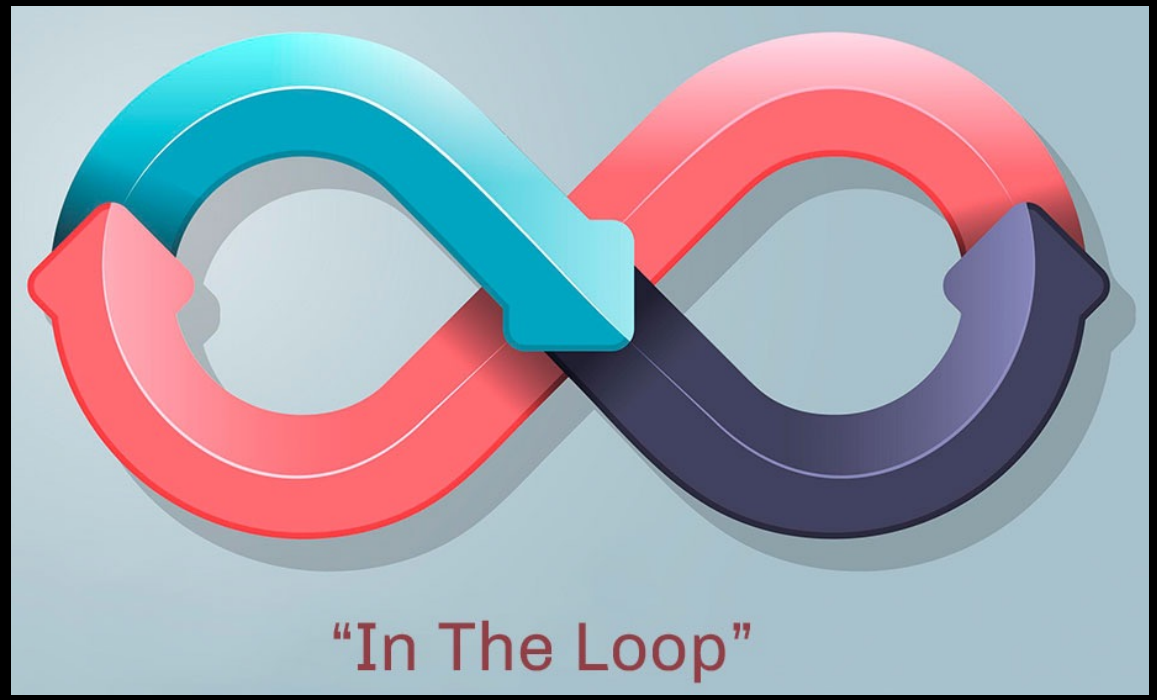
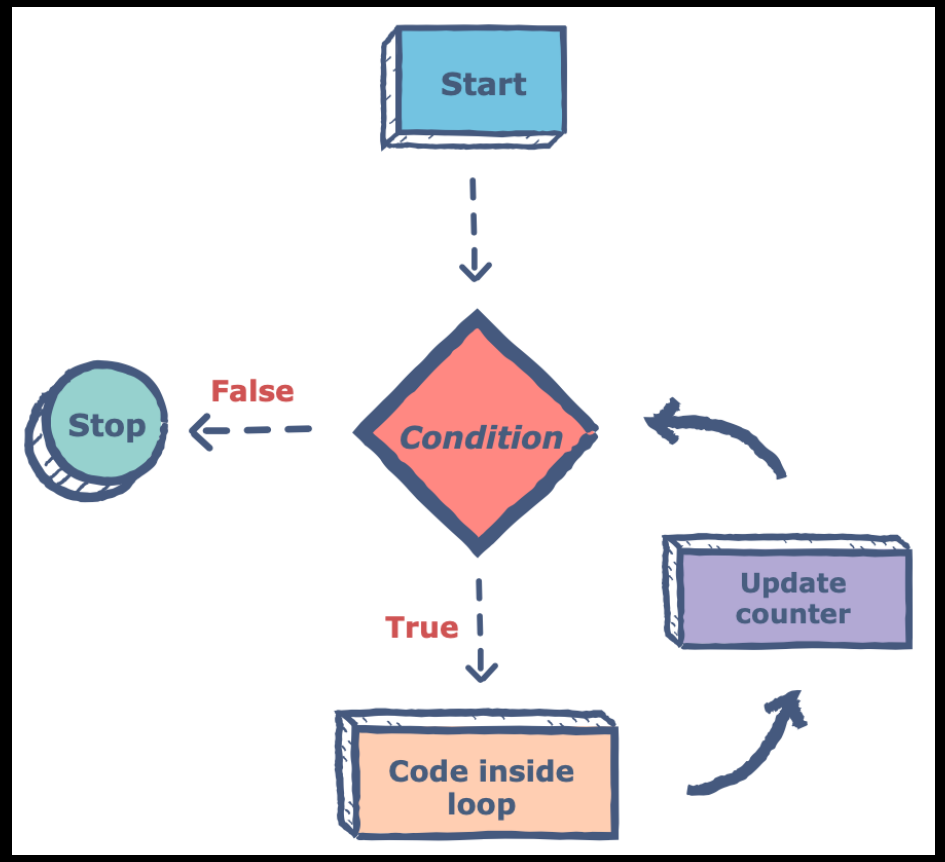
Iteration & Loop Control Structure

- Need of loops
- While Loop
- Do-while Loop
- For Loop
- Break statement
- Continue statement
- Odd Loop
- Infinite Loop
- Accumulator Pattern

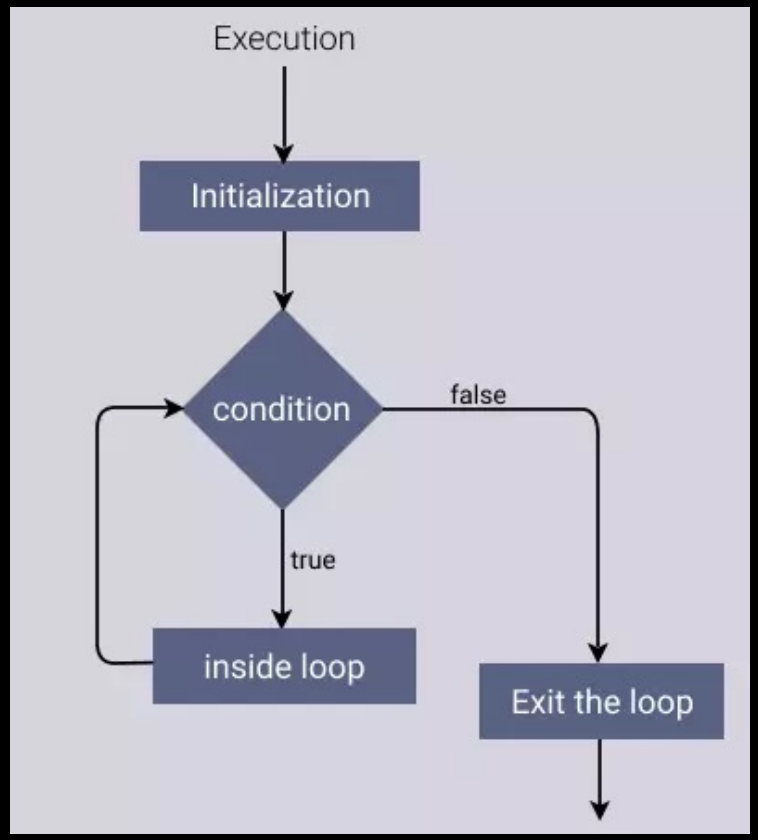


What is a Loop?



1. Code that runs multiple times based on a condition.
2. Loops also alter the flow of execution, similar to functions.
   - Functions: Reusable blocks of code.
   - Loops: Repeated execution of code.
3. Loops automate repetitive tasks.
4. Types of Loops: for, while, do-while.
5. Iterations: Number of times the loop runs.

While Loop




```
while (condition) {
  // Body of the loop
}
```

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

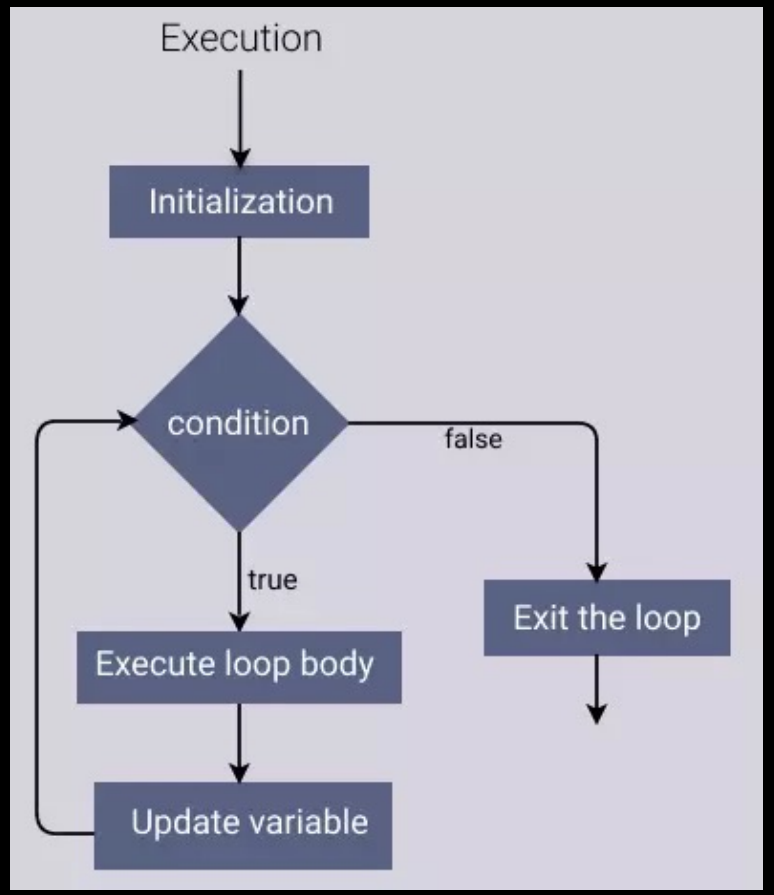
1. Iterations: Number of times the loop runs.
2. Used for non-standard conditions.
3. Repeating a block of code while a condition is true.
4. Remember: Always include an update to avoid infinite loops.

Practice Exercise

While Loop

- Develop a program using while that prints the multiplication table for a given number.
- Create a program to sum all odd numbers from 1 to a specified number N.
- Write a function that calculates the factorial of a given number.
- Create a program that computes the sum of the digits of an integer.
- Create a program to find the Least Common Multiple (LCM) of two numbers.
- Create a program to find the Greatest Common Divisor (GCD) of two integers.
- Create a program to check whether a given number is prime using while.

For Loop




```
for (initialisation; condition; update) {
  // Body of the loop
}
```

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

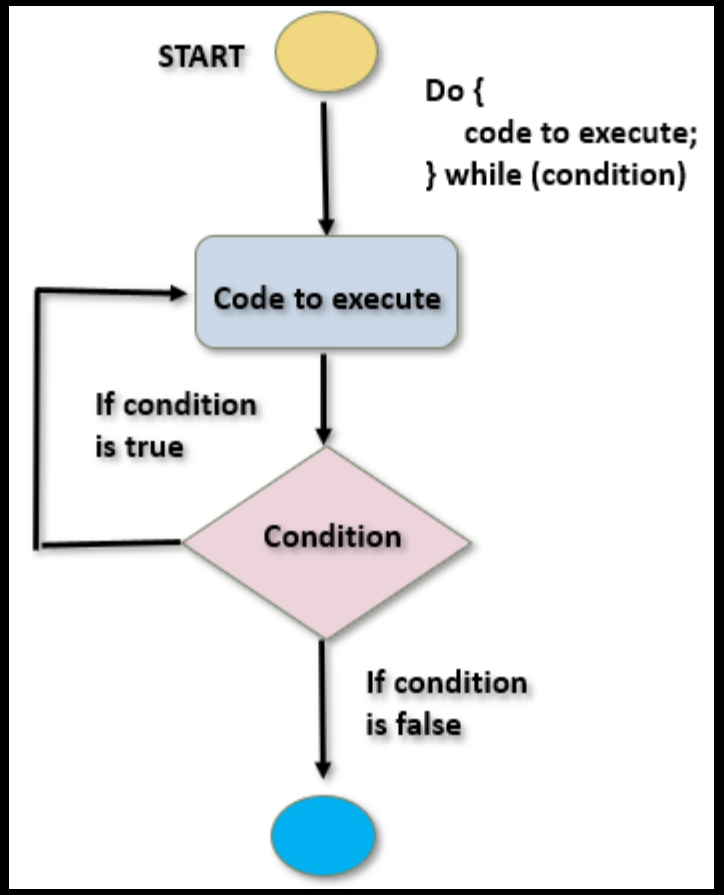
1. Standard loop for running code multiple times.
2. Generally preferred for counting iterations.

Practice Exercise

For Loop

- Create a program to reverse the digits of a number.
- Create a program to print the Fibonacci series up to a certain number.
- Create a program to check if a number is an Armstrong number.
- Create a program to verify if a number is a palindrome.
- Create a program using for loop multiplication table for a number.
- Create a program using for to display if a number is prime or not.
- Create a program that print patterns:

Do While Loop




```
do {
  // Body of the loop
}
while (condition);
```

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

1. Executes block first, then checks condition.
2. Guaranteed to run at least one iteration.
3. Unlike while, first iteration is unconditional.
4. Don't forget to update condition to avoid infinite loops.

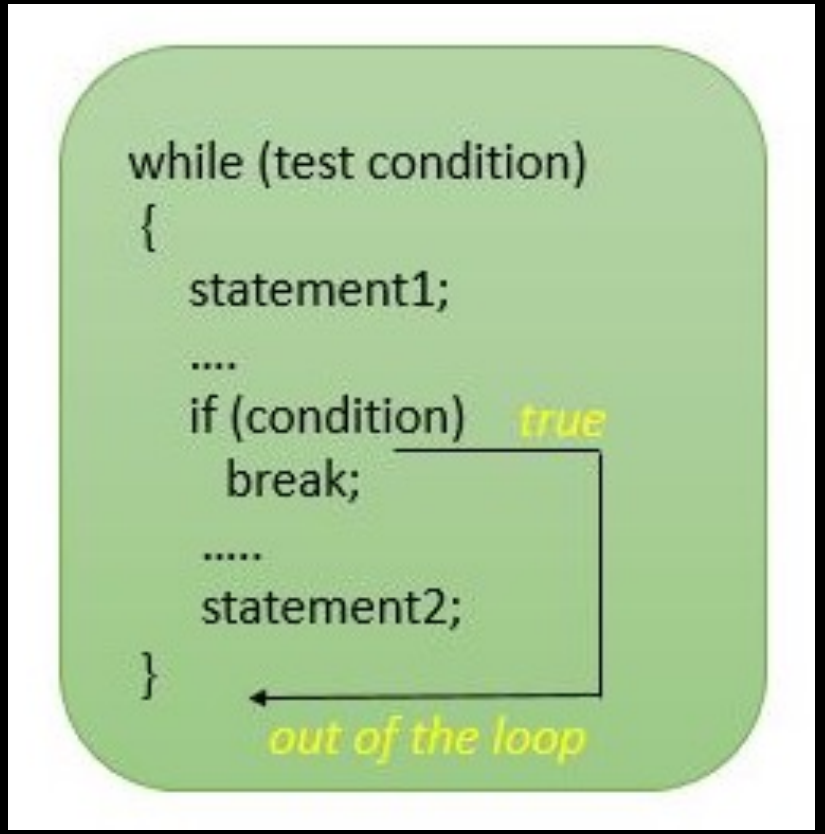
Practice Exercise

Do-While Loop

- Create a program that prompts the user to enter a positive number. Use a do-while loop to keep asking for the number until the user enters a valid positive number.
- Develop a program that calculates the sum of all numbers entered by a user until the user enters 0. The total sum should then be displayed.

Break statement

1. Break lets you stop a loop early, or break out of a loop
2. Exits Loops: Ends for, while, do-while loops early.
3. Ends Switch Cases: Prevents fall-through in switch cases.
4. Immediate Effect: Immediately leaves the loop/switch.
5. Controls Flow: Alters program flow for efficiency.
6. Use Wisely: Important for readability.

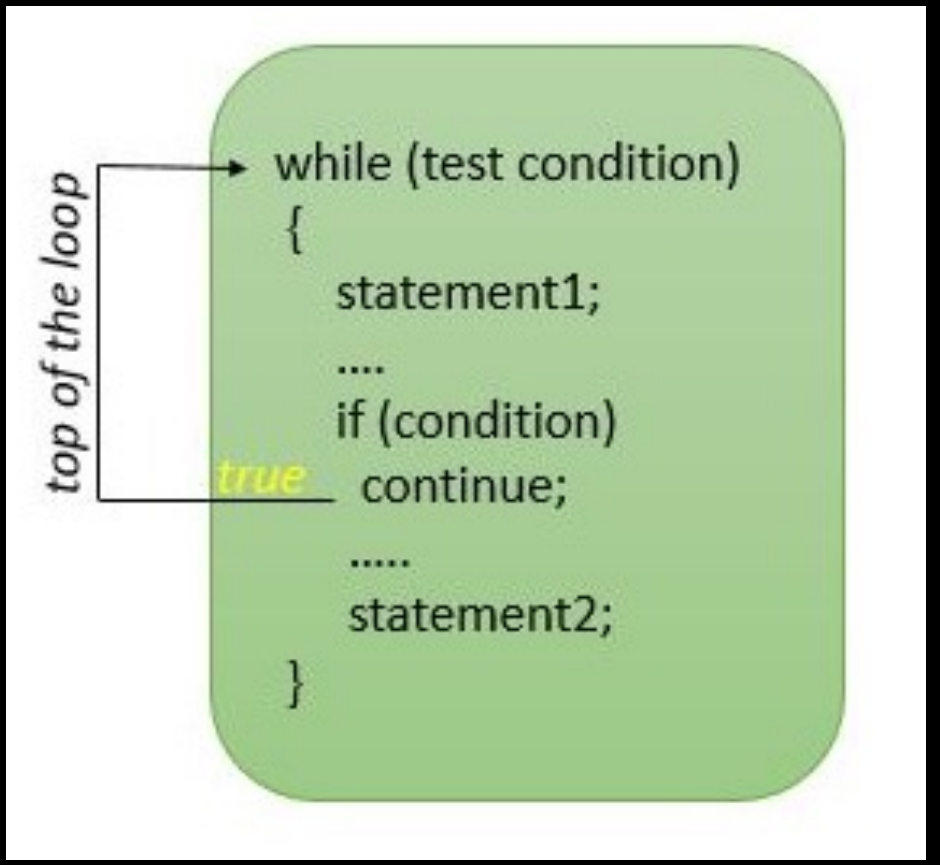


Break statement

```
for (let i = 0; i < 10; i++) {
  if (i === 5) {
    break; // Exit the loop when i is 5
  }
  console.log(i);
}
// Output: 0, 1, 2, 3, 4
```

Continue statement

1. Continue is used to skip one iteration or the current iteration
2. Next Iteration: Immediately starts the next iteration of the loop.
3. In while loop remember to do the increment manually before using continue.
4. Used in Loops: Works within for, while, do-while loops.
5. Not for switch: Unlike break, not used in switch statements.
6. Improves Logic: Helps in avoiding nested conditions within loops.



Continue statement

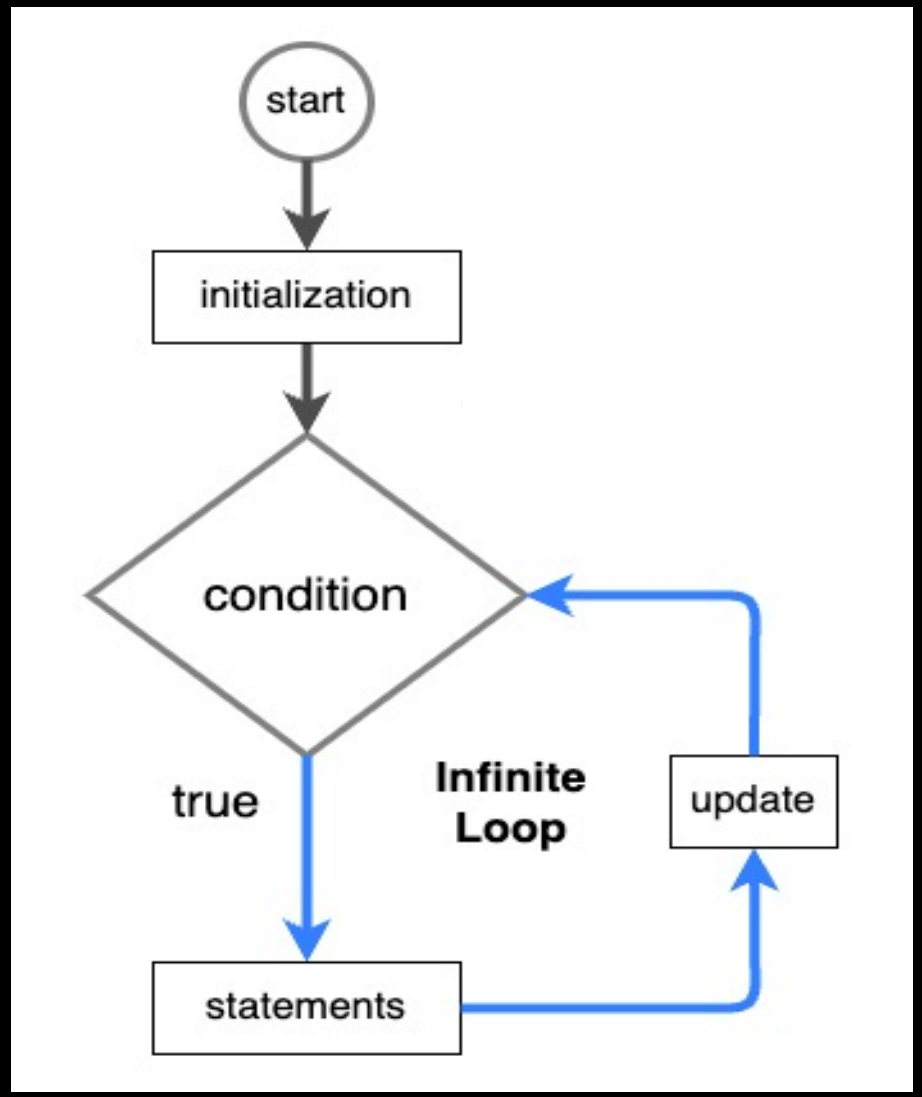
```
for (let i = 0; i < 10; i++) {
    if (i === 5) {
        continue; // Skip the iteration when i is 5
    }
    console.log(i);
}
// Output: 0, 1, 2, 3, 4, 6, 7, 8, 9
```

Odd Loop

```
let another, num;
do {
    num = parseInt(prompt("Enter a number: "), 10);
    console.log(`Square of ${num} is ${num * num}`);
    another = prompt("Want to enter another number (y/n)? ");
} while (another === 'y');
```

1. Condition-Driven: Run until a specific condition is fulfilled.
2. While and Do-While: Commonly used for indeterminate iterations.
3. Dynamic Iteration: Iterations depend on changing conditions or input.
4. Break Usage: May use break for exit in any loop type.
5. Practical Use: Ideal for processing with unknown completion point.
6. Design Carefully: Important to avoid infinite loops by ensuring a valid exit condition.

Infinite Loop




```
let i = 1;
while (true) {
    console.log(i);
    i++;
}
```

1. Endless Execution: They run continuously.
2. Purposeful or Accidental: Used deliberately or by mistake.
3. Exit Strategy: Require break or similar statements for stopping.
4. Resource Intensive: May cause high CPU usage.

Accumulator Pattern

1. A pattern to accumulate values through looping.
2. Common Scenarios:
   - Sum all the numbers.
   - Create a modified copy of an array.

Accumulator Pattern

```
let sum = 0;
let input;

do {
    input = parseInt(prompt("Enter a number (or type 'stop' to
    finish): "), 10);
    if (!isNaN(input)) {
        sum += input;
    }
} while (!isNaN(input));

console.log("Total Sum:", sum);
```

Practice Exercise

Loops

- Create a program using continue to sum all positive numbers entered by the user; skip any negative numbers.
- Create a program using continue to print only even numbers using continue for odd numbers.
- Write a program that continuously reads integers from the user and prints their squares. Use an infinite loop and a break statement to exit when a special number (e.g., -1) is entered.