Operators

.NET

C# provides a number of operators that allow you to perform evaluation and manipulate the data of variables.

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/

Above is the docs for operators.

Type-testing and cast operators – ‘*is*’ operator

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast#typeof-operator

---

The is operator checks if the runtime ***type*** of an expression result is compatible with a given ***type***.

`E is T` returns ***true*** if E is non-null and can be converted to ***type*** T by a ***reference***, a ***boxing***, or an ***unboxing*** conversion.

```
public class Base { }

public class Derived : Base { }

public static class IsOperatorExample
{
    public static void Main()
    {
        object b = new Base();
        Console.WriteLine(b is Base);  // output: True
        Console.WriteLine(b is Derived);  // output: False

        object d = new Derived();
        Console.WriteLine(d is Base);  // output: True
        Console.WriteLine(d is Derived); // output: True
    }
}
```

Type-testing and cast operators – ‘*is*’ operator

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast#typeof-operator

---

The is operator takes into account ***boxing*** and ***unboxing*** conversions but doesn't consider numeric conversions.

```
int i = 27;
Console.WriteLine(i is System.IFormattable);  // output: True

object iBoxed = i;
Console.WriteLine(iBoxed is int);  // output: True
Console.WriteLine(iBoxed is long);  // output: False
```

Type-testing and cast operators – ‘*as*’ operator

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast#as-operator

---

The as operator explicitly converts the result of an expression to a given reference or ***nullable*** value type. If the conversion is not possible, the as operator returns null. Unlike the ***cast*** operator (), the as operator never throws an exception.

```
E as T
```

produces the same result as

```
E is T ? (T)(E) : (T)null
```

The as operator considers only ***reference***, ***nullable***, ***boxing***, and ***unboxing*** conversions.

You cannot use the as operator to perform a user-defined conversion. To do that, use the ***cast*** operator ().

typeof

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast#typeof-operator

---

The typeof operator obtains the System.Type instance ***type***. The argument to the typeof operator must be the name of a ***type*** or a ***type parameter***.

```
void PrintType<T>() => Console.WriteLine(typeof(T));

Console.WriteLine(typeof(List<string>));
PrintType<int>();
PrintType<System.Int32>();
PrintType<Dictionary<int, char>>();
// Output:
// System.Collections.Generic.List`1[System.String]
// System.Int32
// System.Int32
// System.Collections.Generic.Dictionary`2[System.Int32,System.Char]
```

typeof Operator

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/type-testing-and-cast#typeof-operator

- An expression cannot be an argument of the typeof operator. To get the ***System.Type*** instance for the runtime ***type*** of an expression result, use Object.GetType().
- Use the typeof operator to check if the runtime ***type*** of the expression result exactly matches a given ***type***.

```
public class Animal { }

public class Giraffe : Animal { }

public static class TypeOfExample
{
    public static void Main()
    {
        object b = new Giraffe();
        Console.WriteLine(b is Animal);  // output: True
        Console.WriteLine(b.GetType() == typeof(Animal));  // output: False

        Console.WriteLine(b is Giraffe);  // output: True
        Console.WriteLine(b.GetType() == typeof(Giraffe));  // output: True
    }
}
```

This example demonstrates the difference between type checking performed with the typeof operator and the is operator.

?? and ??= - the null-coalescing operators

https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/null-coalescing-operator

---

The null-coalescing operator ?? returns the value of its left-hand operand if it isn't null; otherwise, it evaluates the right-hand operand and returns its result. The ?? operator doesn't evaluate its right-hand operand if the left-hand operand evaluates to non-null.

The null-coalescing assignment operator ??= assigns the value of its right-hand operand to its left-hand operand only if the left-hand operand evaluates to null. The ??= operator doesn't evaluate its right-hand operand if the left-hand operand evaluates to non-null.

??= is a perfect replacement for:

- if(myVar == null)
  {// do domething}