Nullable Value Types

A *nullable value type* `T?` represents all values of its underlying value type `T` and an additional null
value.

For example, you can assign any of the following three values to
a `bool?` variable: `true`, `false`, or `null`. An underlying value type `T` cannot be a nullable value type itself.

Declaration and assignment

As a value type is implicitly convertible to the corresponding nullable value type, you can assign a value to a variable of a nullable value type as you would do that for its underlying value type.

You can also assign the `null` value. For example:

```
double? pi = 3.14;
char? letter = 'a';

int m2 = 10;
int? m = m2;

bool? flag = null;

// An array of a nullable value type:
int?[] arr = new int?[10];
```

Conversion from a nullable value type to an underlying type

If you want to assign a value of a nullable value type to a non-nullable value type variable, you might need to specify the value to be assigned in place of `null`.

Use the null-coalescing operator `??` to do that (you can also use
the Nullable<T>.GetValueOrDefault(T) method for the same purpose):

```
int? a = 28;
int b = a ?? -1;
Console.WriteLine($"b is {b}");  // output: b is 28

int? c = null;
int d = c ?? -1;
Console.WriteLine($"d is {d}");  // output: d is -1
```

Lifted operators

The predefined unary and binary operators or any overloaded operators that are supported by a value type `T` are also supported by the corresponding nullable value type `T?`.

These operators, also known as *lifted operators*, produce `null` if one or both operands are `null`; otherwise, the operator uses the contained values of its operands to calculate the result.

For example:

```
int? a = 10;
int? b = null;
int? c = 10;

a++;         // a is 11
a = a * c;  // a is 110
a = a + b;  // a is null
```