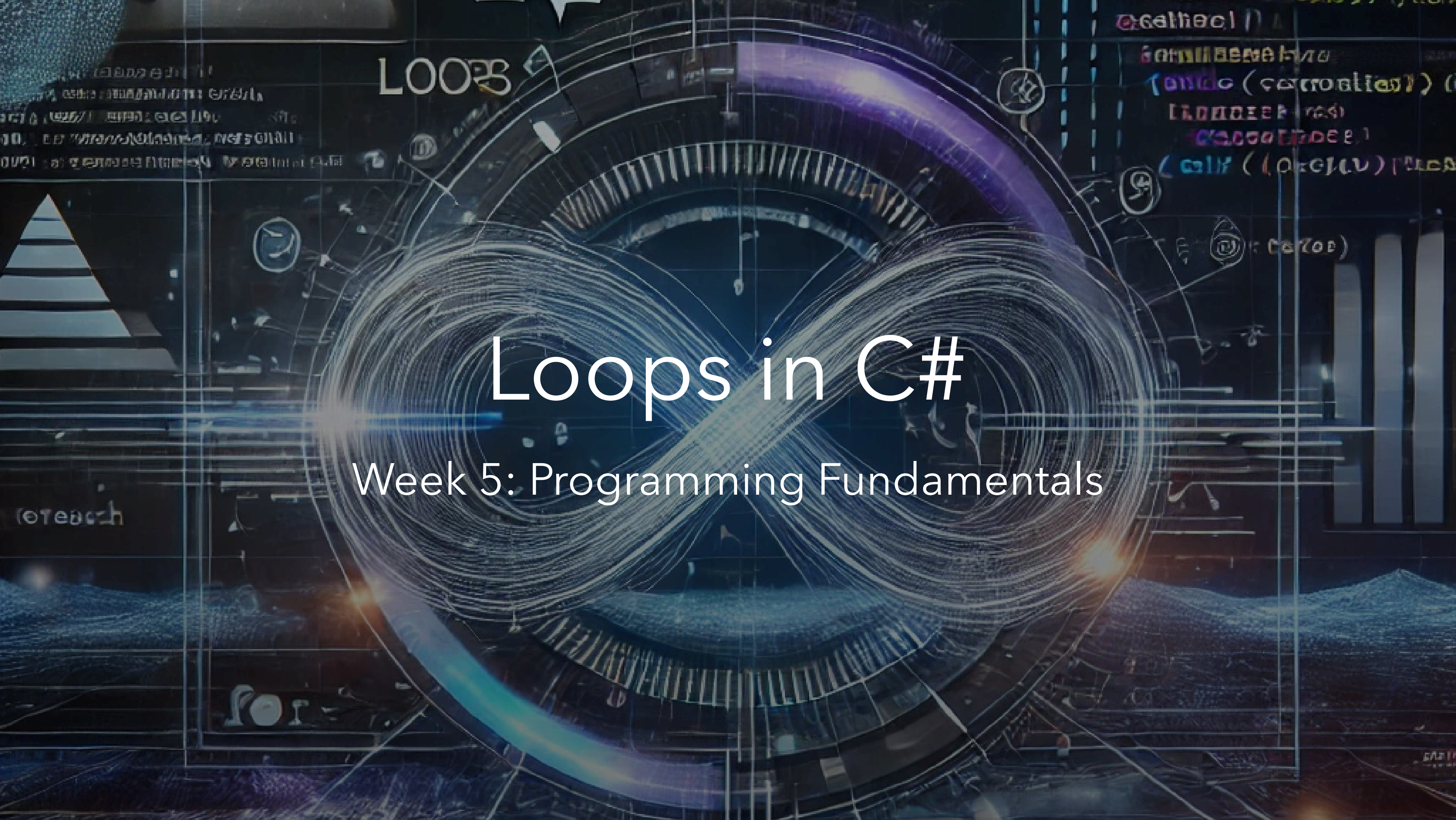

Loops in C#

Week 5: Programming Fundamentals

Today’s Agenda

- Review: Why Do We Need Loops?
- Types of Loops in C#
    - for loops
    - while loops
    - do-while loops
- Loop Control Mechanisms
- Practical Examples
- Common Pitfalls & Best Practices

Why Do We Need Loops?

- Loops allow us to **automate repetitive tasks**
- They help us process collections of data efficiently
- Essential for working with arrays, lists, and other data structures
- Save time and reduce code duplication

```
// Without loops:
Console.WriteLine("Count: 1");
Console.WriteLine("Count: 2");
Console.WriteLine("Count: 3");
Console.WriteLine("Count: 4");
Console.WriteLine("Count: 5");

// With loops:
for (int i = 1; i < = 5; i++) {
    Console.WriteLine($"Count: {i}");
}
```

Types of Loops in C#

C# provides three main types of loops:

For Loop

- Best when you know number of iterations
- Compact syntax

While Loop

- Continues while condition is true
- Checks condition before execution

Do-While Loop

- Similar to while loop
- Always executes at least once

For Loop

Syntax:

```
for (initializer; condition; iterator) {
    // code to repeat
}
```

Components:

- **Initializer**: Runs once at beginning
- **Condition**: Checked before each iteration
- **Iterator**: Runs after each iteration

Example:

```
// Count from 1 to 5
for (int i = 1; i < = 5; i++) {
    Console.WriteLine(i);
}

// Output:
// 1
// 2
// 3
// 4
// 5
```

For Loop Variations

Decreasing Counter:

```
for (int i = 10; i > 0; i--) {
    Console.WriteLine(i);
}
```

Multiple Variables:

```
for (int i = 0, j = 10; i < j; i++, j--) {
    Console.WriteLine($"i = {i}, j = {j}");
}
```

Skip Iterations:

```
// Print even numbers from 2 to 10
for (int i = 2; i < = 10; i += 2) {
    Console.WriteLine(i);
}
```

Empty Parts:

```
int i = 0;
for (; i < 5;) {
    Console.WriteLine(i);
    i++;
}
```

While Loop

Syntax:

```
while (condition) {
    // code to repeat
}
```

Characteristics:

- Checks condition before executing code
- If initial condition is false, code never executes
- Best when number of iterations is unknown

Example:

```
int count = 1;

while (count < = 5) {
    Console.WriteLine(count);
    count++;
}

// Output:
// 1
// 2
// 3
// 4
// 5
```

Do-While Loop

Syntax:

```
do {
    // code to repeat
} while (condition);
```

Characteristics:

- Executes code first, then checks condition
- Always runs at least once
- Good for validation loops

Example:

```
int count = 1;

do {
    Console.WriteLine(count);
    count++;
} while (count < = 5);

// Output:
// 1
// 2
// 3
// 4
// 5
```

When to Use Each Loop Type?

For Loop

- Known number of iterations
- Array/list traversal
- Increment/decrement patterns

While Loop

- Unknown iteration count
- Condition depends on external factors
- Early exit conditions

Do-While Loop

- Input validation
- User confirmation
- When code must execute at least once

```
// For: Iterate through array
for (int i = 0; i < array.Length; i++) { }

// While: Read until end of file
while (!reader.EndOfStream) { }

// Do-While: Validate user input
do {
    input = Console.ReadLine();
} while (!IsValid(input));
```

Loop Control Mechanisms

Break Statement

Exits the loop immediately

```
for (int i = 1; i < = 10; i++) {
    if (i = = 5) break;
    Console.WriteLine(i); // Prints 1,2,3,4
}
```

Continue Statement

Skips current iteration, continues to next

```
for (int i = 1; i < = 5; i++) {
    if (i = = 3) continue;
    Console.WriteLine(i); // Prints 1,2,4,5
}
```

Nested Loops

Loop inside another loop

```
for (int i = 1; i < = 3; i++) {
    for (int j = 1; j < = 3; j++) {
        Console.WriteLine($"{i},{j}");
    }
}
```

Early Exit Conditions

Complex conditions to exit loops

```
while (condition1) {
    if (condition2) break;
    // process data
}
```

Common Pitfalls to Avoid

Infinite Loops

```
// Infinite loop - no update to i
while (i < 10) {
    Console.WriteLine(i);
    // Missing i++
}
```

Off-by-One Errors

```
// Prints 0-9 (not 1-10)
for (int i = 0; i < 10; i++) {
    Console.WriteLine(i);
}
```

Improper Iterator Updates

```
for (int i = 0; i < 10; i++) {
    // Dangerous - modifying i inside loop
    if (condition) i += 2;
}
```

Forgetting Break in Switch

```
while (true) {
    switch (input) {
        case "exit":
            // Missing 'break' will cause issues
    }
}
```

AI-Assisted Coding Demo

Let’s ask an AI to write a program that:

1. Generates 10 random numbers between 1 and 100
2. Uses different loop types to:
    - Calculate the sum (for loop)
    - Find the maximum value (while loop)
    - Print all values above average (do-while)

Guided Coding Session

Let’s build a number guessing game together:

1. Program generates a random number from 1 to 100
2. User gets multiple attempts to guess
3. Program provides "higher" or "lower" hints
4. Track and display number of attempts

Your Turn: Coding Challenge

Write a program that:

1. Asks the user to enter a positive integer
2. Use loops to display the multiplication table for that number (1-10)
3. Add validation to ensure user enters a valid number
4. BONUS: Ask if user wants to see another table when done

Debugging Loop Issues

Common loop debugging techniques:

- Print counter variables to track iterations
- Use step-through debugging in Visual Studio
- Check boundary conditions (first/last iterations)
- Verify loop conditions and updates

Key Takeaways

- Choose the right loop type for your specific situation
- Be careful with loop conditions to avoid infinite loops
- Use control statements (break/continue) when needed
- Always ensure your loop variables are properly updated
- Consider readability and efficiency when writing loops

Next Week Preview

Next week, we’ll explore:

- Functions & Methods in C#
- AI-Assisted Code Generation
- How to make your code more modular and reusable
- Best practices for function design

Questions?