

Chapter 3: Data Types

C# is one of the languages provided by Microsoft to work with .Net. This language encompasses a rich set of features, which allows developing different types of applications.

C# is an object-oriented programming language and resembles several aspects of the C++ Language. In this tutorial, we see how to develop our first application. This will be a basic console application, we will then explore different data types available in the C# language as well as the control flow statements.

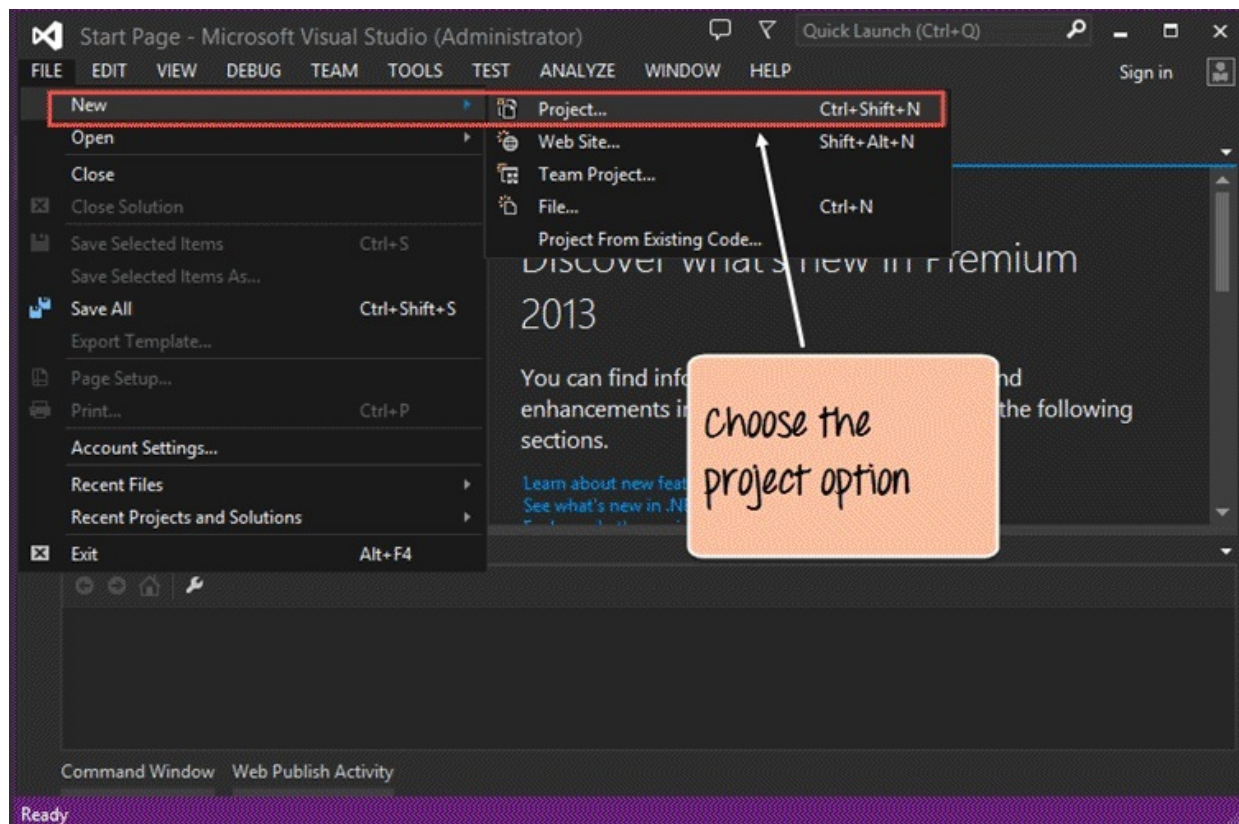
Building the first console application

A console application is an application that can be run in the command prompt in Windows. For any beginner on .Net, building a console application is ideally the first step to begin with.

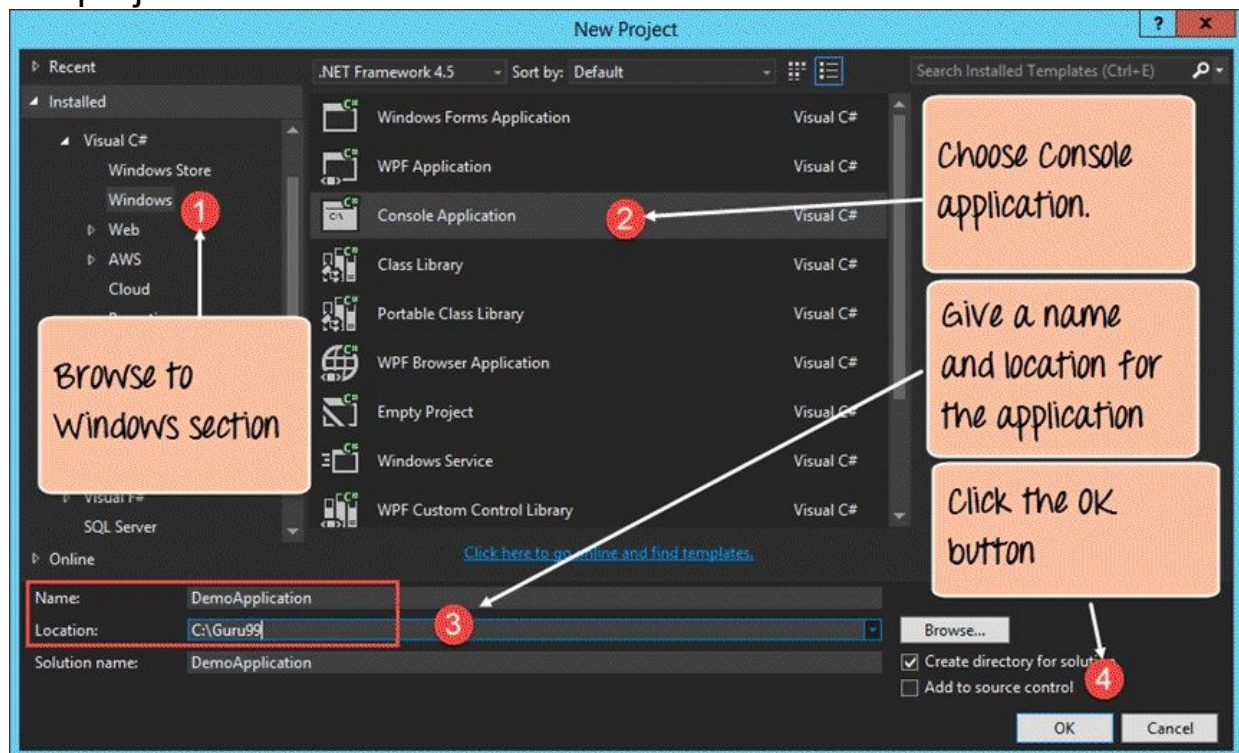
In our example, we are going to use Visual Studio to create a console type project. Next, we are going to use the console application to display a message “Welcome to .Net”. We will then see how to build and run the console application.

Let's follow the below-mentioned steps to get this example in place.

Step 1) The first step involves the creation of a new project in Visual Studio. For that, once the Visual Studio is launched, you need to choose the menu option New->Project.

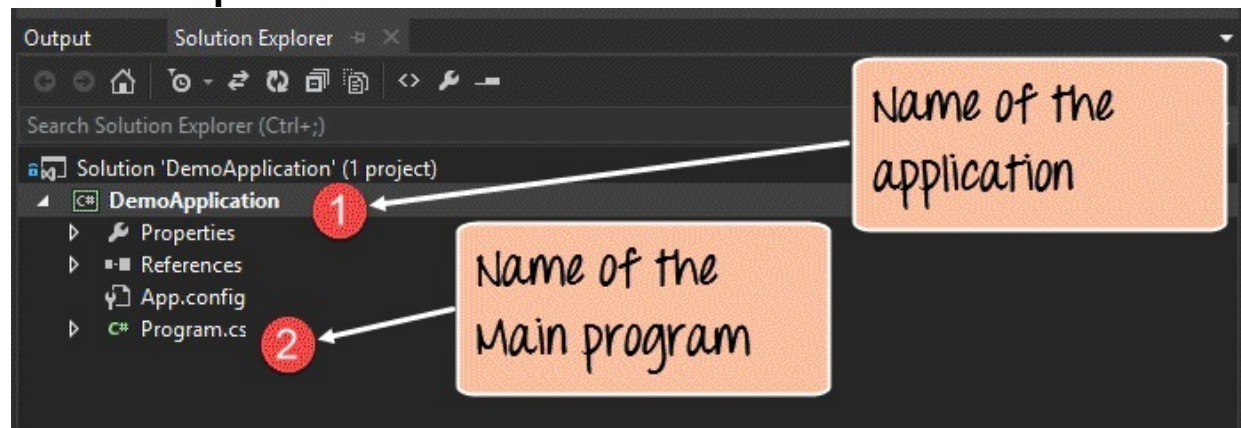


Step 2) The next step is to choose the project type as a Console application. Here, we also need to mention the name and location of our project.



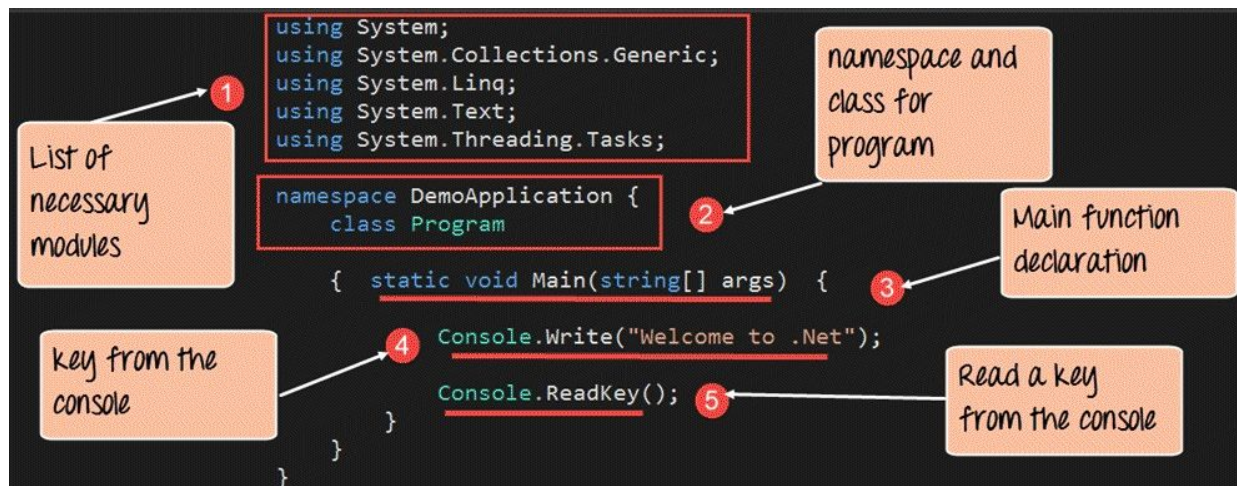
1. In the project dialog box, we can see various options for creating different types of projects in Visual Studio. Click the Windows option on the left-hand side.
2. When we click the Windows options in the previous step, we will be able to see an option for Console Application. Click this option.
3. We then give a name for the application which in our case is DemoApplication. We also need to provide a location to store our application.
4. Finally, we click the 'OK' button to let Visual Studio to create our project.

If the above steps are followed, you will get the below output in Visual Studio. **Output:**



1. A project called 'DemoApplication' will be created in Visual Studio. This project will contain all the necessary artifacts required to run the Console application.
2. The Main program called Program.cs is default code file which is created when a new application is created in Visual Studio. This code will contain the necessary code for our console application.

Step 3) Now let's write our code which will be used to display the string "Welcome to .Net" in the console application. All the below code needs to be entered in the Program.cs file. The code will be used to write "Welcome to .Net" when the console application runs.



Code Explanation:

1. The first lines of code are default lines entered by Visual Studio. The 'using' statement is used to import existing .Net modules in our console application. These modules are required for any .Net application to run properly. They contain the bare minimum code to make a code work on a Windows machine.

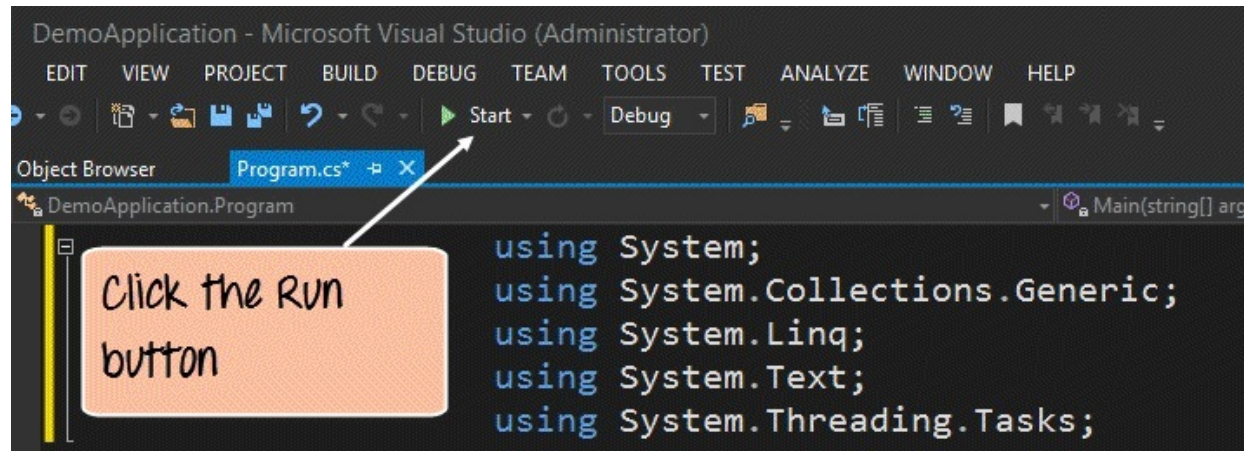
2. Every application belongs to a class. C# is an object-oriented language, and hence, all code needs to be defined in a self-sustaining module called a 'Class.' In turn, every class belongs to a namespace. A namespace is just a logically grouping of classes.

3. The Main function is a special function which is automatically called when a console application runs. Here you need to ensure to enter the code required to display the required string in the console application.

4. The Console class is available in .Net which allows one to work with console applications. Here we are using an inbuilt method called 'Write' to write the string "Welcome to .Net" in the console.

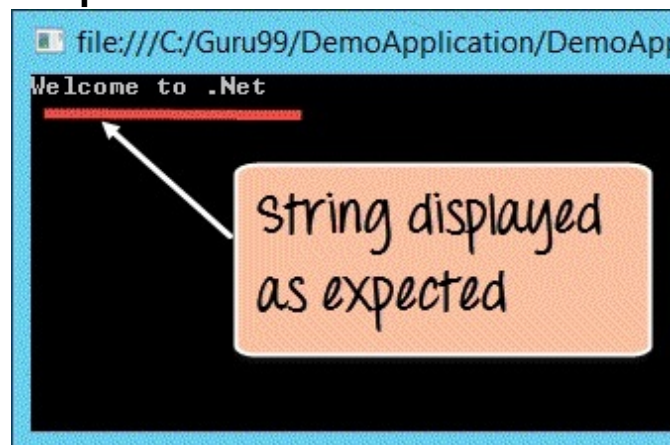
5. We then use the Console.ReadKey() method to read any key from the console. By entering this line of code, the program will wait and not exit immediately. The program will wait for the user to enter any key before finally exiting. If you don't include this statement in code, the program will exit as soon as it is run.

Step 4) Run your .Net program. To run any program, you need to click the Start button in Visual Studio.



If the above code is entered properly and the program is executed successfully, the following output will be displayed.

Output:

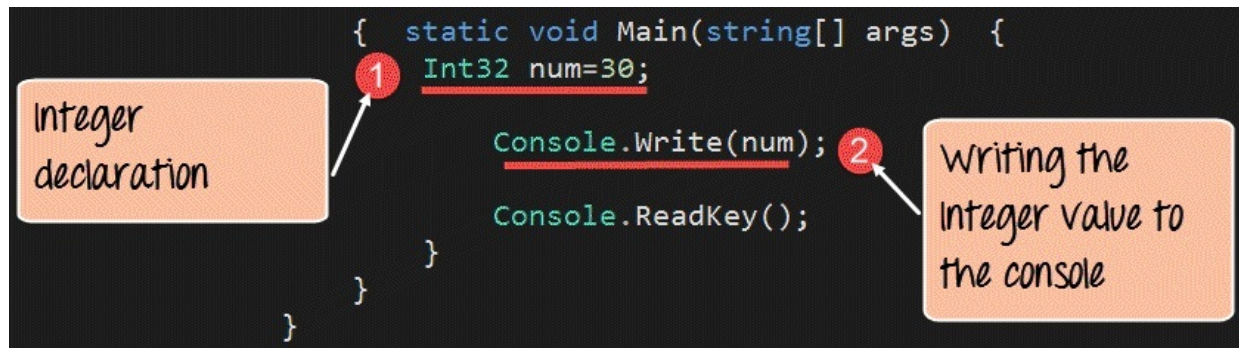


From the output, you can clearly see that the string "Welcome to .Net" is displayed properly. This is because the Console.write statement causes this string to be sent to the console.

C# Data Types

The C# language comes with a set of Basic data types. These data types are used to build values which are used within an application. Let's explore the basic data types available in C#. For each example, we will modify just the main function in our Program.cs file.

1. **Integer** – An Integer data types is used to work with numbers. In this case, the numbers are whole numbers like 10, 20 or 30. In C#, the datatype is denoted by the **Int32 keyword**. Below is an example of how this datatype can be used. In our example, we will define an Int32 variable called num. We will then assign an Integer value to the variable and then display it accordingly.



```
{ static void Main(string[] args) {  
    Int32 num=30;  
    Console.Write(num);  
    Console.ReadKey();  
}
```

The image shows a C# code snippet with two annotations. Annotation 1, labeled 'Integer declaration', points to the line `Int32 num=30;`. Annotation 2, labeled 'Writing the Integer value to the console', points to the line `Console.Write(num);`. The code is enclosed in a `{ static void Main(string[] args) {` block and ends with closing braces `}`.

Code Explanation:

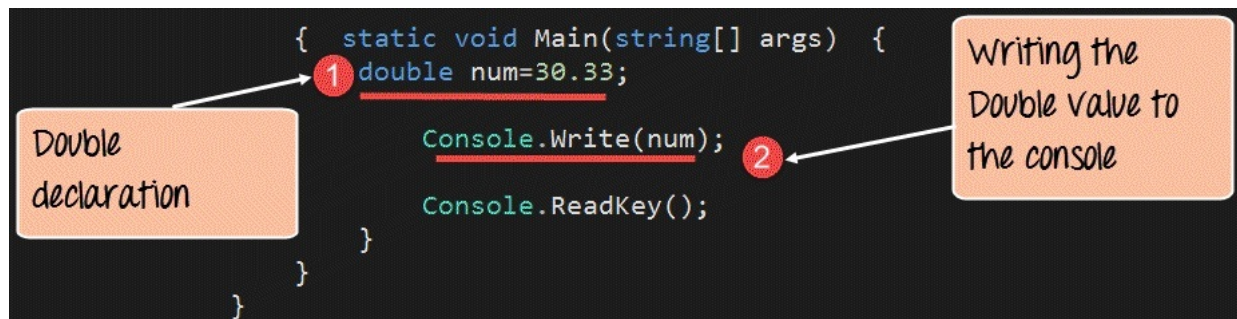
1. The Int32 data type is specified to declare an Integer variable called num. The variable is then assigned a value of 30.
2. Finally the console.write function is used to display the number to the console.

If the above code is entered properly and the program is executed successfully, following output will be displayed.

From the output, you can clearly see that the Integer variable called num was displayed in the console

2. **Double** - A double data type is used to work with decimals. In this case, the numbers are whole numbers like 10.11, 20.22 or 30.33. In C#, the datatype is denoted by the keyword "**Double**". Below is an example of this datatype .

In our example, we will define a double variable called num. We will then assign a Double value to the variable and then display it accordingly.



Code Explanation:

1. The double data type is specified to declare a double type variable called num.

The variable is then assigned a value of 30.33.

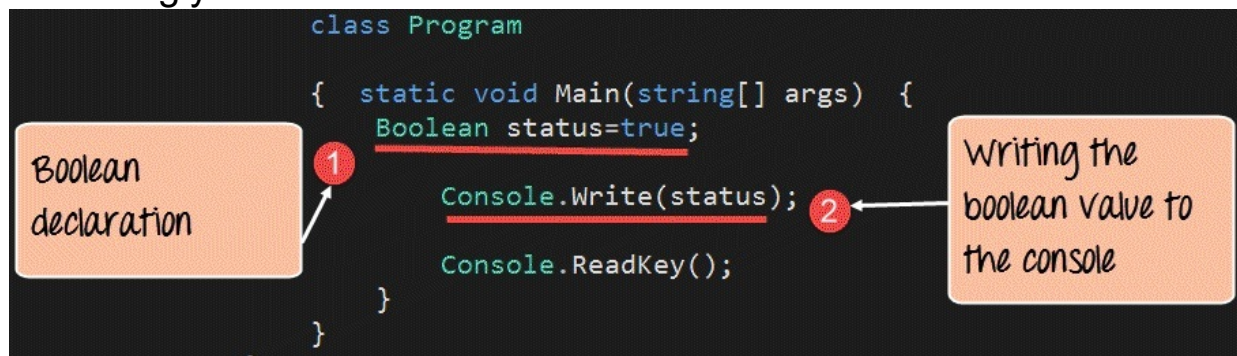
2. Finally the console.write function is used to display the number to the console.

If the above code is entered properly and the program is executed successfully, following output will be displayed.

From the output, you can clearly see that the double variable called num was displayed in the console

3. **Boolean** - A boolean data type is used to work with Boolean values of **true and false**. In C#, the datatype is denoted by the Boolean keyword. Below is an example of this datatype can be used.

In our example, we will define a Boolean variable called 'status.' We will then assign a boolean value to the variable and then display it accordingly.



Code Explanation:

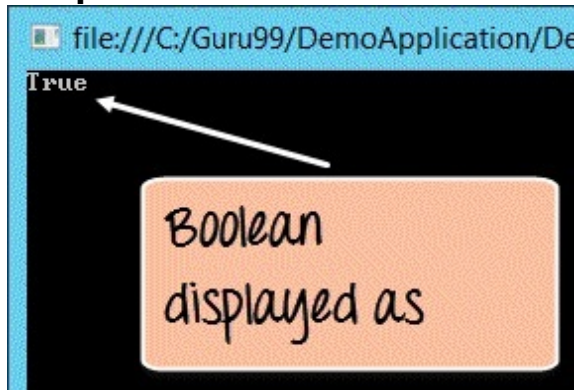
1. The boolean data type is specified to declare a Boolean variable called 'status.'

The variable is then assigned a value of true/false.

2. Finally the console.write function is used to display the Boolean value to the console.

If the above code is entered properly and the program is executed successfully, the output will be displayed.

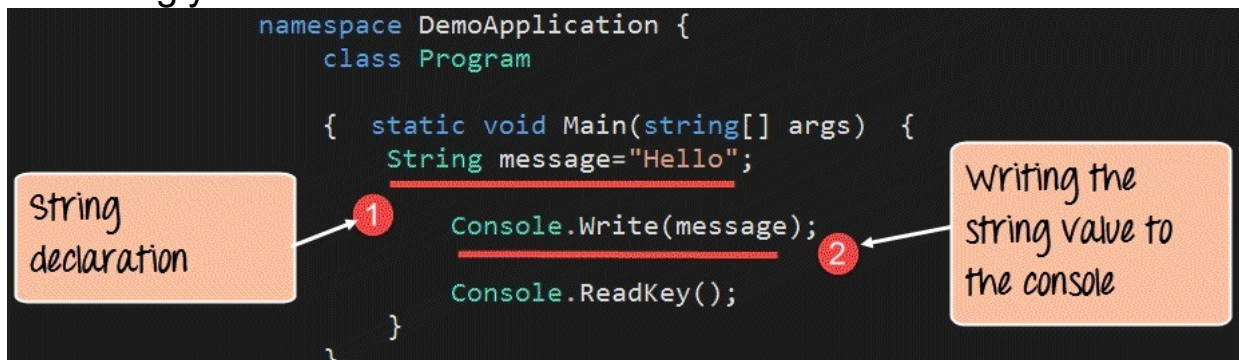
Output:



From the output, you can clearly see that the Boolean variable which equals true was displayed in the console

4. **String** - A String data type is used to work with String values. In C#, the datatype is denoted by the keyword 'String'. Below is an example of this datatype.

In our example, we will define a String variable called 'message.' We will then assign a String value to the variable and then display it accordingly.



Code Explanation:

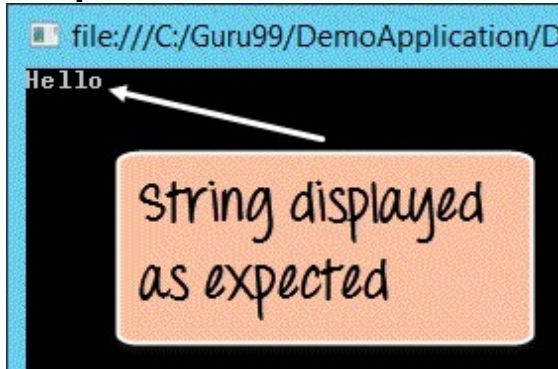
1. The String data type is specified to declare a string variable called message.

The variable is then assigned a value of "Hello".

2. Finally, the console.write function is used to display the string value to the console.

If the above code is entered properly and the program is executed successfully, the output will be displayed.

Output:



From the output, you can clearly see that the String variable called message was displayed in the console

C# Enumeration

An enumeration is used in any programming language to define a constant set of values. For example, the days of the week can be defined as an enumeration and used anywhere in the program. In C#, the enumeration is defined with the help of the keyword 'enum'.

Let's see an example of how we can use the 'enum' keyword.

In our example, we will define an enumeration called days, which will be used to store the days of the week. For each example, we will modify just the main function in our Program.cs file.

```

namespace DemoApplication {
    class Program
    {
        1 enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };
        static void Main(string[] args)
        {
            Console.WriteLine(Days.Sun);
            Console.ReadKey();
        }
    }
}

```

enum data type declaration

2 Displaying a value of the enum data type

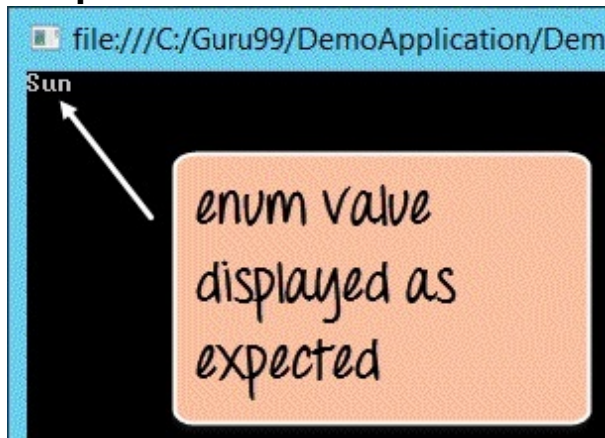
Code Explanation:

1. The 'enum' data type is specified to declare an enumeration. The name of the enumeration is Days. All the days of the week are specified as values of the enumeration.

2. Finally the console.write function is used to display one of the values of the enumeration.

If the above code is entered properly and the program is executed successfully, the following output will be displayed.

Output:



file:///C:/Guru99/DemoApplication/Demo

Sun

enum value displayed as expected

From the output, you can clearly see that the 'Sun' value of the enumeration is displayed in the console.

C# Operators and Variables

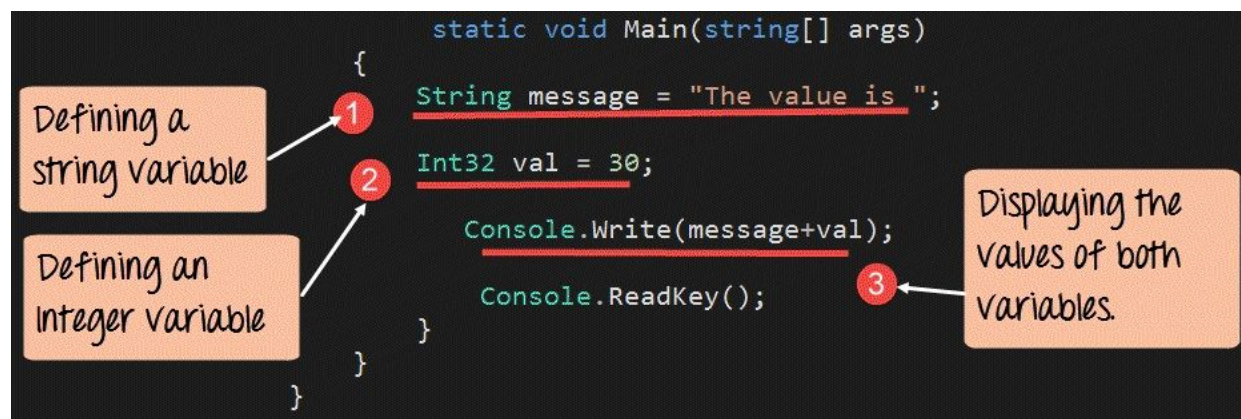
A variable is a name given to a storage area that is used to store values of various data types. Each variable in C# needs to have a specific type, which determines the size and layout of the variable's memory.

For example, a variable can be of the type String, which means that it will be used to store a string value. Based on the data type, specific operations can be carried out on the variable.

For instance, if we had an Integer variable, then operations such as addition and subtraction can be carried out on the variable. One can declare multiple variables in a program.

Let's look at a quick example on the declaration of multiple variables of different data types.

In our example, we will define 2 variables, one of the type 'string' and the other of the type 'Integer'. We will then display the values of these variables to the console. For each example, we will modify just the main function in our Program.cs file.

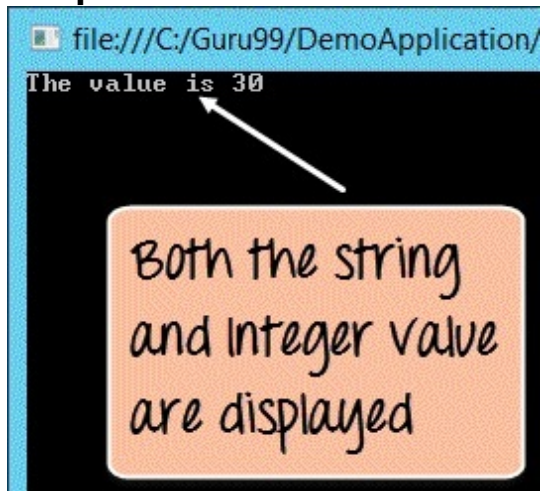


Code Explanation:

1. A variable of the data type String is declared. The name of the variable is 'message'. The value of the variable is "The value is ".
2. A variable of the data type Integer (Int32) is declared. The name of the variable is 'val'. The value of the variable is 30.
3. Finally the Console.write statement is used to output both the value of the String and Integer variable.

If the above code is entered properly and the program is executed successfully, the following output will be displayed.

Output:



From the output, you can clearly see that the values of both the string and integer variable are displayed to the console.

Operators are used to perform operations on values of various data types. For example, to perform the addition of 2 numbers, the + operator is used. Let's see the table of operators available for the various data types

1. Arithmetic Operators – These are operators used for performing mathematic operations on numbers. Below is the list of operators available in C#.

Operator Description

- + Adds two operands
- Subtracts second operand from the first
- * Multiplies both operands
- / Divides numerator by de-numerator
- % Modulus Operator and remainder of after an integer division
- ++ Increment operator increases integer value by one
- Decrement operator decreases integer value by one

2. Relational Operators – These are operators used for performing Relational operations on numbers. Below is the list of relational operators available in C#.

Operator Description

tor

`==` Checks if the values of two operands are equal or not, if yes then condition becomes true. `!=` Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.

`>` Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.

`<` Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.

`>=` Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.

`<=` Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

3. Logical Operators – These are operators used for performing Logical operations on values. Below is the list of operators available in C#.

Operator Description

`&&` This is the Logical AND operator. If both the operands are true, then condition becomes true. `||` This is the Logical OR operator. If any of the operands are true, then condition becomes true. `!` This is the Logical NOT operator.

Let's look at a quick example of how the operators can be used in .Net. In our example, we will define 2 Integer variables and one Boolean variable. We will then perform the following operations

```
namespace DemoApplication {
    class Program
    {
        static void Main(string[] args)
        {
            Int32 val1=10,val2=20;
            bool status = true;

            Console.WriteLine(val1+val2);
            Console.WriteLine(val1 < val2);
            Console.WriteLine(!(status));
            Console.ReadKey();
        }
    }
}
```

The diagram illustrates the use of operators in the provided C# code. Annotations include:

- Defining Variables**: Points to the variable declarations `Int32 val1=10, val2=20;` and `bool status = true;` (labeled 1).
- Using arithmetic operators**: Points to the expression `val1+val2` in `Console.WriteLine(val1+val2);` (labeled 2).
- Using Relational operators**: Points to the expression `val1 < val2` in `Console.WriteLine(val1 < val2);` (labeled 3).
- Using logical operators**: Points to the expression `!(status)` in `Console.WriteLine(!(status));` (labeled 4).

Code Explanation:

1. Two Integer variables are defined, one being `val1` and the other being `val2`. These will be used to showcase relational and arithmetic

operations. A Boolean variable is defined to showcase logical operations.

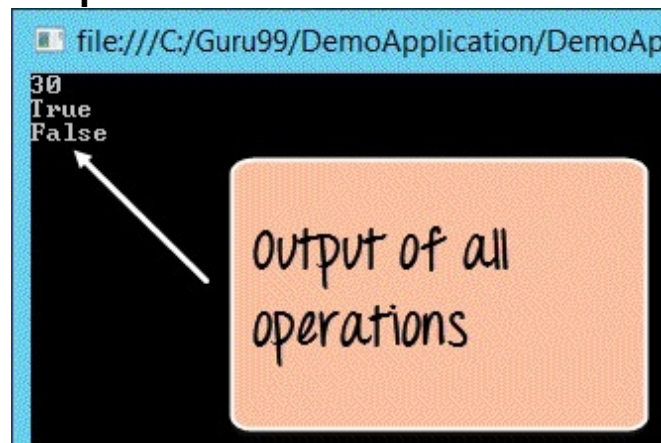
2. An example of the arithmetic operation is shown wherein the addition operator is carried out on val1 and val2. The result is written to the console.

3. An example of the relational operation is shown wherein the less than operator is carried out on val1 and val2. The result is written to the console.

4. An example of the logical operation is shown, wherein the logical operator (!) is applied to the status variable. The logical NOT operator reverses the current value of any Boolean value. So if a Boolean value is 'true', the logical NOT will return the value 'false' and vice versa. In our case since the value of the status variable is 'true', the result will show 'false'. The result is written to the console.

If the above code is entered properly and the program is executed successfully, the output will be displayed.

Output:



```
file:///C:/Guru99/DemoApplication/DemoAp
30
True
False
```

output of all operations

Flow Control and conditional statements

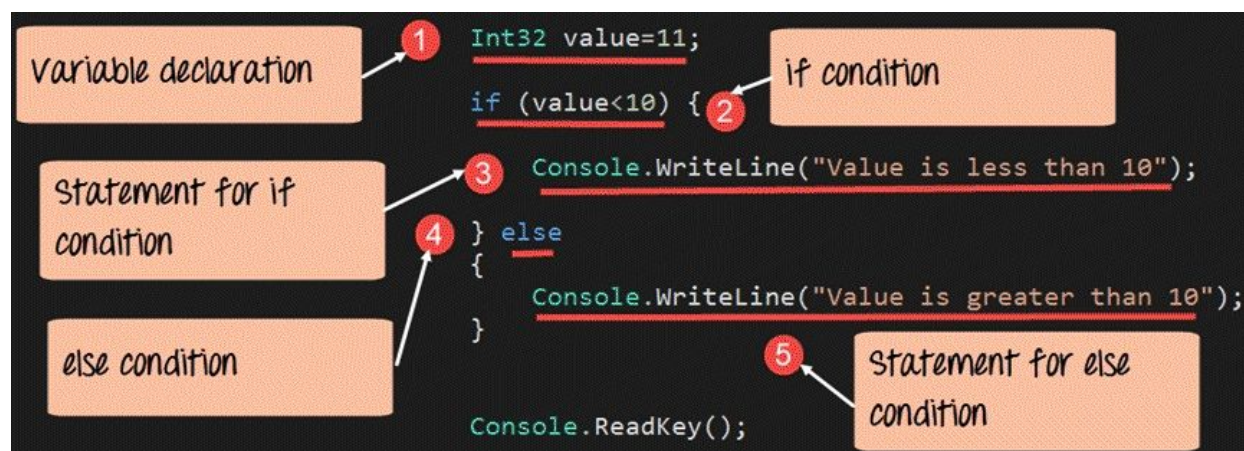
Flow control and conditional statements are available in any programming language to alter the flow of a program. For example, if someone want to execute only a particular set of

statements based on some certain logic, then Flow control and conditional statements will be useful. You will get a better understanding as we go through the various statements which are available in C#.

Please note that all the code below is made to the Program.cs file.

1. If statement – The if statement is used to evaluate a boolean expression before executing a set of statements. If an expression evaluates to true, then it will run one set of statements else it will run another set of statements.

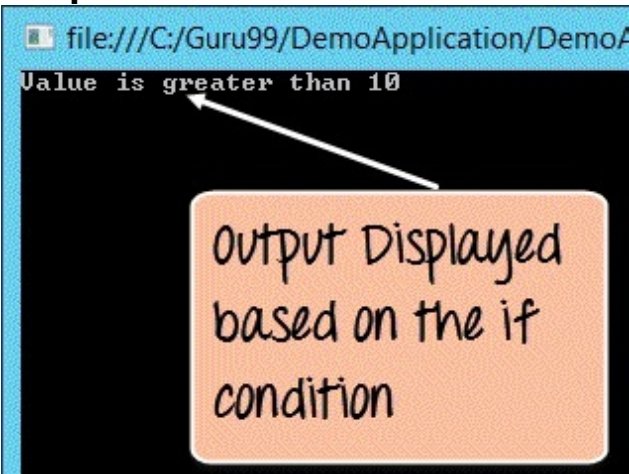
In our example below, a comparison is made for a variable called value. If the value of the variable is less than 10 ,then it will run one statement , or else it will run on another statement.



Code Explanation:

1. We first define a variable called value and set it to the value of 11.
 2. We then use the 'if' statement to check if the value is less than 10 of the variable. The result will either be true or false.
 3. If the if condition evaluates to true, we then send the message "Value is less than 10" to the console.
 4. If the if condition evaluates to false, we then send the message "Value is greater than 10" to the console.
- If the above code is entered properly and the program is executed successfully, the following output will be displayed.

Output:

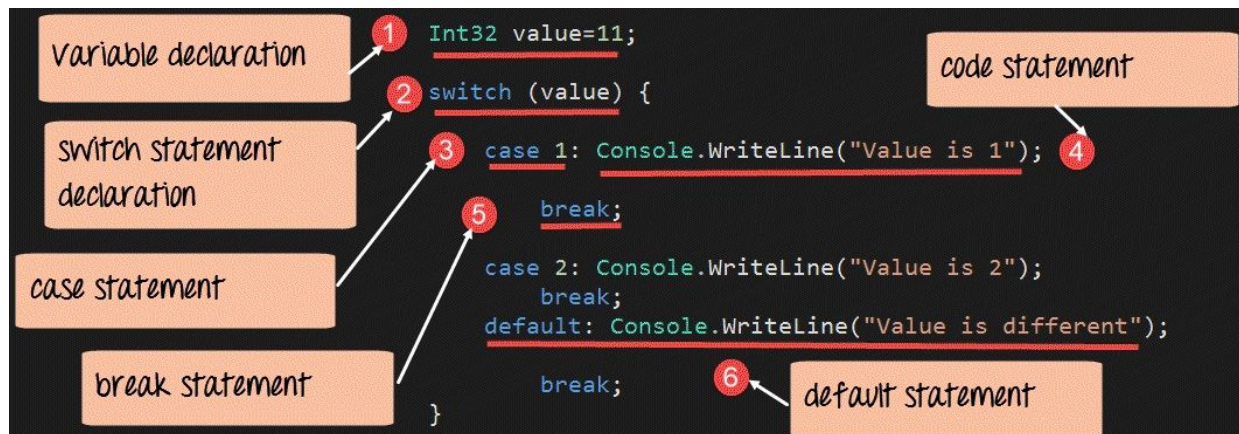


We can clearly see that the 'if' statement was evaluated to false. Hence the message "Value is greater than 10" was sent to the console.

2. Switch statement – The switch statement is an enhancement to the 'if' statement. If you have multiple expressions that need to be evaluated in one shot, then writing multiple 'if' statements becomes an issue.

The switch statement is used to evaluate an expression and run different statements based on the result of the expression. If one condition does not evaluate to true, the switch statement will then move to the next condition and so forth.

Let's see, how this works with the below example. Here, we are again comparing the value of a variable called 'value.' We then check if the value is equal to 1, or 2, or something totally different.

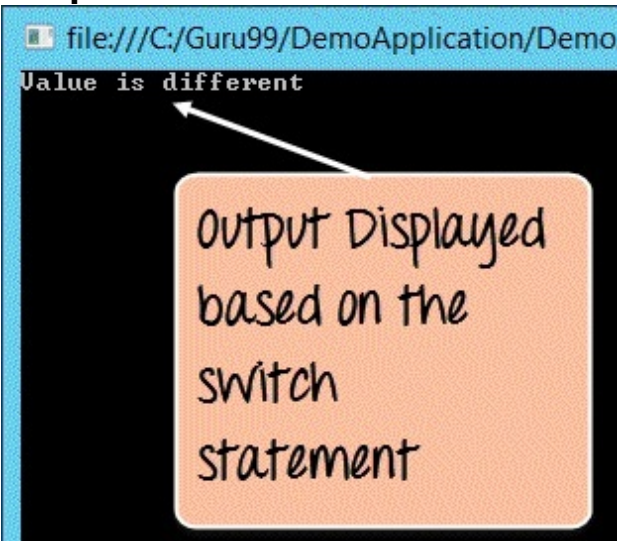


Code Explanation:

1. We first define a variable called 'value' and set it to the value of 11.
2. We then use the 'switch' statement to check the value of the variable 'value.'
3. Case statements are used to set different conditions. Based on the conditions, a set of statements can be executed. A switch statement can have multiple case conditions. The first case statement checks to see if the value of the variable is equal to 1.
4. If the first case statement is true, then the message "Value is 1" is written to the console.
5. The break statement is used to break from the entire switch statement, once a condition is true.
6. The default condition is a special condition. This just means that if no case expression evaluates to true, then run the set of statements for the default condition.

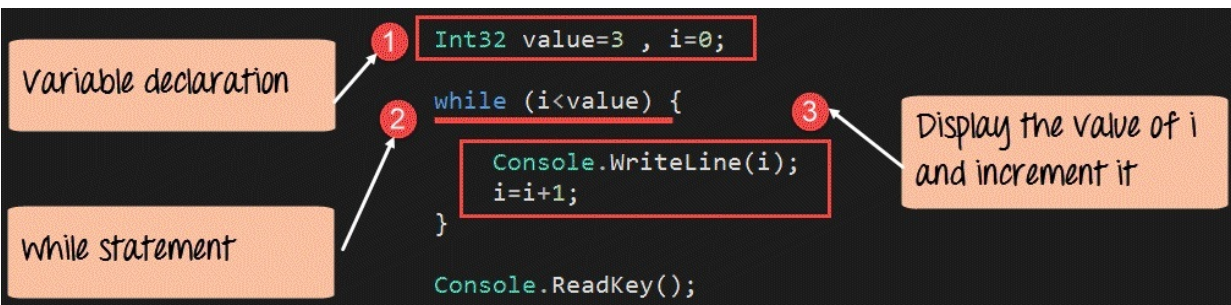
If the above code is entered properly and the program is executed successfully, the following output will be displayed. The output prints the default value "Value is different", since no condition is satisfied.

Output:



3. **While loop** – The while loop is used for iterative purposes. Suppose if you want to repeat a certain set of statements for a particular number of times, then while loop is used.

In our example below, we use the while statement to display the value of a variable 'i'. The while statement is used to display the value 3 times.



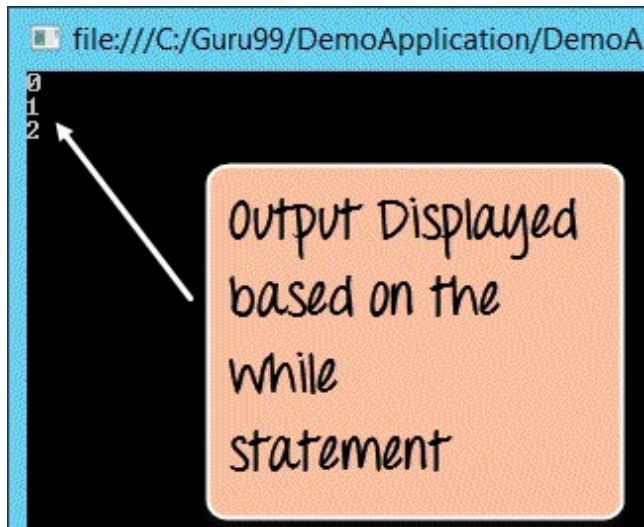
Code Explanation:

1. Two Integer variables are defined, one being value and the other being 'i'. The value variable is used as the upper limit to which we should iterate our while statement. And 'i' is the variable which will be processed during the iteration.
2. In the while statement, the value of 'i' is constantly checked against the upper limit.
3. Here we display the value of 'i' to the console. We also increment

the value of 'i'.

If the above code is entered properly and the program is executed successfully, the following output will be displayed.

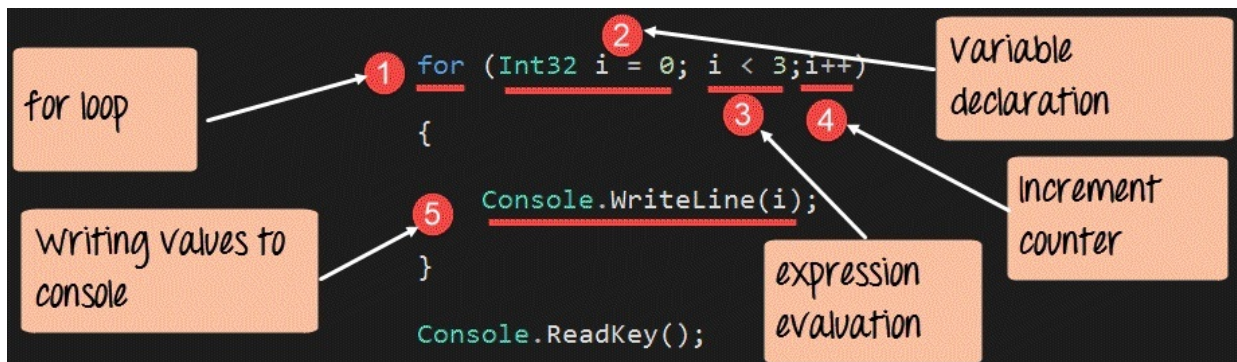
Output:



Here you can see that the while statement is executed 3 times and incremented at the same time. And each time, it displayed the current value of the variable 'i'.

4. For loop - The 'for' loop is also used for iterative purposes. Suppose if you want to repeat a certain set of statements for a particular number of times, then for loop is used.

In our example below, we use the 'for' statement to display the value of a variable 'i'. The 'for' statement is used to display the value 3 times.



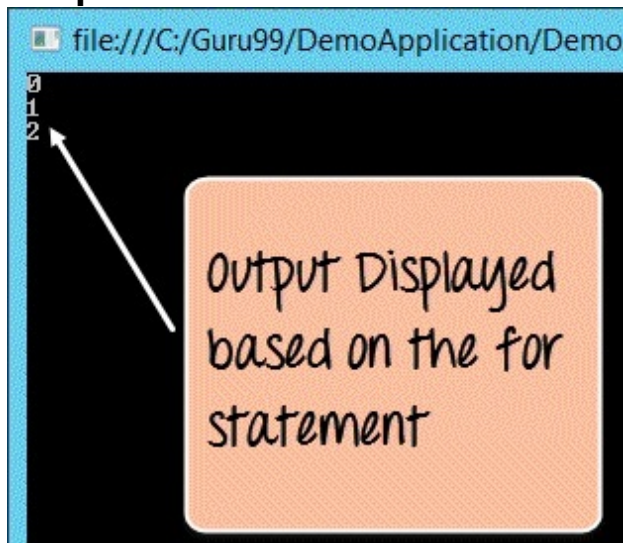
Code Explanation:

1. The 'for' keyword is used to start off the 'for loop' statement.

2. In the 'for loop', we define 3 things. The first is to initialize the value of a variable, which will be used in the 'for loop'.
3. The second is to compare the value of the 'i' against an upper limit. In our case, the upper limit is the value of 3 ($i < 3$).
4. Finally, we increment the value of 'i' accordingly.
5. Here we display the value of 'i' to the console.

If the above code is entered properly and the program is executed successfully, the following output will be displayed.

Output:



Here you can see that the 'for' statement is executed 3 times. And each time, it displayed the current value of the variable 'i'.

C# Arrays

An array is used to store a collection or series of elements. These elements will be of the same type.

So for example, if you had an array of Integer values, the array could be a collection of values such as [1, 2, 3, 4]. Here the number of elements in the array is 4.

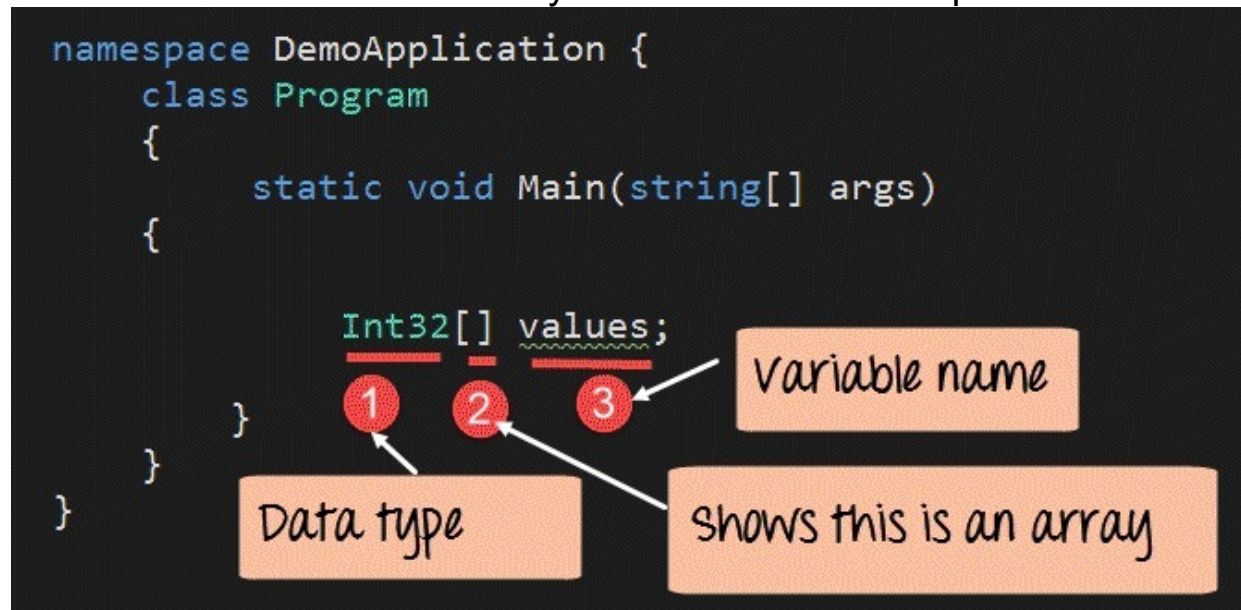
Arrays are useful when you want to store a collection of values of the same type. So instead of declaring a variable for each and every element, you can just declare one variable.

This variable will point to an array or list of elements, which will be responsible for storing the elements of the array.

Let's look at how we can work with arrays in C#. In our example, we will declare an array of Integers and work with them accordingly.

Note that all of the below code is being made to the Program.cs file.

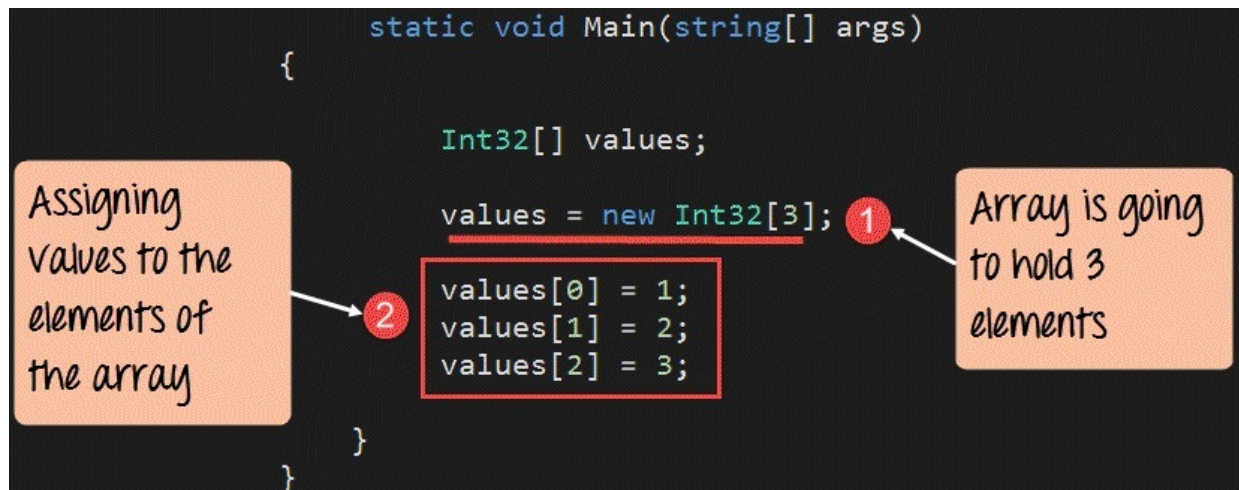
Step 1) Declaring an array – The first step is to declare an array. Let's see how we can achieve this by the below code example.



Code Explanation:

1. The first part is the datatype. It specifies the type of elements used in the array. So in our case, we are creating an array of Integers.
2. The second part `[]`, which specifies the rank of the array. (The rank is a placeholder which specifies the number of elements the array will contain)
3. Next is the Name of the array which in our case is 'values'. Note you are seeing a green squiggly underline, don't worry about that. That is just .Net saying that you have declared an array, but not using it anywhere.

Step 2) The next step is to initialize the array. Here we are going to specify the number of values the array will hold. We are also going to assign values to each element of the array.



Code Explanation:

1. First, we are setting the number of elements the array will hold to 3. So in the square brackets, we are saying that the array will hold 3 elements.
2. Then we are assigning values to each individual element of the array. We can do this by specifying the variable name + the index position in the array.

So `values[0]` means that we are storing a value in the first position of the array. Similarly to access the second position, we use the notation of `values[1]` and so on and so forth.

Note: - In Arrays, the index position starts from .

Step 3) Let's now display the individual elements of the array in the Console. Let's add the below code to achieve this.

```
values = new Int32[3];
```

```
values[0] = 1;
```

```
values[1] = 2;
```

```
values[2] = 3;
```

```
Console.WriteLine(values[0]);
```

```
Console.WriteLine(values[1]);
```

```
Console.WriteLine(values[2]);
```

```
Console.ReadKey();
```

Using
Console.WriteLine
to send each
element value to
the console

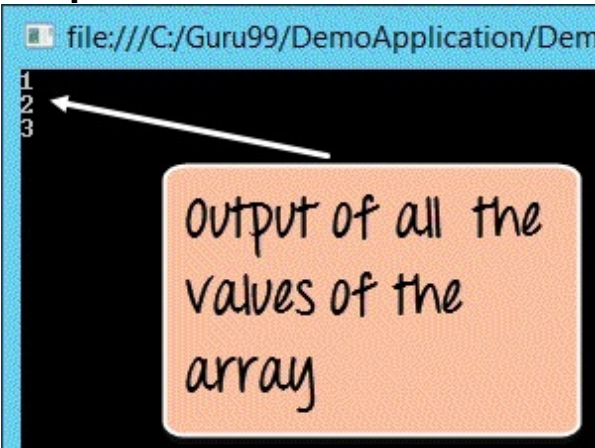
Code Explanation:

This is the simple part wherein we just use the Console.Write method to send each value of the element to the console.

Note that again, we are accessing each element with the help of the array variable name along with the index position.

If the above code is entered properly and the program is executed, the following output will be displayed.

Output:



```
file:///C:/Guru99/DemoApplication/Dem
1
2
3
```

Output of all the
values of the
array

From the output, you can see all the values of the array being displayed in the Console.

Summary

A Console application is one that can be made to run in the command prompt on a windows machine.

The Console.write method can be used to write content to the console. The basic data types available in C# are Integer, Double, Boolean, and String. Enumerations are used to declare a set of

Constant values. In C# enumerations are declared with the use of the enum keyword.

The various operators available in C# are broadly classified into the categories of Arithmetic, Relational, and Logical operators.

Variables are used to point to memory locations which contain values of a particular data type.

Arrays are used to store elements of the same type. Individual elements of the array can be assigned values.