Javascript Day -2

Operators, Conditionals and Loops

Types of operators

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators

Arithmetic Operators

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modulus (%)
- Increment (++)
- Decrement (--)

Assignment Operators

- Simple Assignment (=)
- Add and Assign (+=)
- Subtract and Assign (-=)
- Multiply and Assign (*=)
- Divide and Assign (/=)
- Modulus and Assign (%=)

Comparison Operators

- Equal (==)
- Not Equal (!=)
- Strict Equal (===)
- Strict Not Equal (!==)
- Greater Than (>)
- Less Than (<)
- Greater Than or Equal To (>=)
- Less Than or Equal To (<=)

Logical Operators

- Logical AND (&&)
- Logical OR (||)
- Logical NOT (!)

```
let x = 5;
let y = 10;

console.log(x > 3 && y < 20);  // true
console.log(x < 3 || y > 15);  // false
```

Combining Operators

Using different operators in combination

```
let age = 25;
let isAdult = age >= 18 && age <= 65;

console.log(isAdult);  // true
```

Common Mistakes

- Mixing Data Types
- Understanding the difference between == and ===
- Order of operations

Mixing Data Types

JavaScript is a loosely typed language, but mixing incompatible data types can lead to unexpected results.

For example, using the + operator to concatenate strings and add numbers can cause confusion.

```
let result = '10' + 5; // Result: '105'
```

Understanding the difference between == and ===

The == operator performs type coercion, meaning it tries to convert operands to the same type before making a comparison. This can lead to unexpected results.

```
let num1 = 10;
let num2 = '10';

console.log(num1 == num2);
console.log(num1 === num2);
```

Order of Operations

Understanding the order in which operators are executed is crucial to getting the desired results.

```
let result = 5 + 10 * 2;
```

Level	Operators	Description	Associativity
15	() [] -> . ++ --	Function Call Array Subscript Member Selectors Postfix Increment/Decrement	Left to Right
14	++ -- + - ! ~ (type) * & sizeof	Prefix Increment / Decrement Unary plus / minus Logical negation / bitwise complement Casting Dereferencing Address of Find size in bytes	Right to Left
13	* / %	Multiplication Division Modulo	Left to Right
12	+ -	Addition / Subtraction	Left to Right
11	>> <<	Bitwise Right Shift Bitwise Left Shift	Left to Right
10	< <= > >=	Relational Less Than / Less than Equal To Relational Greater / Greater than Equal To	Left to Right
9	== !=	Equality Inequality	Left to Right
8	&	Bitwise AND	Left to Right
7	^	Bitwise XOR	Left to Right
6	|	Bitwise OR	Left to Right
5	&&	Logical AND	Left to Right
4	||	Logical OR	Left to Right
3	?:	Conditional Operator	Right to Left
2	= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operators	Right to Left
1	,	Comma Operator	Left to Right

Conditional Branching (if statement)

- If else
- If else if
- Nested if else

```
1  if (year == 2015) {
2    alert( "That's correct!" );
3    alert( "You're so smart!" );
4  }
```

```
if (year == 2015) {
  alert( 'You guessed it right!' );
} else {
  alert( 'How can you be so wrong?' );
}
```

```
if (year < 2015) {
  alert( 'Too early...' );
} else if (year > 2015) {
  alert( 'Too late' );
} else {
  alert( 'Exactly!' );
}
```

Conditional (Ternary) Operator

The conditional (ternary) operator (?) provides a concise way to write 'if' statements in a single line, but it is not recommended to use as a replacement for if statements because it hampers code readability.

Use Case: Sometimes, we need to assign a variable depending on a condition.

```
condition ? expressionIfTrue : expressionIfFalse;
```

```
let accessAllowed = age > 18 ? true : false;
```

Nullish Coalescing Operator '??'

The Nullish Coalescing Operator (??) is a powerful addition to JavaScript, especially when dealing with default values and nullish (null or undefined) conditions.

```
let result = someValue ?? defaultValue;
```

```
let userInput = null;
let defaultValue = 'Default Value';

let result = userInput ?? defaultValue;
console.log(result); // Outputs: 'Default Value'
```

Nullish Coalescing Operator '??' (comparison with ||)

The || operator returns the right-hand operand if the left-hand operand is falsy.

It does not specifically check for nullish values but includes all falsy values.

i.e If the first value is either **false, o(zero), empty string, null or undefined** the right hand side value will be taken.

Loops in Javascript

Definition: Loops allow us to repeat a set of instructions.

Use Cases: Iterating through arrays, executing code until a condition is met.

1. While loop
2. Do while loop
3. For loop

While Loop

The while loop in JavaScript is used to repeatedly execute a block of code as long as a specified condition remains true.

It is particularly useful when the number of iterations is not known beforehand, and the loop continues until a certain condition is no longer met.

Example: Counting down, User Input Validation

```
while (condition) {
  // Code to be executed as long as the condition is true
}
```

Do While Loop

The do-while loop is another iteration structure in JavaScript, similar to the while loop. However, it has a distinct characteristic: it guarantees that the code block will be executed at least once, even if the condition is false from the beginning.

When to Use Do-While Loops ?

When you need to execute a block of code at least once, regardless of the condition.

Use when the loop's body must be executed before checking the condition.

```
do {
  // Code to be executed
} while (condition);
```

For Loop

The for loop is a powerful and concise way to iterate over a range of values or elements in JavaScript.

It is particularly useful when the number of iterations is known in advance

```
for (initialization; condition; update) {
  // Code to be executed in each iteration
}
```

Break and Continue Statement

Break: Used to exit the loop prematurely if a certain condition is met.

Continue: Skips the rest of the loop's code for the current iteration and moves to

```
for (let i = 0; i < 10; i++) {
  if (i === 5) {
    break; // exit the loop when i is 5
  }
  console.log("Value: " + i);
}
```

```
for (let i = 0; i < 5; i++) {
  if (i === 2) {
    continue; // skip the rest of the loop body when i is 2
  }
  console.log("Value: " + i);
}
```