C# is a object-oriented programming language developed by Microsoft as part of the .NET framework. It is widely used for building desktop applications, web applications, games, and enterprise software. Here are the basics of C#:

1. Syntax & Structure

- Case-sensitive language.
- Statements end with a semicolon [;].
- Uses curly braces {} for defining blocks of code.
- using System → Allows access to built-in .NET libraries.
- Name space → Organizes code into containers for classes and other namespaces.
- Main Method → Entry point of a C# program
- class → Defines a blueprint for objects and holds data and methods

2. Variables & Data Types

C# is a strongly-typed language, meaning every variable must have a type.

Common data types:

- int – Integer numbers (e.g, 10, -5)
- double – Floating point numbers (e.g, 3.14)
- char – A single character (e.g, 'A')
- string – Text (e.g, "Hello")
  - Supports **concatenation** (+ operator).
  - **Interpolation** ($"Hello {name}").
- bool – Boolean values (e.g, true or false)
  - Used in logical operations (&&, ||, !).
  - Can be compared (==, !=, >, <, >=, <=).
  - 0 is treated as false, any other number is true.

Example:

```
int age = 25;
double price = 9.99;
char grade = 'A';
string name = "John";
bool isActive = true;
```

3. Operators

Arithmetic Operators

+, -, *, /, %

Assignment Operators

=, +=, -=, *=, /=, %=

Comparison Operators

Used in conditions, return `true` or `false`: ==, !=, >, <, >=, <=

4. Control Flow Statements

Conditional Statements

- if statement: Executes code if a condition is true.
- if-else statement: Provides an alternative execution path

```
int num = 10;
if (num > 0)
{
    Console.WriteLine("Positive number");
}
else
{
    Console.WriteLine("Negative number");
}
```

- switch (alternative to multiple if-else statements)

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

Loops

- for loop → Repeats code based on a condition.

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

- while loop → Runs as long as the condition is true.

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

- Do-While Loop
  Unlike a while loop, a do-while loop **executes at least once**, even if the condition is false.
  ```
  int i = 0;
  do
  {
      Console.WriteLine(i);
      i++;
  } while (i < 3);
  ```

- foreach loop (for iterating collections)

  without needing an index, It works with **arrays, lists, and other collections**

```
string[] fruits = { "Apple", "Banana", "Cherry" };
foreach (string fruit in fruits)
{
    Console.WriteLine(fruit);
}
```

5. Collections & Arrays

- Arrays store multiple values of the same type.
- Indexed starting from 0.

```
string[] cars = {"Volvo", "BMW", "Ford"};
Console.WriteLine(cars[0]); // Outputs: Volvo
```

Lists allow dynamic sizing.

```
List<string> names = new List<string>();
names.Add("Alice");
names.Add("Bob");
Console.WriteLine(names[1]); // Output: Bob
```

6. Parsing

Parsing is converting **string** data to other types.

- int.Parse("123") → Converts "123" to integer 123.
- double.Parse("3.14") → Converts "3.14" to double.

7. Ternary Operator

shorthand way of writing an if-else statement in a **single line** and It takes three operands:

condition ? value_if_true : value_if_false;

Ex:

```
int number = 10;
string result = (number > 5) ? "Greater" : "Smaller";
Console.WriteLine(result); // Output: Greater
```

8. Methods (Functions)

Methods allow code reuse, by allowing you to define a block of code once and call it multiple times.

```
void SayHello()
{
   Console.WriteLine("Hello!");
}
```

```
SayHello();
```

With parameters:

```
int Add(int a, int b)
{
    return a + b;
}

int sum = Add(5, 3); // Returns 8
```