical AND**: `true` if both the operands/boolean values are true, else evaluates to `false`	`true && false; // false`
`||`	**Logical OR**: `true` if either of the operands/boolean values is `true`. evaluates to `false` if both are `false`	`true || false; // true`
`!`	**Logical NOT**: `true` if the operand is `false` and vice-versa.	`!true; // false`

Example 9: Logical AND Operator

```
const a = true, b = false;
const c = 4;

// logical AND
console.log(a && a); // true
console.log(a && b);  // false

console.log((c > 2) && (c < 2)); // false
```

`&&` evaluates to `true` if both the operands are `true`, else evaluates to `false`.

Note: You can also use logical operators with numbers. In JavaScript, 0 is `false` a
all non-zero values are `true`.

Example 10: Logical OR Operator

```
const a = true, b = false, c = 4;

// logical OR
console.log(a || b); // true
console.log(b || b); // false
console.log((c>2) || (c<2)); // true
```

`||` evaluates to `true` if either of the operands is `true`. If both operands are `fals`
the result is `false`.

Example 11: Logical NOT Operator

```
const a = true, b = false;

// logical NOT
console.log(!a); // false
console.log(!b); // true
```

`!` evaluates to `true` if the operand is `false` and vice-versa.

JavaScript if...else Statement

In JavaScript, there are three forms of the `if...else` statement.

1. **if** statement
2. **if...else** statement

3. **if...else if...else** statement

JavaScript if Statement

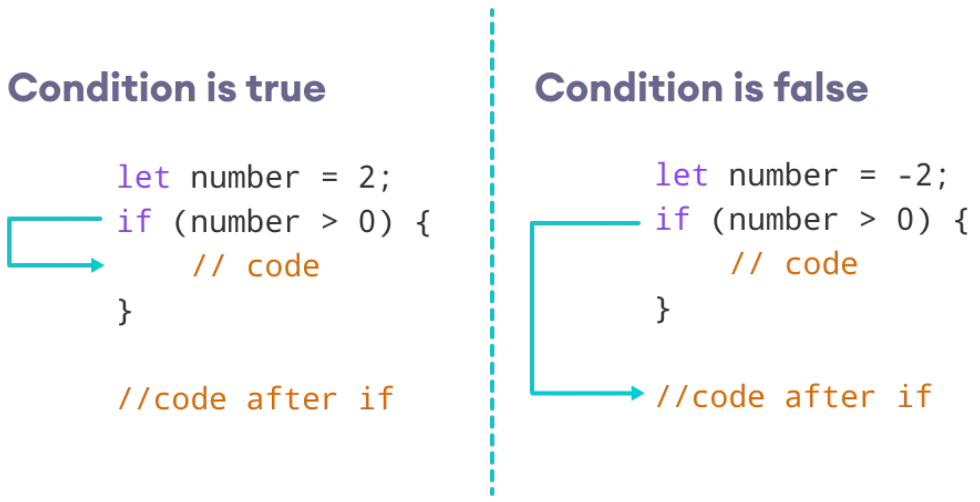
The syntax of the `if` statement is:

```
if (condition) {
    // the body of if
}
```

The `if` statement evaluates the condition inside the parenthesis `()`.

1. If the condition is evaluated to `true`, the code inside the body of `if` is executed.
2. If the condition is evaluated to `false`, the code inside the body of `if` is skipped.

Note: The code inside `{ }` is the body of the `if` statement.



```
let number = 2;
if (number > 0) {
    // code
}

//code after if
```

```
let number = -2;
if (number > 0) {
    // code
}

//code after if
```

Example 1: if Statement

```
// check if the number is positive

const number = prompt("Enter a number: ");

// check if number is greater than 0
if (number > 0) {
 // the body of the if statement
  console.log("The number is positive");
}

console.log("The if statement is easy");
```

Output 1

```
Enter a number: 2
The number is positive
The if statement is easy
```

Suppose the user entered **2**. In this case, the condition `number > 0` evalua to `true`. And, the body of the `if` statement is executed.

Output 2

```
Enter a number: -1
The if statement is easy
```

Suppose the user entered **-1**. In this case, the condition `number > 0` evalua to `false`. Hence, the body of the `if` statement is skipped.

Since `console.log("The if statement is easy");` is outside the body the `if` statement, it is always executed.

JavaScript if...else statement

An `if` statement can have an optional `else` clause. The syntax the `if...else` statement is:

```
if (condition) {
    // block of code if condition is true
} else {
   // block of code if condition is false
}
```

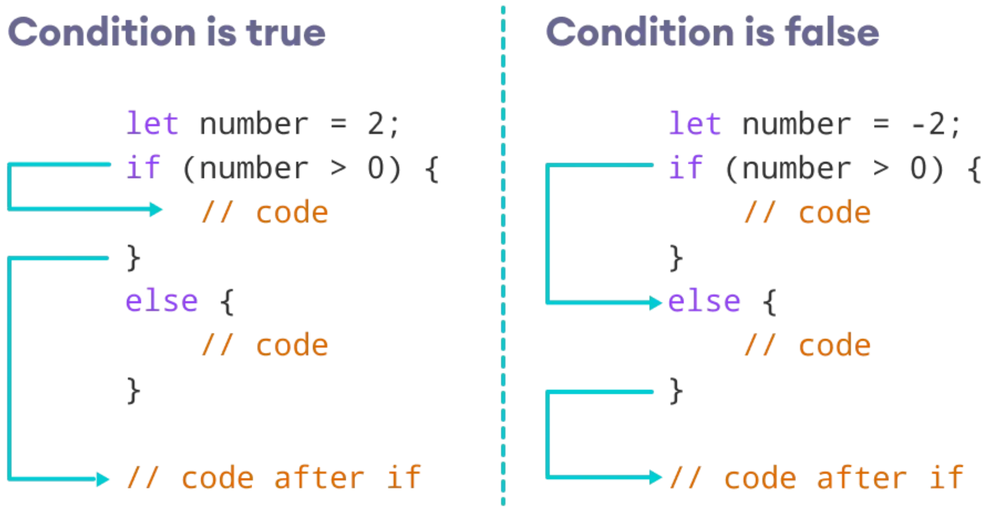
The `if..else` statement evaluates the **condition** inside the parenthesis.

If the condition is evaluated to `true`,

1. the code inside the body of `if` is executed
2. the code inside the body of `else` is skipped from execution

If the condition is evaluated as `false`,

1. the code inside the body of `else` is executed
2. the code inside the body of `if` is skipped from execution



```
let number = 2;
if (number > 0) {
    // code
}
else {
    // code
}

// code after if
```

```
let number = -2;
if (number > 0) {
    // code
}
else {
    // code
}

// code after if
```

Working on the if...else statement

Example 2: if…else Statement

```
// check if the number is positive or negative/zero

const number = prompt("Enter a number: ");

// check if number is greater than 0
if (number > 0) {
  console.log("The number is positive");
}
// if number is not greater than 0
else {
  console.log("The number is either a negative number or 0");
}

console.log("The if...else statement is easy");
```

Output 1

```
Enter a number: 2
The number is positive
The if...else statement is easy
```

Suppose the user entered **2**. In this case, the condition `number > 0` evalua to `true`. Hence, the body of the `if` statement is executed and the body the `else` statement is skipped.

Output 2

```
Enter a number: -1
The number is either a negative number or 0
The if...else statement is easy
```

Suppose the user entered **-1**. In this case, the condition `number > 0` evalua to `false`. Hence, the body of the `else` statement is executed and the body the `if` statement is skipped.

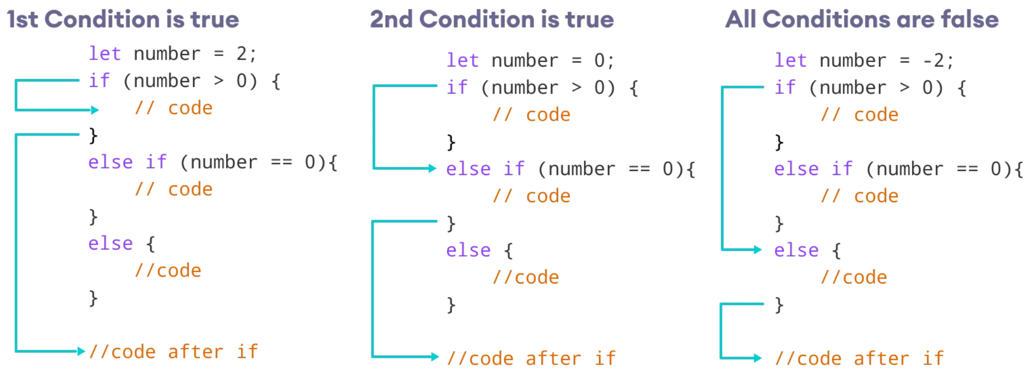
JavaScript if...else if statement

The `if...else` statement is used to execute a block of code among two alternativ However, if you need to make a choice between more than two alternatives, `if...e if...else` can be used.

The syntax of the `if...else if...else` statement is:

```
if (condition1) {
    // code block 1
} else if (condition2){
    // code block 2
} else {
    // code block 3
}
```

- If **condition1** evaluates to `true`, the **code block 1** is executed.
- If **condition1** evaluates to `false`, then **condition2** is evaluated.
- If the **condition2** is `true`, the **code block 2** is executed.
- If the **condition2** is `false`, the **code block 3** is executed.



```
let number = 2;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

```
let number = 0;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

```
let number = -2;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

Example 3: if...else if Statement

```
// check if the number if positive, negative or zero
const number = prompt("Enter a number: ");

// check if number is greater than 0
if (number > 0) {
    console.log("The number is positive");
}
// check if number is 0
else if (number == 0) {
  console.log("The number is 0");
}
// if number is neither greater than 0, nor zero
```

```
else {
    console.log("The number is negative");
}

console.log("The if...else if...else statement is easy");
```

Output

```
Enter a number: 0
The number is 0
The if...else if...else statement is easy
```

Suppose the user entered **0**, then the first test condition `number > 0` evalua to `false`. Then, the second test condition `number == 0` evaluates to `true` and corresponding block is executed.

Nested if...else Statement

You can also use `if...else` statement inside of an `if...else` statement. This is kno as a **nested if...else** statement.

Example 4: Nested if...else Statement

```
// check if the number is positive, negative or zero
const number = prompt("Enter a number: ");

if (number >= 0) {
    if (number == 0) {
        console.log("You entered number 0");
    } else {
        console.log("You entered a positive number");
    }
} else {
    console.log("You entered a negative number");
}
```

Output

```
Enter a number: 5
You entered a positive number
```

Suppose the user entered **5**. In this case, the condition `number >= 0` evalua to `true`, and the control of the program goes inside the outer `if` statement.

Then, the test condition, `number == 0`, of the inner `if` statement is evaluated. Sir it's false, the `else` clause of the inner `if` statement is executed.

Note: As you can see, nested `if...else` makes our logic complicated and we should to avoid using nested `if...else` whenever possible.

Body of if...else With Only One Statement

If the body of `if...else` has only one statement, we can omit `{ }` in our prograr For example, you can replace

```
const number = 2;
if (number > 0) {
    console.log("The number is positive.");
} else {
   console.log("The number is negative or zero.");
}
```

with

```
const number = 2;
if (number > 0)
    console.log("The number is positive.");
 else
   console.log("The number is negative or zero.");
```

Output

```
The number is positive.
```

JavaScript Switch Statement

The JavaScript `switch` statement is used in decision making.

The `switch` statement evaluates an expression and executes the corresponding bc
that matches the expression's result.

The syntax of the `switch` statement is:

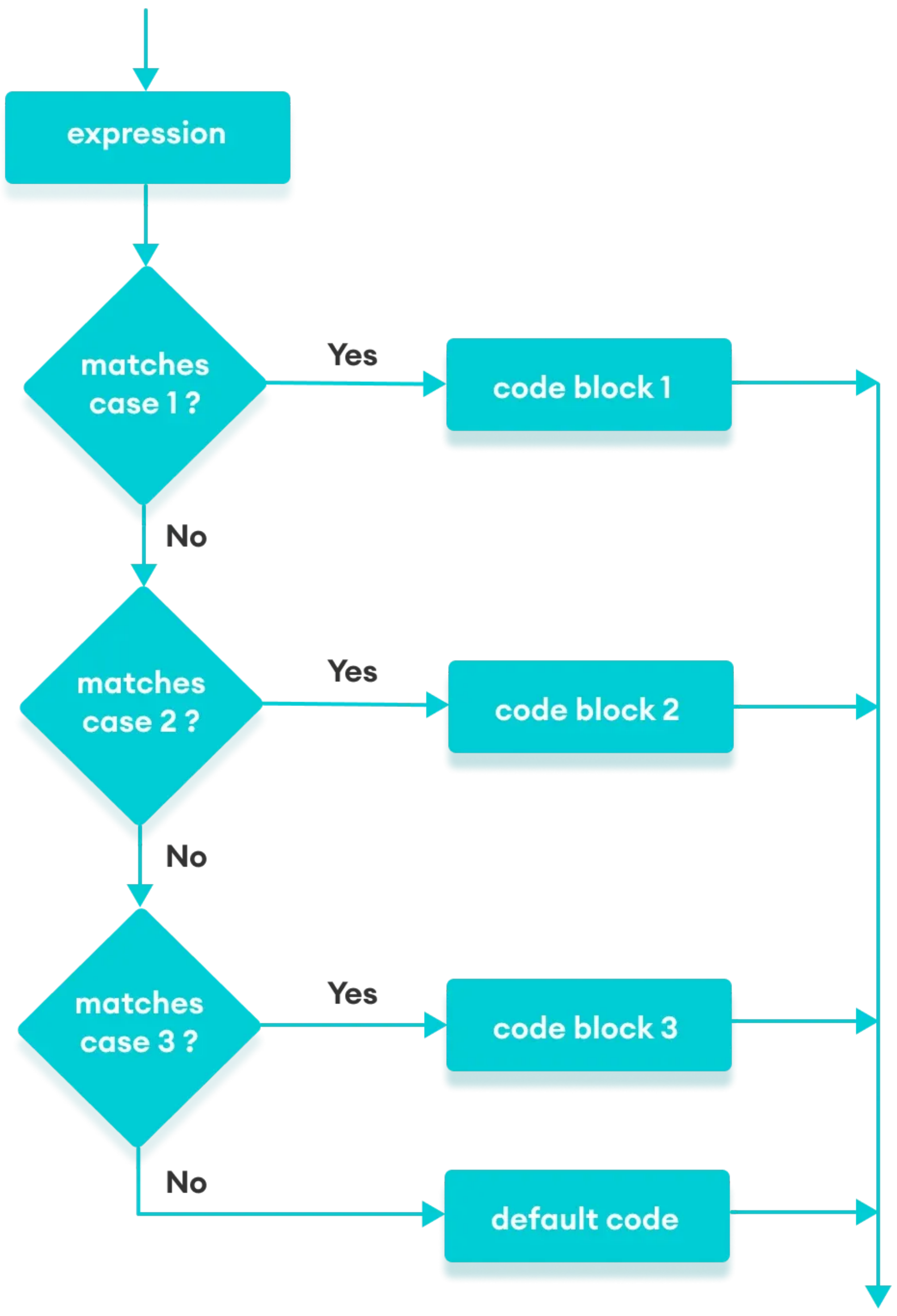
```
switch(variable/expression) {
    case value1:
        // body of case 1
        break;

    case value2:
        // body of case 2
        break;

    case valueN:
        // body of case N
        break;

    default:
        // body of default
}
```

The `switch` statement evaluates a variable/expression inside parentheses `()`.

- If the result of the expression is equal to `value1`, its body is executed.
- If the result of the expression is equal to `value2`, its body is executed.
- This process goes on. If there is no matching case, the `default` body executes.

Notes:

- The `break` statement is optional. If the break statement is encountered, the switch statement ends.
- If the `break` statement is not used, the cases after the matching case are also executed.
- The `default` clause is also optional.



Example 1: Simple Program Using switch Statement

```
// program using switch statement
let a = 2;
```

```
switch (a) {

    case 1:
        a = 'one';
        break;
    case 2:
        a = 'two';
        break;
    default:
        a = 'not found';
        break;
}
console.log(`The value is ${a}`);
```

Output

```
The value is two.
```

In the above program, an expression `a = 2` is evaluated with a `switch` statement.

- The **expression's** result is evaluated with `case 1` which results in `false`.
- Then the `switch` statement goes to the second case. Here, the **expression's** result matches with `case 2`. So `The value is two` is displayed. The value is two
- The `break` statement terminates the block and control flow of the program jumps to outside of the `switch` block.

Example 2: Type Checking in switch Statement

```
// program using switch statement
let a = 1;

switch (a) {
    case "1":
        a = 1;
        break;
    case 1:
        a = 'one';
        break;
    case 2:
        a = 'two';
        break;

    default:
        a = 'not found';
        break;
}
console.log(`The value is ${a}`);
```

Output

```
The value is one.
```

In the above program, an expression `a = 1` is evaluated with a `switch` statement.

- In JavaScript, the switch statement checks the value strictly. So the expression's result does not match with `case "1"`.
- Then the `switch` statement goes to the second case. Here, the expressions' result matches with `case 1`. So `The value is one` displayed. The value is one

- The `break` statement terminates the block and the control flow of the program jumps outside of the `switch` block.

Note: In JavaScript, the switch statement checks the cases strictly (should be of the same data type) with the expression's result. Notice in the above example, that **1** dc not match with **"1"**.

Let's write a program to make a simple calculator with the `switch` statement.

Example 3: Simple Calculator

```
// program for a simple calculator
let result;

// take the operator input
const operator = prompt('Enter operator ( either +, -, * or / ): ');

// take the operand input
const number1 = parseFloat(prompt('Enter first number: '));
const number2 = parseFloat(prompt('Enter second number: '));

switch(operator) {
    case '+':
        result = number1 + number2;
        console.log(`${number1} + ${number2} = ${result}`);
        break;
    case '-':
        result = number1 - number2;
        console.log(`${number1} - ${number2} = ${result}`);
        break;
    case '*':
        result = number1 * number2;
        console.log(`${number1} * ${number2} = ${result}`);
        break;
    case '/':
        result = number1 / number2;
        console.log(`${number1} / ${number2} = ${result}`);
        break;

    default:
        console.log('Invalid operator');
        break;
}
```

Output

```
Enter operator: +
Enter first number: 4
Enter second number: 5
4 + 5 = 9
```

In the above program, the user is asked to enter either +, -, * or /, and two operan
Then, the `switch` statement executes cases based on the user input.

JavaScript switch With Multiple Case

In a JavaScript switch statement, cases can be grouped to share the same code.

Example 4: switch With Multiple Case

```
// multiple case switch program
let fruit = 'apple';
switch(fruit) {
    case 'apple':
```

```
    case 'mango':
    case 'pineapple':
      console.log(`${fruit} is a fruit.`);
      break;
    default:
      console.log(`${fruit} is not a fruit.`);
      break;
  }
```

Output

```
apple is a fruit.
```

In the above program, multiple cases are grouped. All the grouped cases share the sa code.

If the value of the fruit variable had the value `mango` or `pineapple`, the out would have been the same.

What happens if I forgot a `break` [?]

If you forget a `break` then the script will run from the `case` where the criterior met and will run the cases after that **regardless if a criterion was met.**

See example here:

```
var foo = 0;
switch (foo) {
 case -1:
  console.log('negative 1');
  break;
 case 0: // foo is 0 so criteria met here so this block will run
  console.log(0);
  // NOTE: the forgotten break would have been here
  case 1: // no break statement in 'case 0:' so this case will run as well
    console.log(1);
    break; // it encounters this break so will not continue into 'case 2:'
  case 2:
    console.log(2);
    break;
  default:
    console.log('default');
}
```

Can I put a `default` between cases?

Yes, you can! JavaScript will drop you back to the `default` if it can't find a match:

```
var foo = 5;
switch (foo) {
  case 2:
    console.log(2);
    break; // it encounters this break so will not continue into 'default:'
  default:
    console.log('default')
    // fall-through
  case 1:
    console.log('1');
}
```

It also works when you put `default` before all other `case`s.