Basics of C# Programming

Mirolim Majidov

1. C# Language and .NET Platform

What is this?

C# (pronounced C sharp) is a modern, object-oriented programming and a strongly typed language developed by Microsoft. It is part of the .NET framework, providing a powerful platform for building Windows applications, web applications, and more.

How/where can I use this?

C# is widely used for developing various applications, including desktop applications, web applications (using ASP.NET), mobile applications (using Xamarin), cloud-based applications, and game development (using Unity).

2. Variables/Переменные

What is this?

Variables are containers for storing data values. In C#, you must declare a variable before using it, specifying its type.

How/where can I use this?

Variables are used to store and manipulate data in your program, making it dynamic and adaptable.

Example:

```
int age = 25; // Declaration and initialization of an integer variable
string name = "John"; // Declaration and initialization of a string variable
```

3. Data Types

What is this?

Data types define the type of data a variable can hold. C# supports various data types, including int, double, string, etc.

How/where can I use this?

Choosing data types is crucial for efficient memory usage and accurate data representation in your program.

Example:

```
int integerNumber = 42;
double floatingNumber = 3.14;
string text = "Hello, C#!";
```

4. Static variables and Constants

What is this?

Static variables are shared among all instances of a class, while constants are values that cannot be changed.

How/where can I use this?

Use static variables for data shared across instances, and constants for values that should not be modified.

Example:

```
public static int staticVariable = 10;
public const double PI = 3.14159;
```

5. Console Input/Output

What is this?

Console input/output functions allow interaction with the user through the console.

How/where can I use this?

Useful for debugging, simple user interfaces, and text-based applications.

Example:

```
Console.WriteLine("Enter your name:");
string userName = Console.ReadLine();
Console.WriteLine($"Hello, {userName}!");
```

6. Arrays/Массивы

What is this?

Arrays are used to store multiple values of the same type under a single variable name.

How/where can I use this?

Useful when dealing with collections of data, like lists of numbers, names, etc.

Example:

```
string[] names = { "Jake", "James", "Alisa" };
names[1] = "Jamshed";

int[] numbers = { 1, 2, 3, 4, 5 };
Console.WriteLine(numbers[2]); // Accessing the third element
```

7. Arithmetic/Comparison Operators

What is this?

Operators perform operations on variables and values. Arithmetic operators handle mathematical operations, while comparison operators compare values.

How/where can I use this?

Basically, for mathematical calculations and conditional statements.

Example:

```
int a = 10;
int b = 4;

//Arithmetic Operators:
int sum = a + b; // Addition
int difference = a - b; // Subtraction
int product = a * b; // Multiplication
int quotient = a / b; // Division
int remainder = a % b; // Modulus
```

```
int x = 10;
int y = 4;

//Comparison Operators:
bool isEqual = (x == y); // Equal to
bool isNotEqual = (x != y); // Not equal to
bool isGreaterThan = (x > y); // Greater than
bool isLessThan = (x < y); // Less than
bool isGreaterOrEqual = (x >= y); // Greater than or equal to
bool isLessOrEqual = (x <= y); // Less than or equal to
```

8. If...Else and Switch Case

What is this?

The **if...else** statement allows the program to make decisions based on a given condition. If the condition is true, the code inside the 'if' block is executed; otherwise, the code inside the 'else' block is executed.

The **switch case** statement is another way to handle multiple conditions. It compares the value of an expression against possible constant values and executes the code block associated with the first matching value.

How/where can I use this?

Use **if...else** statements to create branching logic based on specific conditions.

Use **switch case** when you have a variable with discrete values and want to execute different blocks of code based on its value.

Examples of If...Else and Switch Case

```
//If-Else
int number = 7;

if (number > 0)
{
    Console.WriteLine("Less than zero");
}
else if (number < 0)
{
    Console.WriteLine("Greater than zero");
}
else
{
    Console.WriteLine("Equal to zero");
}
```

```
//Switch-case
int gender = 3;
string genderName;

switch (gender)
{
    case 0:
        genderName = "Female";
        break;
    case 1:
        genderName = "Male";
        break;
    default:
        genderName = "Invalid gender";
        break;
}
```

9. Loops (For, Foreach, While, Do-While)

What is this?

Loops are structures that repeat a block of code until a specific condition is met.

How/where can I use this?

Useful for iterating through collections, performing repetitive tasks, and controlling program flow.

Example:

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

for (int i = 0; i < numbers.Length; i++)
{
    Console.WriteLine(numbers[i]);
}

foreach (var item in numbers)
{
    Console.WriteLine(item);
}
```

```
var counter = 0;
while (counter < numbers.Length)
{
    Console.WriteLine(numbers[counter]);
    counter++;
}

counter = 0;
do
{
    Console.WriteLine(numbers[counter]);
    counter++;
} while (counter < numbers.Length);
```

10. Methods

What is this?

Named blocks of code that can be executed by calling their name.

How/where can I use this?

Use methods to organize code into reusable and modular units.

Example:

```
PrintMessage("Nabijon");
var sum = Sum(5, 8);

void PrintMessage(string name)
{
    Console.WriteLine($"Hello, {name}!");
}

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

11. Method with Params Parameter

What is this?

A method that can accept a variable number of parameters.

How/where can I use this?

Use when you need flexibility in the number of arguments passed to a method.

Example:

```
var sum = Sum();
sum = Sum(1, 10, 15, 70);

int Sum(params int[]  numbers)
{
    var sum = 0;
    foreach (int number in numbers)
        sum += number;

    return sum;
}
```

12. Ref and Out Keywords in Methods

What is this?

Ref stands for "reference." When you pass a variable using ref to a method, you're passing a reference to the original variable, not a copy of its value.

Out is similar to ref, but it is designed for scenarios where the method is expected to assign a value to the parameter before it exits.

How/where can I use this?

Use **ref** when you want to modify the original value of a variable within a method, and you want the changes to be reflected outside the method.

Use **out** when you want the method to initialize the variable inside the method, and you don't require the variable to be initialized before passing it to the method.

Examples of Ref and Out keywords

```
int number = 5;
ModifyValue(ref number);
Console.WriteLine(number);
// Output: 10

void ModifyValue(ref int value)
{
    value = value * 2;
}
```

```
int a;
GetValues(out a, out int b);
Console.WriteLine($"a: {a}, b: {b}");
// Output: a: 10, b: 20

void GetValues(out int x, out int y)
{
    x = 10;
    y = 20;
}
```

13. Local and Recursive Methods

What is this?

Methods declared within another method (local) or methods calling themselves (recursive).

How/where can I use this?

Use local methods for encapsulation or recursive methods for solving problems iteratively.

Example:

```
// Local method
void OuterMethod()
{
    void InnerMethod()
    {
    }
}
```

```
// Recursive method
int Factorial(int n)
{
    if (n < 0)
        return 0;

    if (n == 0)
        return 1;

    return n * Factorial(n - 1);
}
```

14. Tuple and Enum

What is this?

Tuple: A lightweight data structure to group multiple values.

Enum: A set of named integer constants.

How/where can I use this?

Use tuples for returning multiple values from a method, and Enums for creating named sets of constants.

Example:

```
// Tuple
var numbers = (15, 47);
var names = Tuple.Create("Jamshed", "Jahongir");
var person = (Name: "John", Age: 25);
var name = person.Name;
var age = person.Age;
```

```
//Enum
Gender gender = Gender.Male;

2 references
enum Gender { Male, Female, Unknown };
```