JAVASCRIPT-3

1. State the difference between while loop, for loop and do while loop.

Differences Between while Loop, for Loop, and do-while Loop in JavaScript

In JavaScript, loops are used to execute a block of code repeatedly until a certain condition is met. The three most common types of loops are **while**, **for**, and **do-while**. Here's how they differ:

1. While Loop

- **Definition** : A **while** loop executes a block of code as long as the specified condition evaluates to **true**.
- **Syntax** :

javascript

```
while (condition) {
    // Code to execute
}
```

- **Key Characteristics** :
    - The condition is checked **before** the execution of the loop body.
    - If the condition is **false** initially, the loop body will **never execute** .
    - Suitable when the number of iterations is not known beforehand.
- **Example** :

javascript

```
let i = 0;
while (i < 5) {
    console.log(i);
    i++;
}
// Output: 0 1 2 3 4
```

2. For Loop

- **Definition** : A **for** loop is used when you know in advance how many times you want to execute a block of code.
- **Syntax** :

javascript

```
for (initialization; condition; increment/decrement) {
    // Code to execute
}
```

- **Key Characteristics** :

- Combines **initialization** , **condition checking** , and **increment/decrement** in one line.
- The condition is checked **before** each iteration.
- If the condition is **false** initially, the loop body will **never execute** .
- Ideal for iterating over arrays or when the number of iterations is known.

- **Example** :

javascript

```
for (let i = 0; i < 5; i++) {
    console.log(i);
}
// Output: 0 1 2 3 4
```

3. Do-While Loop

- **Definition** : A **do-while** loop is similar to a **while** loop, but it guarantees that the loop body will execute **at least once** , even if the condition is **false** from the start.
- **Syntax** :

javascript

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

- **Key Characteristics** :
  - The condition is checked **after** the execution of the loop body.
  - The loop body will always execute **at least once** , regardless of whether the condition is **true** or **false**.
  - Useful when you need to ensure that the code inside the loop runs at least once before checking the condition.
- **Example** :

javascript

```
let i = 0;
do {
    console.log(i);
    i++;
} while (i < 5);
// Output: 0 1 2 3 4
```

Summary of Differences

- **Condition Check Timing** :

    - **while** loop: Condition is checked **before** the loop body.
    - **for** loop: Condition is checked **before** each iteration.
    - **do-while** loop: Condition is checked **after** the loop body.
- **Guaranteed Execution** :
    - **while** loop: May **not execute** if the condition is **false** initially.
    - **for** loop: May **not execute** if the condition is **false** initially.
    - **do-while** loop: Always executes **at least once** , even if the condition is **false**.
- **Use Case** :
    - **while** loop: Best when the number of iterations is **unknown** and depends on a condition.
    - **for** loop: Best when the number of iterations is **known** or when iterating over a range (e.g., arrays).
    - **do-while** loop: Best when you need to ensure the loop body runs **at least once** .

Additional Notes:

- **Break and Continue** : Both **break** and **continue** statements can be used in all three types of loops to control the flow of execution.
    - **break**: Exits the loop entirely.
    - **continue**: Skips the current iteration and proceeds to the next one.
- **Iterating Over Arrays** :
    - In JavaScript, the **for** loop is often used to iterate over arrays, but modern JavaScript also provides methods like **forEach**, **map**, and **for...of** for more concise array iteration.

2. What is the difference between break and continue.

Difference Between break and continue

In programming, both **break** and **continue** are control flow statements used within loops (**for**, **while**, **do-while**) to alter the normal execution of the loop. However, they serve different purposes and behave differently. Here's a detailed explanation of each:

1. break Statement

Definition :

The **break** statement is used to **terminate** the execution of a loop or switch-case block prematurely. When encountered, it immediately exits the loop, and the program continues with the next statement after the loop.

Key Characteristics :

- **Terminates the loop entirely** : Once the **break** statement is executed, the loop stops, and no further iterations occur.
- **Used to exit early** : It is often used when a specific condition is met, and there is no need to continue iterating.
- **Can be used in loops and switch-case blocks** : In addition to loops, **break** is commonly used in **switch-case** statements to prevent "fall-through" behavior.

Syntax :

javascript

```
for (let i = 0; i < 10; i++) {
    if (i === 5) {
        break; // Exit the loop when i equals 5
    }
    console.log(i);
}
```

Example :

javascript

```
for (let i = 0; i < 10; i++) {
    if (i === 5) {
        break; // Loop terminates when i equals 5
    }
    console.log(i);
}
// Output: 0 1 2 3 4
```

In this example, the loop stops executing when **i** equals 5, so the output only includes numbers from 0 to 4.

2. continue Statement

Definition :

The **continue** statement is used to **skip the current iteration** of a loop and proceed to the next iteration. When encountered, it skips the remaining code inside the loop for the current iteration and moves to the next iteration.

Key Characteristics :

- **Skips the current iteration** : The **continue** statement does not terminate the loop but skips the rest of the code for the current iteration.
- **Useful for skipping specific conditions** : It is often used to bypass certain values or conditions without stopping the entire loop.
- **Only works within loops** : Unlike **break**, **continue** cannot be used in **switch-case** blocks.

Syntax :

javascript

```
for (let i = 0; i < 10; i++) {
    if (i === 5) {
        continue; // Skip the current iteration when i equals 5
    }
    console.log(i);
}
```

Example :

javascript

```
for (let i = 0; i < 10; i++) {
    if (i === 5) {
        continue; // Skip the iteration when i equals 5
    }
    console.log(i);
}
// Output: 0 1 2 3 4 6 7 8 9
```

In this example, the number **5** is skipped, and the loop continues with the next iteration.

Summary of Differences

FEATURE	BREAK	CONTINUE
PURPOSE	Terminates the loop entirely	Skips the current iteration and continues
EXECUTION FLOW	Exits the loop immediately	Moves to the next iteration
USE CASE	Exit the loop when a condition is met	Skip specific iterations based on a condition
WORKS IN SWITCH-CASE?	Yes	No
LOOP BEHAVIOR	Stops further iterations	Continues with the next iteration

When to Use break vs. continue

- **Use break** :
    - When you want to exit the loop entirely after meeting a specific condition.
    - Example: Searching for an item in an array and stopping once the item is found.
- **Use continue** :
    - When you want to skip certain iterations based on a condition but continue with the rest of the loop.
    - Example: Skipping even numbers while iterating through a range of numbers.

Practical Examples

Example 1: Using break

javascript

```
// Find the first number divisible by 7 in an array
let numbers = [3, 8, 15, 21, 25];
for (let num of numbers) {
    if (num % 7 === 0) {
        console.log("First number divisible by 7:", num);
        break; // Exit the loop once the condition is met
    }
}
// Output: First number divisible by 7: 21
```

Example 2: Using continue

javascript

```
// Print only odd numbers from 1 to 10
for (let i = 1; i <= 10; i++) {
    if (i % 2 === 0) {
        continue; // Skip even numbers
    }
    console.log(i);
}
// Output: 1 3 5 7 9
```