Statements

Statements cover a wide range of actions, like: show this button, send this e-mail and execute this and that code when a user clicks that button, and so on. However, simply executing these actions is not enough. You often need to execute some code only **when a certain *condition*** is true. Another important set of statements is the loops. **Loops** allow you to repeat a certain piece of code a number of times. For example, you can have a loop that goes from 1 to 10, performing some action on each iteration. The final important set of statements is the operators. **Operators** allow you to do something with your values; or, to be more exact, they allow you to operate on them. For example, you use operators to subtract values, concatenate (combine) them, or compare them to each other.

Operators

The most important operators can be grouped logically into five different types. These five groups, the assignment operators are probably the easiest to understand and use.

1- Assignment Operators

The assignment operators are used to assign a value to a variable. This value can come from many sources: a constant value, like the number 6, the value of another variable, and the result of an expression or a function. In its simplest form, an assignment looks like this:

int age = 36;

2- Arithmetic Operators

Arithmetic operators allow you to perform most of the familiar calculations on variables and values, like adding, subtracting, and dividing. The following table lists the common arithmetic operators.

C#	Usage
+	Adds two values to each other. These values can be numeric types like `Int32`, but also `String`, in which case they are concatenated.
-	Subtracts one value from another.
*	Multiplies two values.
/	Divides two values.
%	Divides two whole numbers and returns the remainder.

Example 1:

```
int firstNumber = 100;
float secondNumber = 23.5F;
```

```
double result = 0;
result = firstNumber + secondNumber; // Results in 123.5
result = firstNumber - secondNumber; // Results in 76.5
result = firstNumber * secondNumber; // Results in 2350
result = firstNumber / secondNumber; // Results in 4.25531914893617
```

Example 2:

```
result = Math.Pow(2, 3); // Results in 8
result = Math.Pow(3, 2); // Results in 9
```

Example 3:

```
int firstNumber = 17;
int secondNumber = 3;
int result;
result = firstNumber % secondNumber; // Results in 2.
```

3- Comparison Operators

A comparison operator always compares two values or expressions and then returns a Boolean value as the result. The following table lists the most common comparison operators.

C#	Usage
==	Checks if two values are equal to each other.
!=	Checks if two values are not equal.
<	Checks if the first value is less than the second.
>	Checks if the first value is greater than the second.
<=	Checks if the first value is less than or equal to the second.
>=	Checks if the first value is greater than or equal to the second.
is	In VB.NET: Compares two objects. In C#: Checks if a variable is of a certain type.

Example 1:

```
TextBox myTextBox = new TextBox();
if (myTextBox is TextBox)
{
// Run some code when myTextBox is a TextBox
}
```

4- Concatenation Operators

To concatenate two strings, you use the +. Additionally, you can use += to combine the concatenation and assignment operators. Consider this example:

```
string firstString = “Hello “;
string secondString = “World”;
string result;
// The following three blocks are all functionally equivalent
// and result in the value “Hello World”
result = firstString + secondString;
result = firstString;
result = result + secondString;
result = firstString;
result += secondString;
```

5- Logical Operators

The logical operators are used to combine the results of multiple individual expressions, and to make sure that multiple conditions are true or false, for example. The following table lists the most common logical operators.

C#	Usage
&&	Returns True when both expressions result in a True value
||	Returns True if at least one expression results in a True value.

Example:

```
int num1 = 3;
int num2 = 7;
if (num1 == 3 && num2 == 7) // Evaluates to true because both expressions are true
if (num1 == 2 && num2 == 7) // Evaluates to false because num1 is not 2
if (num1 == 3 || num2 == 11) // Evaluates to true because num1 is 3
if (!(num1 == 5)) // Evaluates to true because num1 is not 5
```

Making Decisions

Making decisions in an application is one of the most common things you do as a developer. For example, you need to hide a button on a Web Form when a user is not an administrator. Or you need to display the even rows in a table with a light grey background while the odd rows get a white background. All these decisions can be made with a few different logic constructs: If, If Else, ElseIf, and switch or Select Case statements.

If, If Else, and ElseIf Constructs

The **If statement** is the simplest of all decision making statements. It contains two relevant parts: the condition being tested and the code that is executed when the condition evaluates to True. For example:

```
if (User.IsInRole(“Administrators”))
{
btnDeleteArticle.Visible = true;
}
```

Often you want to perform a different action if the condition is not True. Using the negation operator Not or! You could simply write another statement:

```
if (User.IsInRole(“Administrators”))
{
btnDeleteArticle.Visible = true;
}
if (!User.IsInRole(“Administrators”))
{
btnDeleteArticle.Visible = false;
}
```

Clearly, this leads to messy code, as you need to repeat each expression evaluation twice: once for the True case and once for the False case. Fortunately, there is an easier solution: the **else**:

```
if (User.IsInRole(“Administrators”))
{
btnDeleteArticle.Visible = true;
}
else
{
btnDeleteArticle.Visible = false;
}
```

For simple conditions, this **if else** construct works fine. But consider a scenario where you have more than two options. In those scenarios you can use **else if**. Imagine that your site uses three different roles: administrators, content managers, and standard members.

Administrators can create and delete content; content managers can only create new content, whereas members can’t do either of the two. To show or hide the relevant buttons, you can use the following code:

```
if (User.IsInRole(“Administrators”))
{
btnCreateNewArticle.Visible = true;
btnDeleteArticle.Visible = true;
}
else if (User.IsInRole(“ContentManagers”))
{
btnCreateNewArticle.Visible = true;
btnDeleteArticle.Visible = false;
}
else if (User.IsInRole(“Members”))
{
btnCreateNewArticle.Visible = false;
btnDeleteArticle.Visible = false;
}
```

Switches Constructs

Imagine you’re building a web site for a concert hall that has shows on Saturday. During the week, visitors can buy tickets online for Saturday’s gig. To encourage visitors to buy tickets as early as possible, you decide to give them an early-bird discount. The earlier in the week they buy their tickets, the cheaper they are. Your code to calculate the discount rate can look like this, using **switch statement**:

```
DateTime today = DateTime.Now;
double discountRate = 0;
switch (today.DayOfWeek)
{
case DayOfWeek.Monday:
discountRate = 0.4;
break;
case DayOfWeek.Tuesday:
discountRate = 0.3;
break;
case DayOfWeek.Wednesday:
```

```
discountRate = 0.2;
break;
case DayOfWeek.Thursday:
discountRate = 0.1;
break;
default:
discountRate = 0;
break;
}
```

Creating a Simple Web-Based Calculator

In this exercise you will create a simple calculator that is able to add, subtract, multiply, and divide values. It shows you how to use some of the logical and assignment operators and demonstrates the If and **switch constructs**.

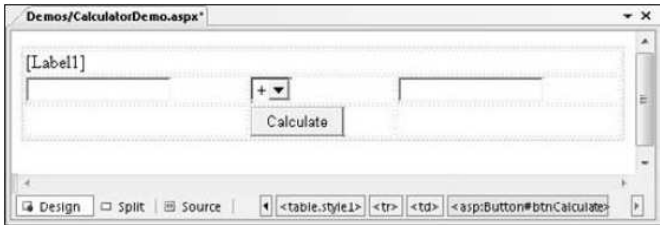

Figure 1: Calculator Page

```
protected void btnCalculate_Click(object sender, EventArgs e)
{
if (txtValue1.Text.Length > 0 && txtValue2.Text.Length > 0)
{
double result = 0;
double value1 = Convert.ToDouble(txtValue1.Text);
double value2 = Convert.ToDouble(txtValue2.Text);
switch (lstOperator.SelectedValue)
{
case "+":
result = value1 + value2;
break;
case "-":
result = value1 - value2;
break;
case "*":
result = value1 * value2;
break;
case "/":
result = value1 / value2;
break;
}
lblResult.Text = result.ToString();
}
else
{
lblResult.Text = string.Empty;
}
}
```

Loops

Loops are extremely useful in many applications, as they allow you to execute code repetitively, without the need to write that code more than once. For example, if you have a web site that needs to send news - letter by e-mail to its 20,000 subscribers, you write the code to send the newsletter once, and then use a loop that sends the newsletter to each subscriber the code finds in a database. Loops come as a few different types, each with their own usage and advantages.

The For Loop

The For loop simply repeats its code a predefined number of times. You define the exact number of iterations when you set up the loop. The For loop takes the following format:

```
for (startCondition; endCondition; step definition)
{
// Code that must be executed for each iteration}
for (int loopCount = 1; loopCount <= 10; loopCount ++)
{
Label1.Text += loopCount.ToString() + “<br />”;
}
```

The foreach Loop

The foreach simply iterate over all the items in a collection. Taking the roles array as an example, you can execute the following code to print each role name on the Label control:

```
foreach (string role in roles)
{
Label1.Text += role + “<br />”;
}
```

Since the roles variable is an array of strings, you need to set up the loop with a String as well. Likewise, if the collection that is being looped over contained Integer or Boolean data types, you would set up the loop with an Integer or Boolean, respectively.

The while Loop and do while

While loop is able to loop while a certain condition is true. While loop could potentially loop forever if you’re not careful. The following example shows how to use the While loop:

```
while (i>0)
{
Do something;
}
```

Besides the While loop, there are a few other alternatives, like the **do while loop** (that ensures that the code to be executed is always executed at least once).

```
protected void Button1_Click(object sender, EventArgs e)
    {
        int i=0;
        do
        {
            Response.Write("The value of i is: " + i + "</br>");
            i = i + 1;
        } while (i < 10);}
```