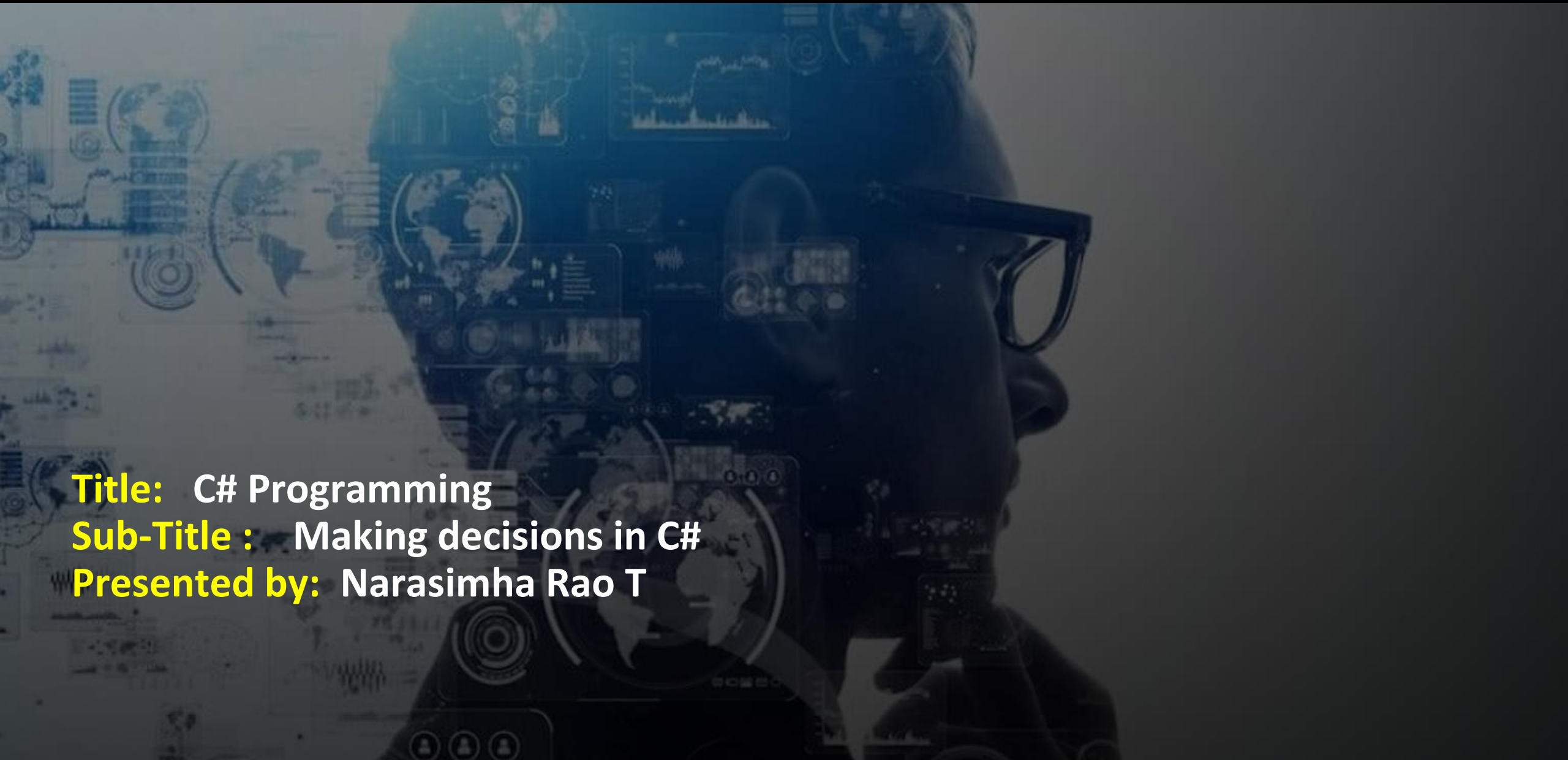
upGrad ENTERPRISE

.NET FSD Bootcamp

14th July, 2025

upGrad ENTERPRISE



Title: C# Programming

Sub-Title : Making decisions in C#

Presented by: Narasimha Rao T

upGrad ENTERPRISE

Making Decisions in C#

By
Narasimha Rao T
Microsoft.Net FSD Trainer
Professional Development Trainer
tnrao.trainer@gmail.com

Day-4

Index

Making Decisions in C#

1. if, else if, else, nested conditions,
2. switch-case, fallthrough, pattern matching basics,
3. Loops: for, while, do-while, foreach,
4. break, continue, flow control,
5. pattern generation (triangles, pyramids, etc.)

Introduction to Control Structures

Control structures determine the flow of execution in a program. They allow your program to make **decisions**, **repeat tasks**, and **handle different conditions dynamically**.

Types of control structures in C#:

- **Conditional Statements**: if, else if, else, switch
- **Looping Constructs**: for, while, do-while, foreach
- **Jump Statements**: break, continue, return

Note: These tools are essential to building logic in real-world programs — from handling user input to processing data conditionally.

1. if, else if, else

Syntax:

```
if (condition) {
    // code block if condition is true
} else if (anotherCondition) {
    // code block if anotherCondition is true
} else {
    // code block if none of the above conditions are true
}
```

Example:

```
int num = 10;

if (num > 0) {
    Console.WriteLine("Positive");
} else if (num == 0) {
    Console.WriteLine("Zero");
} else {
    Console.WriteLine("Negative");
}
```

2. Nested Conditions

Definition: One conditional statement inside another.

Example:

```
int age = 25;
bool hasID = true;

if (hasID) {
    if (age >= 18) {
        Console.WriteLine("Access granted.");
    } else {
        Console.WriteLine("Underage.");
    }
} else {
    Console.WriteLine("No ID provided.");
}
```

3. switch-case

Syntax:

```
switch (variable) {
    case value1:
        // code
        break;
    case value2:
        // code
        break;
    default:
        // code
        break;
}
```

Example:

```
int day = 3;

switch (day) {
    case 1:
        Console.WriteLine("Monday");
        break;
    case 2:
        Console.WriteLine("Tuesday");
        break;
    case 3:
        Console.WriteLine("Wednesday");
        break;
    default:
        Console.WriteLine("Another day");
        break;
}
```

4. Fallthrough (C# specific)

- Unlike C or Java, **C# does NOT allow fallthrough** by default between `case` labels.
- Each case must end with `break`, `return`, or `goto`.

Allowed Fallthrough Example using `goto`:

```
int value = 1;

switch (value) {
    case 1:
        Console.WriteLine("One");
        goto case 2; // fallthrough
    case 2:
        Console.WriteLine("Two");
        break;
}
```

5. Pattern Matching (Basics)

Used with `switch`, `is`, and `when`.

Example: Type pattern:

```
object obj = "hello";

if (obj is string s) {
    Console.WriteLine($"String length: {s.Length}");
}
```

Example: switch pattern matching:

```
object data = 3.14;

switch (data) {
    case int i:
        Console.WriteLine($"Integer: {i}");
        break;
    case double d when d > 3:
        Console.WriteLine("Double > 3");
        break;
    default:
        Console.WriteLine("Other type");
        break;
}
```

Introduction to Loops

- Loops allow your program to repeat a block of code multiple times based on a condition.
- They are a core part of automating repetitive tasks in C#.
- Types of loops in C#:
    - **for** – when the number of iterations is known
    - **while** – when looping is based on a condition
    - **do-while** – similar to while, but guarantees at least one execution
    - **foreach** – best for iterating over collections or arrays

- **Loops help minimize code duplication and handle dynamic input sizes.**

Loops

6. for Loop

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

7. while Loop

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

8. do-while Loop

Executes at least once.

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

9. foreach Loop

Used for collections.

```
int[] numbers = { 1, 2, 3 };

foreach (int n in numbers) {
    Console.WriteLine(n);
}
```

10. break and continue

`break` : Exits the loop

`continue` : Skips to the next iteration

```
for (int i = 0; i < 10; i++) {
    if (i == 5) break;
    if (i % 2 == 0) continue;
    Console.WriteLine(i); // prints odd numbers until 5
}
```

11. Flow Control Summary

Statement	Description
`if/else`	Decision-making
`switch`	Multi-way branching
`for`	Count-controlled loop
`while`	Condition-controlled loop
`do-while`	Post-condition loop
`break`	Exit loop/switch
`continue`	Skip iteration
`goto`	Jump (use sparingly)

12. Pattern Generation (Triangles, Pyramids)

Right-Angled Triangle

```
int rows = 5;

for (int i = 1; i <= rows; i++) {
    for (int j = 1; j <= i; j++) {
        Console.Write("*");
    }
    Console.WriteLine();
}
```

Inverted Triangle

```
for (int i = 5; i >= 1; i--) {
    for (int j = 1; j <= i; j++) {
        Console.Write("*");
    }
    Console.WriteLine();
}
```

Pyramid Pattern

```
int rows = 5;

for (int i = 1; i <= rows; i++) {
    for (int j = i; j < rows; j++) {
        Console.Write(" ");
    }
    for (int k = 1; k <= (2 * i - 1); k++) {
        Console.Write("*");
    }
    Console.WriteLine();
}
```

upGrad ENTERPRISE

Q & A

Narasimha Rao T
tnrao.trainer@gmail.com