C# Beginner's Guide

C# is a modern, object-oriented programming language developed by Microsoft. It is part of the .NET ecosystem and is widely used for building applications, especially for Windows. Below is an overview of essential C# topics with examples.

---

1. History

- **C#** was developed by Microsoft in the early 2000s as part of the .NET framework, and it has since evolved into a versatile language for Windows applications, mobile apps (via Xamarin), and web development.

---

2. Environment

C# is primarily used in the **.NET environment**, which provides libraries and frameworks for software development. Visual Studio is the most popular IDE for C# development.

---

3. Program Structure

A C# program typically consists of namespaces, classes, methods, and code logic.

```
using System;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello, World!");
    }
}
```

---

4. Data Types

C# is strongly typed, meaning every variable has a defined data type. Common data types include:

- **int**: Integer

- **double**: Floating point
- **char**: Single character
- **bool**: True/False
- **string**: A sequence of characters

Example:

```
int age = 25;
double price = 19.99;
char grade = 'A';
bool isActive = true;
string name = "John Doe";
```

---

5. Type Conversions

Type conversion is used to change a value from one type to another.

- **Implicit Conversion**: Automatically done by the compiler (e.g., from `int` to `double`).
- **Explicit Conversion**: Requires a cast.

```
int i = 10;
double d = i;  // Implicit
int i2 = (int) d;  // Explicit
```

---

6. Identifiers

Identifiers are the names you give to variables, methods, classes, etc. They must begin with a letter or an underscore and can be followed by letters, digits, or underscores.

---

7. Variables

Variables store data. Each variable must be declared with a type.

```
int number = 5;
string message = "Hello, C#!";
```

---

8. Constants

Constants are values that cannot be changed after initialization.

```
const double Pi = 3.14159;
```

---

9. Operators

C# uses operators for arithmetic, comparison, logical, and assignment operations.

```
int x = 10;
int y = 5;
int sum = x + y;  // Addition
bool isEqual = (x == y);  // Comparison
bool isTrue = (x > 0 && y < 10);  // Logical
```

---

10. Decision Making

Decision-making statements such as `if`, `else if`, `else`, and `switch` allow you to control program flow based on conditions.

```
int age = 18;
if (age >= 18)
{
    Console.WriteLine("Adult");
}
else
{
    Console.WriteLine("Minor");
}
```

---

11. Loops

Loops allow you to repeat a block of code.

- **for loop**
- **while loop**
- **foreach loop**

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

---

12. Methods

Methods are blocks of code that perform a task and can return a value.

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

---

13. Nullables

Nullable types allow variables to store null values in addition to their regular type values.

```
int? nullableInt = null;
```

---

14. Array

Arrays are used to store multiple values in a single variable.

```
int[] numbers = {1, 2, 3, 4, 5};
```

---

15. Strings

Strings represent text and are immutable in C#.

```
string message = "Hello, C#!";
Console.WriteLine(message.Length); // String length
```

---

16. Structure

Structures are value types that can store data and methods.

```
struct Person
{
   public string Name;
```

```
    public int Age;
}
```

---

17. Enum

Enums are used to define named constants.

```
enum Day { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday }
Day today = Day.Monday;
```

---

18. Classes

Classes define the blueprint for objects and support methods, properties, fields, and events.

```
class Car
{
    public string Model;
    public void Drive()
    {
        Console.WriteLine("Driving");
    }
}
```

---

19. Interfaces

Interfaces define contracts for classes to implement.

```
interface IDriveable
{
    void Drive();
}

class Car : IDriveable
{
    public void Drive()
    {
        Console.WriteLine("Car is driving");
    }
}
```

---

20. Namespace

Namespaces organize code into logical groups.

```
namespace MyApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello from MyApplication!");
        }
    }
}
```

---

21. Preprocessor Directives

Preprocessor directives help to control the compilation process, such as conditional compilation.

```
#define DEBUG
using System;

class Program
{
    static void Main()
    {
#if DEBUG
        Console.WriteLine("Debug Mode");
#endif
    }
}
```

---

22. Exception Handling

Exception handling is done using `try`, `catch`, `finally` blocks.

```
try
{
    int result = 10 / 0;
}
```

```
catch (DivideByZeroException ex)
{
    Console.WriteLine("Error: " + ex.Message);
}
finally
{
    Console.WriteLine("This is always executed.");
}
```

---

23. Collections

Collections such as `List`, `Dictionary`, `Queue`, and `Stack` are used to store multiple items.

```
List<int> numbers = new List<int> {1, 2, 3};
numbers.Add(4);
```

---

24. Generics

Generics allow you to define classes, methods, and interfaces with a placeholder type.

```
class GenericClass<T>
{
    public T Value { get; set; }
}

GenericClass<int> obj = new GenericClass<int>();
obj.Value = 42;
```

---

25. Anonymous Methods

Anonymous methods are defined without a name.

```
Action greet = delegate { Console.WriteLine("Hello!"); };
greet();
```

---

26. Multithreading

Multithreading allows you to execute multiple threads in parallel for better performance.

```
class Program
{
    static void Main()
    {
        Thread thread1 = new Thread(() => Console.WriteLine("Thread 1"));
        thread1.Start();
    }
}
```

OOP Concepts in C#

1. **Encapsulation**: Bundling data and methods that operate on that data within a class.
2. **Abstraction**: Hiding complex implementation and showing only the necessary details.
3. **Inheritance**: Allowing one class to inherit the properties and methods of another class.
4. **Polymorphism**: Allowing different classes to be treated as instances of the same class through inheritance.

```
// Example of Inheritance
class Animal
{
    public void Eat() { Console.WriteLine("Eating..."); }
}

class Dog : Animal
{
    public void Bark() { Console.WriteLine("Barking..."); }
}

class Program
{
    static void Main()
    {
        Dog dog = new Dog();
        dog.Eat();
        dog.Bark();
    }
}
```

This guide covers essential C# concepts for beginners. As you grow in C#, you can dive deeper into more advanced topics like LINQ, reflection, and async programming. Happy coding!