**Nullable Types in C#**

**Definition**

A nullable type in C# is a data type that contains the defined data type or the value of null. You should note here that variable datatype has been given, and only it can be used. This nullable type concept is not compatible with "var".

This article explains nullable types in C# and how to use a nullable type in a .NET application.

**Declaration**

Any DataType can be declared a nullable type with the help of the operator "?".

An example of the syntax is as follows:

int? i = null;

As discussed in the previous section, "var" is incompatible with this Nullable type.

So we will have Compile Time error if we are declaring something like:

var? i = null;

Though the following syntax is completely fine:

var i = 4;

**Usage**

This nullable type's main usage is when we pass any parameter to Stored Procedure or Database objects. If a column in a Table allows a Null value, we should use the Nullable type in this case.

For example, say I have a Stored Procedure that accepts two parameters, @A and @B. It gives me back one out of Parameter @C.

This output Parameter can also be null. So, in that case, we should use a Nullable Variable, which can also take invalid as an allowed value.

So to conclude, Nullable Types  allows us to declare variables in .net which can be used effectively while dealing with databases.

**Different Types of Operations on this Nullable Type and Expected Results**

Now let us discuss in detail a few operations on these Nullable type

1. A Nullable Type is incompatible with a General Data Type. This means we cannot operate between a nullable  type and a general datatype.  
     
   For example
2. int? x = 4;
3. int y = 3;
4. int z = x \* y;
5. Console.WriteLine(z.ToString());

Console.ReadLine();

Here x is nullable while y is not nullable. So The above Program will generate the following compile time Error:  
  
*Error 1 Cannot implicitly convert type 'int?' to 'int'. An explicit conversion exists (are you missing a cast?)*  
  
To avoid this error, we need to declare all the variables as Nullable, like in the following code:

int? x = 4;

int? y = 3;

int? z = x \* y;

Console.WriteLine(z.ToString());

Console.ReadLine();

This program works fine with no compile-time and runtime errors.

1. In the first step, I discussed that no nullable type operation is possible with a not-nullable type. But we can always compare a Nullable Type with a Not Nullable type.  
     
   For example, look at the following code.
2. class Program
3. {
4. static void Main(string[] args)
5. {
6. int? x = 3;
7. int y = 3;
8. bool check = false;
9. if (x == y)
10. {
11. check = true;
12. Console.WriteLine(check.ToString());
13. Console.ReadLine();
14. }
15. }

}

The above code snippet works fine with no error and prints the value of the check as true.

1. The above code snippet also works fine with the "!=" operator.
2. Null coalescing operator  
     
   This operator in C# is represented as "??". Look at the following code snippet for a better understanding of this operator.
3. class Program
4. {
5. static void Main(string[] args)
6. {
7. int? x = null;
8. int y = 4;
9. int? result;
10. result = x ?? y;
11. Console.WriteLine(result.ToString());
12. Console.ReadLine();
13. }

}

Look at the statement:  result = x ?? y;  
  
This operator assigns the value of "x" to" result " if "x" is null; else, it will give the value of "y" to "result".  
  
So if the first variable is null, then that value will be assigned else, the value of the second variable will be assigned.  
  
As expected in the above case "result" will hold null.  
  
**Caution:** The variable "result" can also have null. Therefore it should be defined as a Nullable type.

1. Assigning Nullable Type variable Value to a Not Nullable Type  
     
   Directly assigning Nullable Type Variable Value to a not Nullable Type Variable is illegal.  
     
   Look at the following code snippet.
2. class Program
3. {
4. static void Main(string[] args)
5. {
6. int? x = 4;
7. int y = x;
8. Console.WriteLine(y.ToString());
9. Console.ReadLine();
10. }

}

This code snippet will generate the following compile time error:  
  
*Error 1 Cannot implicitly convert type 'int?' to 'int'. An explicit conversion exists (are you missing a cast?)*  
  
But there is a workaround for this with the help of the null coalescing operator.

class Program

{

static void Main(string[] args)

{

int? x = 4;

int y = x ?? 0;

Console.WriteLine(y.ToString());

Console.ReadLine();

}

}

If you run this program, y will have a value of x, i.e., 4. So in place of 0 in the statement int y = x ?? 0;, you can use any integer.  
  
Whatever integer you use will be the default value of "y."