BASICS OF C#

By:

Engr. Fatima Jaffar

C# Strings

- Strings are used for storing text.
- A string variable contains a collection of characters surrounded by double quotes:
- Example:

```
string greeting = "Hello";
```

String Length

- A string in C# is actually an object, which contain properties and methods that can perform certain operations on strings. For example, the length of a string can be found with the Length property:

```
string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
Console.WriteLine("The length of the txt string is: " + txt.Length);
```

Access Strings

- You can access the characters in a string by referring to its index number inside square brackets [].

```
string myString = "Hello";
Console.WriteLine(myString[0]);  // Outputs "H"
```

C# Booleans

C# If ... Else

- C# supports the usual logical conditions from mathematics:

- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b
- Equal to a == b
- Not Equal to: a != b

The if Statement

- Use the if statement to specify a block of C# code to be executed if a condition is True.
- Syntax:

```
if (condition)
{
  // block of code to be executed if the condition is True
}
```

The else Statement

- Use the else statement to specify a block of code to be executed if the condition is False.
- Syntax:

```
if (condition)
{
  // block of code to be executed if the condition is True
}
else
{
  // block of code to be executed if the condition is False
}
```

The else if Statement

- Use the else if statement to specify a new condition if the first condition is False.
- Syntax:

```
if (condition1)
{
  // block of code to be executed if condition1 is True
}
else if (condition2)
{
  // block of code to be executed if the condition1 is false and condition2 is True
}
else
{
  // block of code to be executed if the condition1 is false and condition2 is False
}
```

Short Hand If...Else (Ternary Operator)

```
variable = (condition) ? expressionTrue :  expressionFalse;
```

C# Switch Statements

```
switch(expression)
{
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
    break;
}
```

Loops

- Loops can execute a block of code as long as a specified condition is reached.

- Loops are handy because they save time, reduce errors, and they make code more readable

C# While Loop

- The while loop loops through a block of code as long as a specified condition is True:
- Syntax:

```
while (condition)
{
  // code block to be executed
}
```

The Do/While Loop

- The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.
- Syntax:

```
do
{
  // code block to be executed
}
while (condition);
```

C# For Loop

- When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop:
- Syntax:

```
for (statement 1; statement 2; statement 3)
{
  // code block to be executed
}
```

- Statement 1 is executed (one time) before the execution of the code block.
- Statement 2 defines the condition for executing the code block.
- Statement 3 is executed (every time) after the code block has been executed.

C# Foreach Loop

- There is also a foreach loop, which is used exclusively to loop through elements in an array (or other data sets):
- Syntax:

```
foreach (type variableName in arrayName)
{
  // code block to be executed
}
```

C# Break and Continue

C# Arrays

- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- To declare an array, define the variable type with square brackets:
- Syntax:

```
string[] cars;
```

Other Ways to Create an Array

```
// Create an array of four elements, and add values later
string[] cars = new string[4];

// Create an array of four elements and add values right away
string[] cars = new string[4] {"Volvo", "BMW", "Ford", "Mazda"};

// Create an array of four elements without specifying the size
string[] cars = new string[] {"Volvo", "BMW", "Ford", "Mazda"};

// Create an array of four elements, omitting the new keyword, and without specifying the size
string[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

Loop Through an Array

C# Multidimensional Arrays

- Two-Dimensional Arrays
- To create a 2D array, add each array within its own set of curly braces, and insert a comma (,) inside the square brackets:
- Example:

```
int[,] numbers = { {1, 4, 2}, {3, 6, 8} };
```

Access Elements of a 2D Array

```
int[,] numbers = { {1, 4, 2}, {3, 6, 8} };
Console.WriteLine(numbers[0, 2]);  // Outputs 2
```

Loop Through a 2D Array

```
int[,] numbers = { {1, 4, 2}, {3, 6, 8} };

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

```
using System;
class HelloWorld {
  static void Main() {
    int[,] numbers={{2,3,4},{6,4,2}};
    for(int i=0;i<numbers.GetLength(0);i++){
        for(int j=0;j<numbers.GetLength(1);j++){
        Console.Write(numbers[i,j]+"  ");
    }
    Console.WriteLine("");
  }
}
}
```

C# Methods

- A method is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a method.
- Methods are used to perform certain actions, and they are also known as functions.
  ↳
- Why use methods? To reuse code: define the code once, and use it many times.

Create a Method

- A method is defined with the name of the method, followed by parentheses (). C# provides some pre-defined methods, which you already are familiar with, such as Main(), but you can also create your own methods to perform certain actions:

```
class Program
{
  static void MyMethod()
  {
    // code to be executed
  }
}
```

Call a Method

```
static void MyMethod()
{
  Console.WriteLine("I just got executed!");
}

static void Main(string[] args)
{
  MyMethod();
}

// Outputs "I just got executed!"
```

C# Method Parameters

- Information can be passed to methods as parameter. Parameters act as variables inside the method.
- They are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

C# Method Parameters

```
using System;

namespace MyApplication
{
  class Program
  {
    static void MyMethod(string fname)
    {
      Console.WriteLine(fname + " 111");
    }

    static void Main(string[] args)
    {
      MyMethod("Abc");
      MyMethod("Xyz");
      MyMethod("123");
    }
  }
}
```

```
Abc 111
Xyz 111
123 111
```

C# Default Parameter Value

- You can also use a default parameter value, by using the equals sign (=).
- If we call the method without an argument, it uses the default value ():

```
using System;

namespace MyApplication
{
  class Program
  {
    static void MyMethod(string country = "Pakistan")
    {
      Console.WriteLine(country);
    }

    static void Main(string[] args)
    {
      MyMethod();
    }
  }
}
```

```
Pakistan
```

C# Return Values

```
static int MyMethod(int x)
{
  return 5 + x;
}

static void Main(string[] args)
{
  Console.WriteLine(MyMethod(3));
}

// Outputs 8 (5 + 3)
```