LECTURE15: C# PROGRAMMING BASICS (PART 2)

BUILD YOUR OWN ASP.NET 3.5 WEB SITE USING C# & VB

Outlines of today’s lecture

- In this lecture we will explore C# programming fundamentals such as:

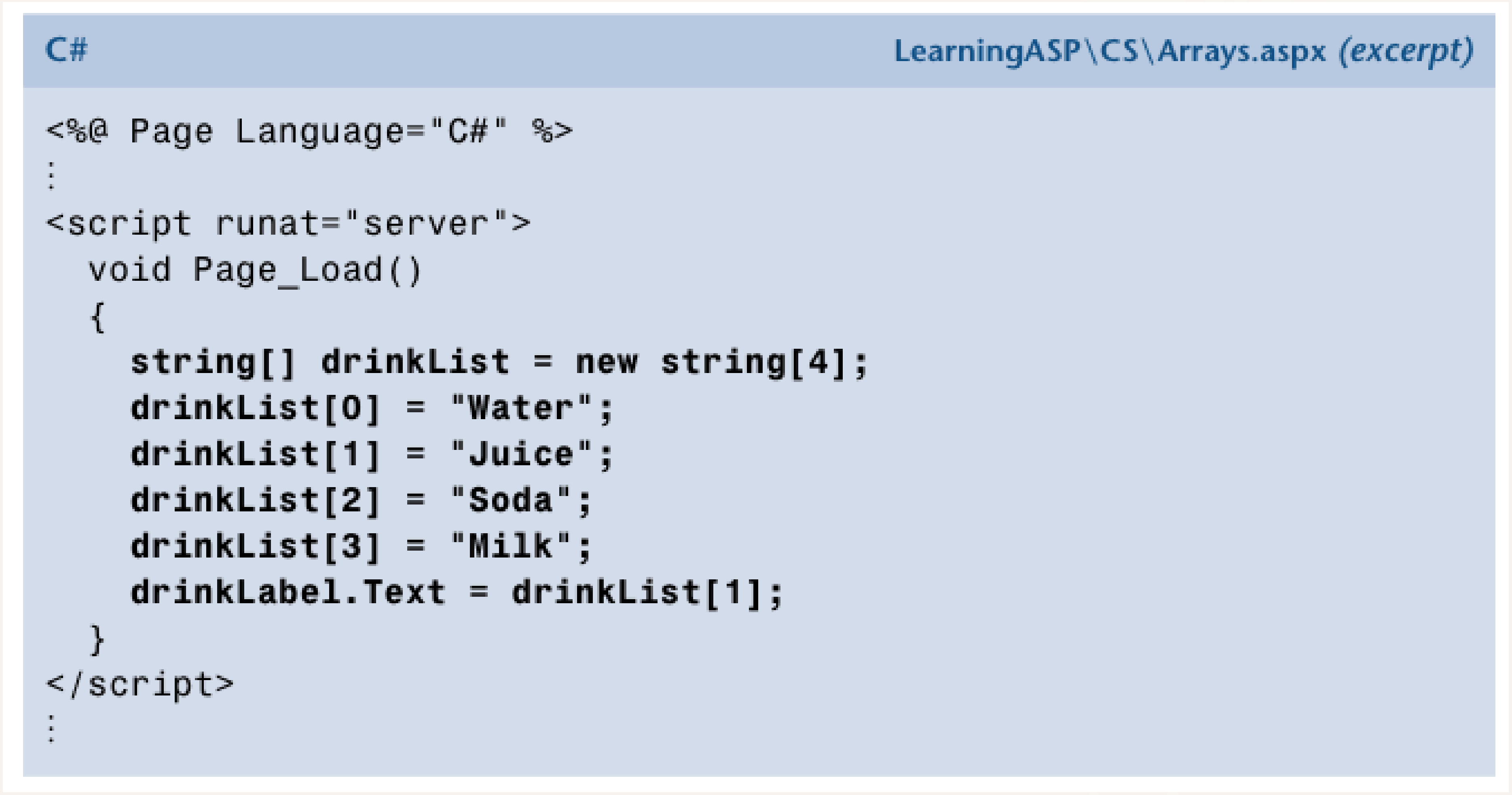
- Arrays
- Functions
- Operators
- Conditional statements
- Loops

Arrays

- Arrays are a special kind of variable that's tailored for storing related items of the same data type.

- Any one item in an array can be accessed using the array's name, followed by that item's position in the array (its offset).

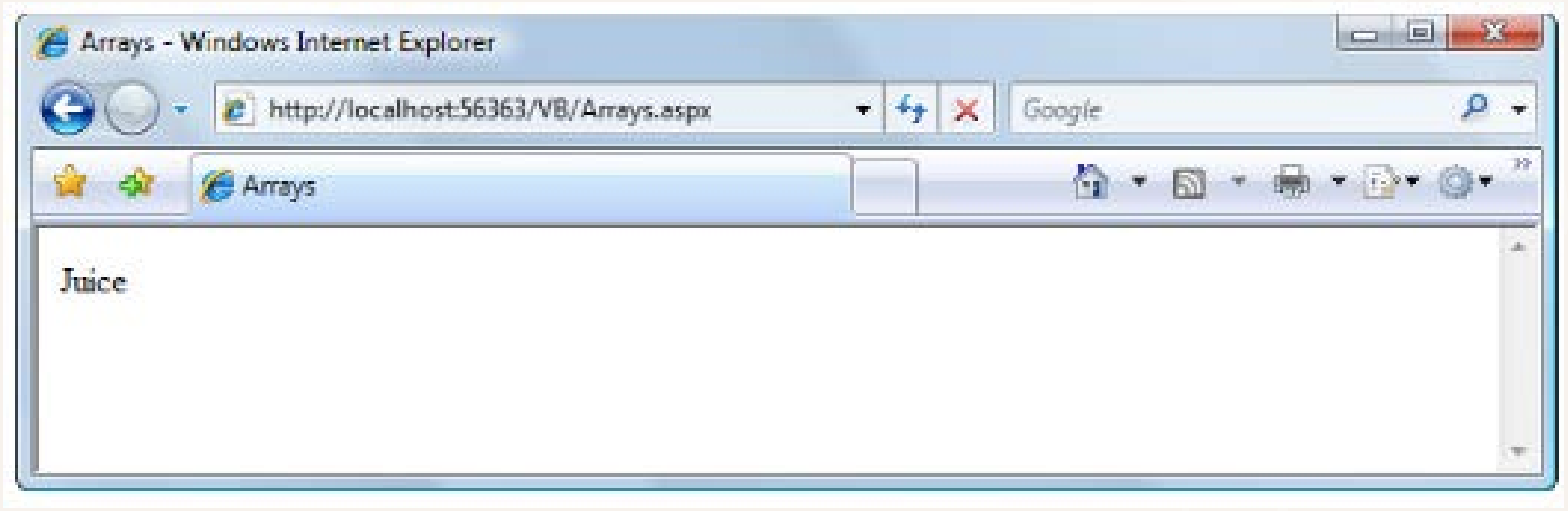
Array Example



C# LearningASP\CS\Arrays.aspx *(excerpt)*

```
<%@ Page Language="C#" %>
⋮
<script runat="server">
  void Page_Load()
  {
    string[] drinkList = new string[4];
    drinkList[0] = "Water";
    drinkList[1] = "Juice";
    drinkList[2] = "Soda";
    drinkList[3] = "Milk";
    drinkLabel.Text = drinkList[1];
  }
</script>
⋮
```

Array Example (Cont.)


Arrays - Windows Internet Explorer
http://localhost:56363/VB/Arrays.aspx
Google
Arrays

```
Juice
```

Functions

- **Functions** are very similar to subroutines, but for one key difference: they return a value.
- We simply have to specify the return type in place of void.
- The following code shows a simple example:

Function Example

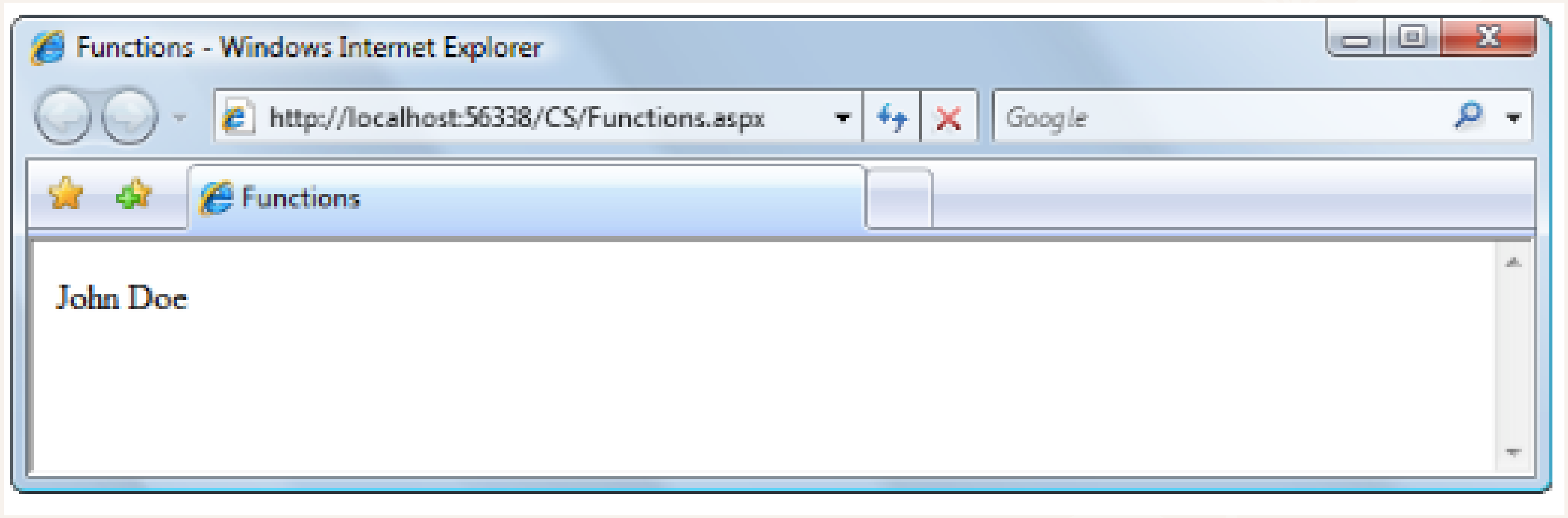
```
<%@ Page Language="C#" %>
⋮
<script runat="server">
  string getName()
  {
    return "John Doe";
  }

  void Page_Load()
  {
    messageLabel.Text = getName();
  }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <title>ASP.NET Functions</title>
  </head>
  <body>
    <form id="form1" runat="server">

    <div>
        <asp:Label id="messageLabel" runat="server" />
    </div>
    </form>
  </body>
</html>
```

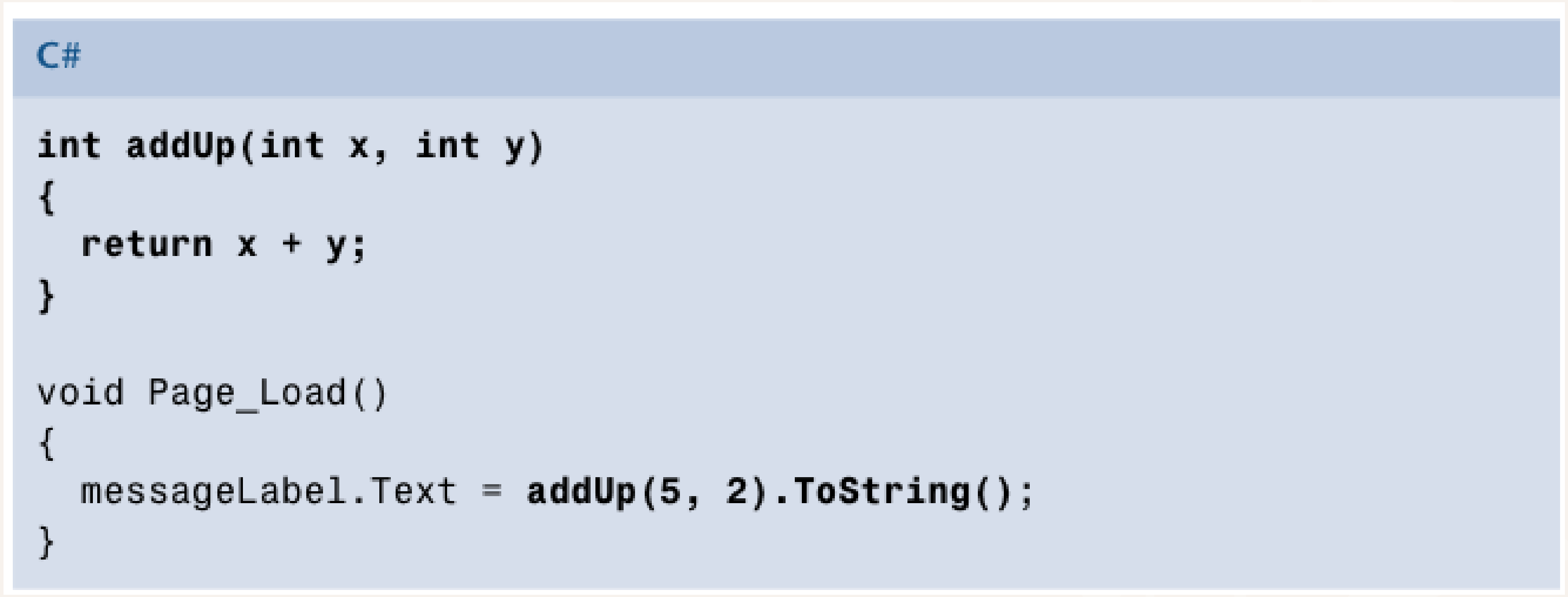
Function Example (Cont.)


Functions - Windows Internet Explorer
http://localhost:56338/CS/Functions.aspx
Google
Functions

```
John Doe
```

A function that adds two integer numbers?

Function– another example


C#

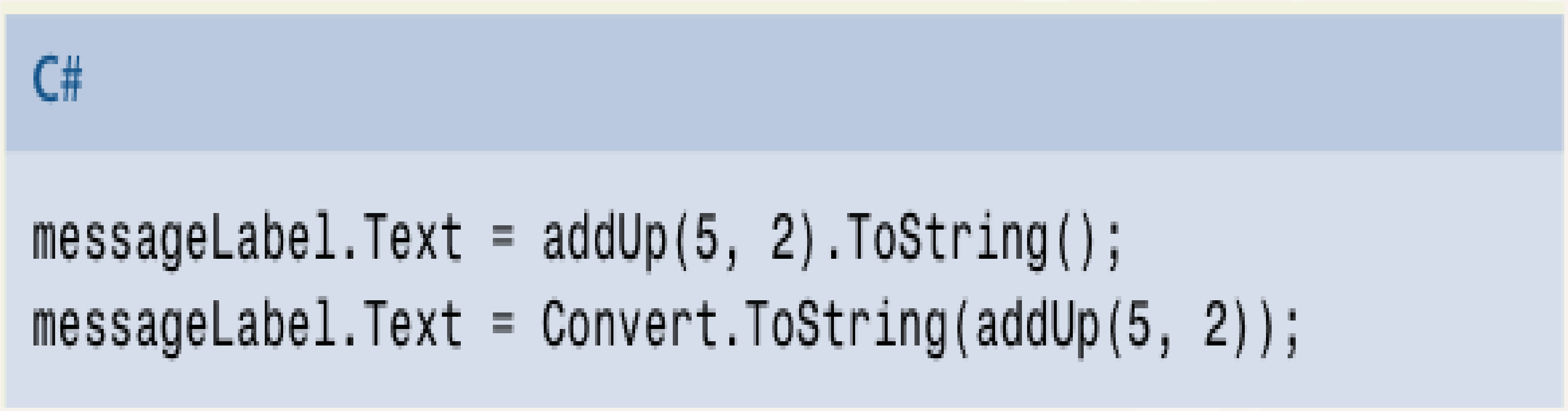
```
int addUp(int x, int y)
{
  return x + y;
}

void Page_Load()
{
  messageLabel.Text = addUp(5, 2).ToString();
}
```

Demo!

Converting Numbers to Strings

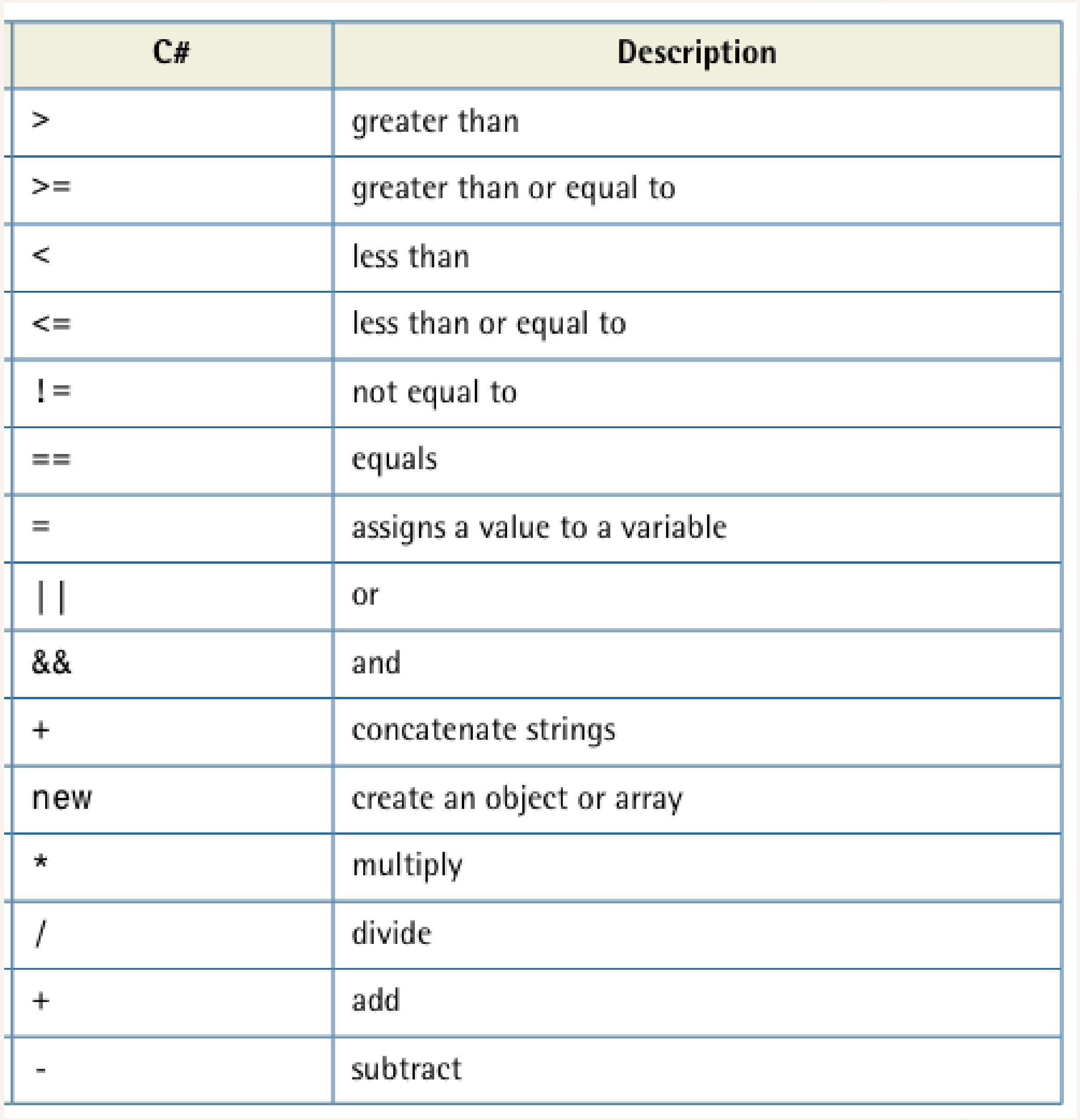
- There are other ways to convert numbers to strings in .NET.



C#

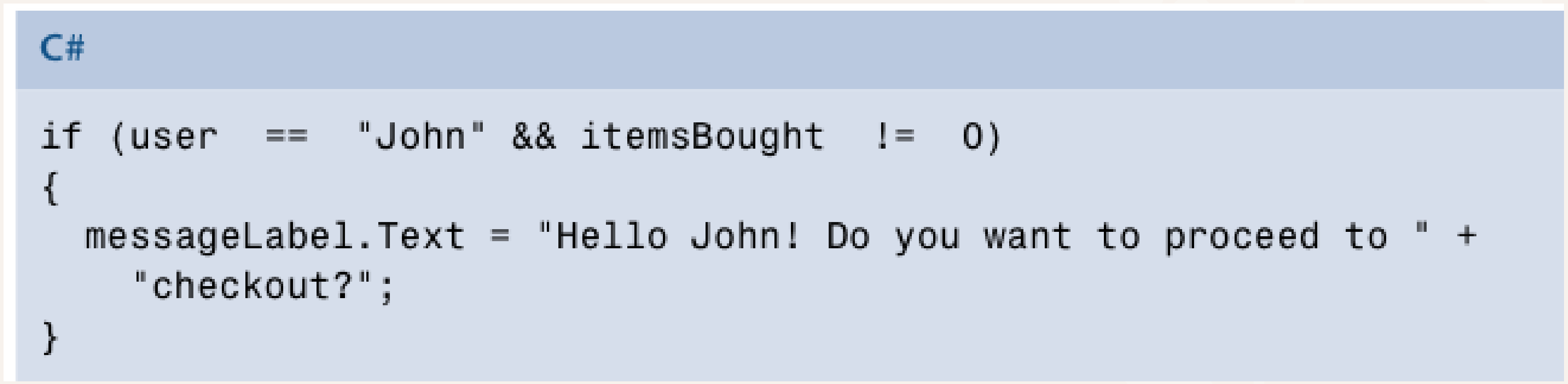
```
messageLabel.Text = addUp(5, 2).ToString();
messageLabel.Text = Convert.ToString(addUp(5, 2));
```

Operators



C#	Description
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
!=	not equal to
==	equals
=	assigns a value to a variable
||	or
&&	and
+	concatenate strings
new	create an object or array
*	multiply
/	divide
+	add
-	subtract

Operators–example



C#

```
if (user  ==  "John" && itemsBought  !=  0)
{
  messageLabel.Text = "Hello John! Do you want to proceed to " +
    "checkout?";
}
```

Breaking Long Lines of Code

C#

```
if (user  ==  "John" && itemsBought  !=  0)
{
  messageLabel.Text = "Hello John! Do you want to proceed to " +
    "checkout?";
}
```

- Since the message string in the above example was too long to fit on one line, we used the string concatenation operator (+) to combine two shorter strings on separate lines to form the complete message.

Conditional Logic

- As you develop ASP.NET applications, there will be many instances in which you'll need to perform an action only if a certain condition is met.

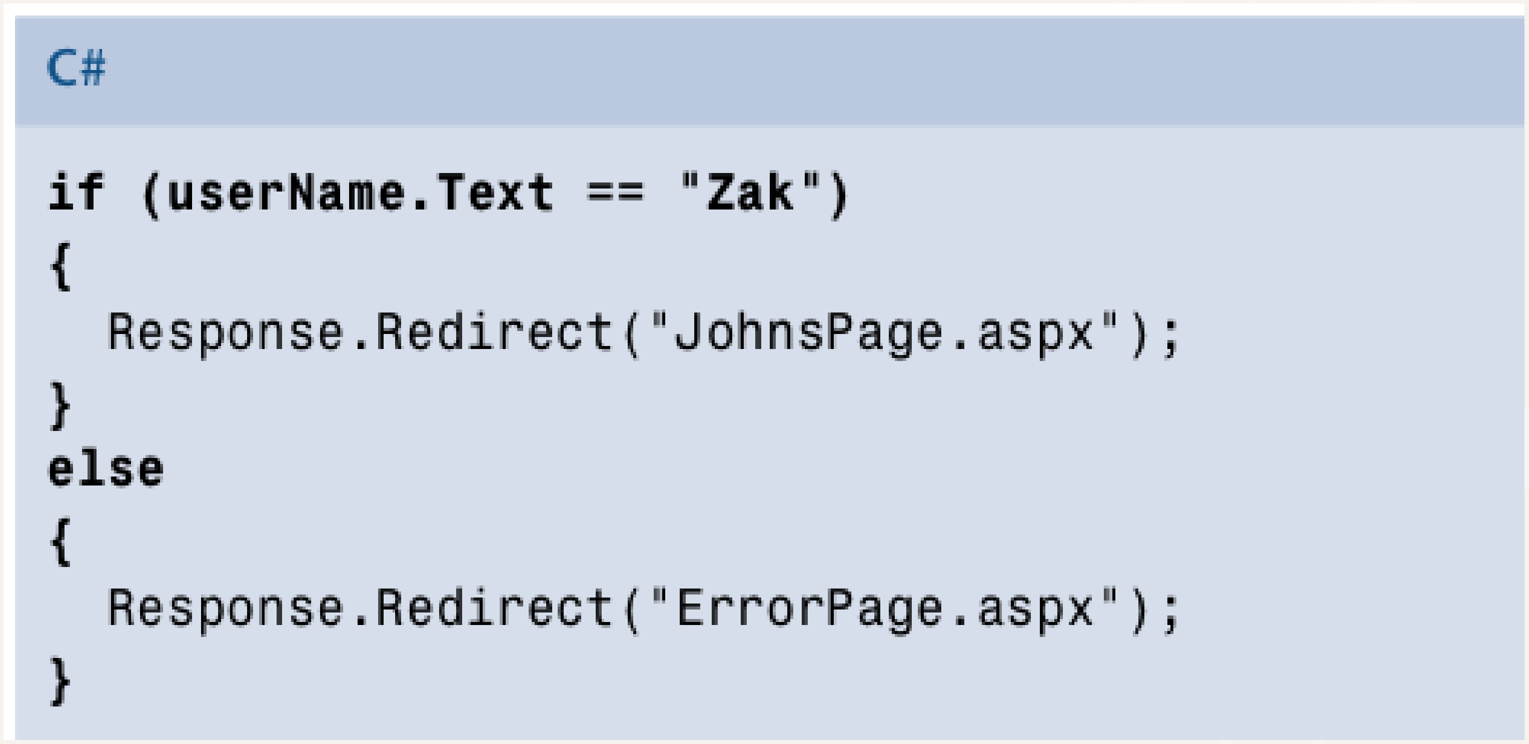
- We check for such occurrences using **conditionals**—statements that execute different code branches based upon a specified condition.

- The simplest of which is probably the If statement.

- This statement is often used in conjunction with an else statement, which specifies what should happen if the condition is not met.

Conditional Logic–if statement

- We may wish to check whether or not the name entered in a text box is Zak, redirecting the user to a welcome page if it is, or to an error page if it's not:

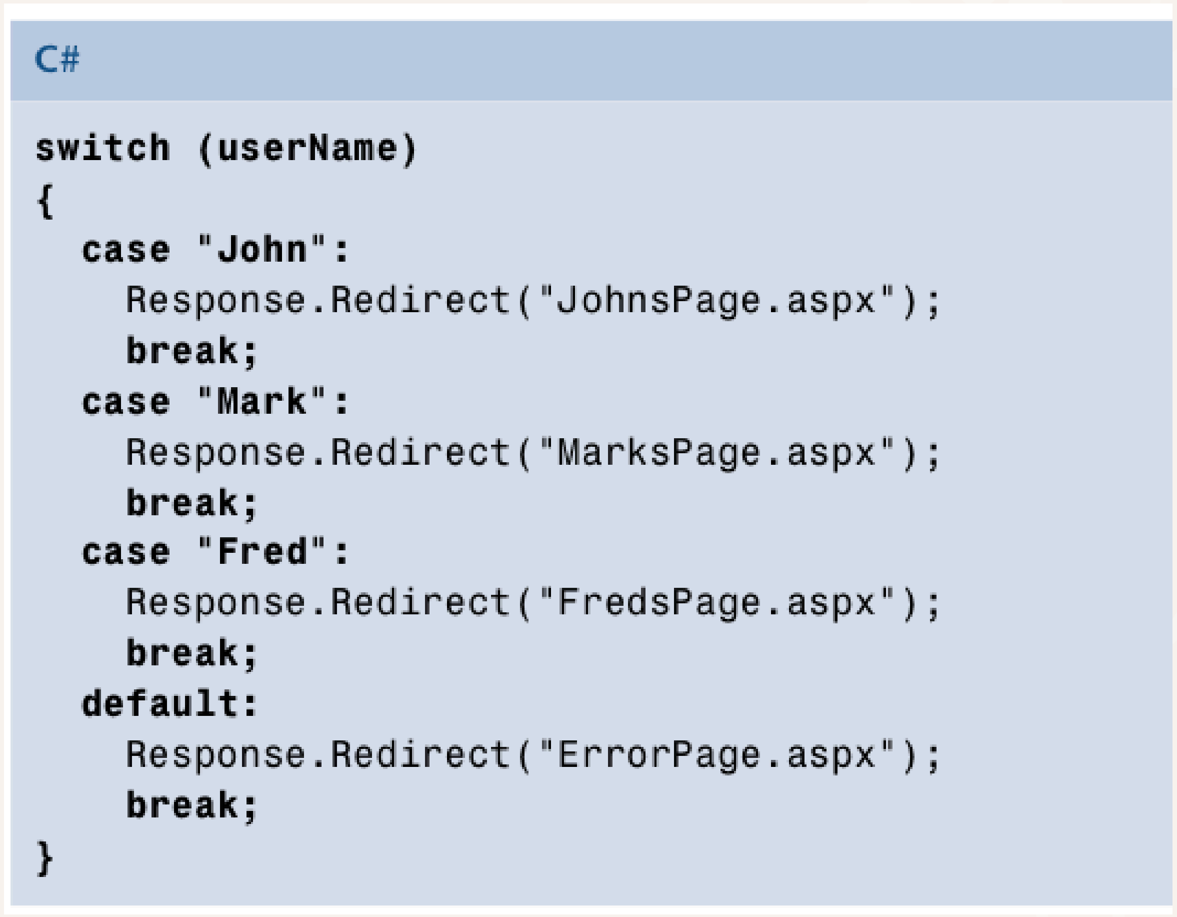


C#

```
if (userName.Text == "Zak")
{
  Response.Redirect("JohnsPage.aspx");
}
else
{
  Response.Redirect("ErrorPage.aspx");
}
```

Conditional Logic–switch statement

- Often, we want to check for many possibilities, and specify that our application performs a particular action in each case. To achieve this, we use the switch construct, as follows:


C#

```
switch (userName)
{
  case "John":
    Response.Redirect("JohnsPage.aspx");
    break;
  case "Mark":
    Response.Redirect("MarksPage.aspx");
    break;
  case "Fred":
    Response.Redirect("FredsPage.aspx");
    break;
  default:
    Response.Redirect("ErrorPage.aspx");
    break;
}
```

Loops

- Loops cause a code block to execute repeatedly for as long as the test expression remains true.

- There are two basic kinds of loop:
  - While loops.
  - For loops, including For Each.

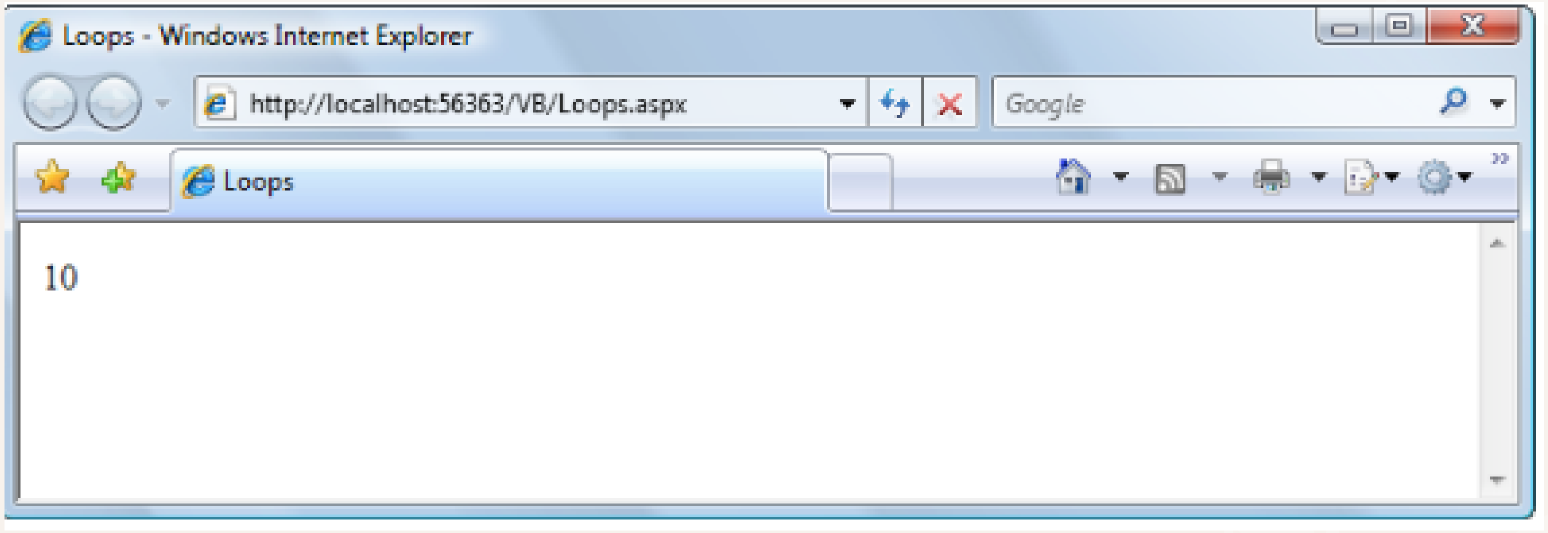
Loops–while loop

- A While loop is the simplest form of loop; it makes a block of code repeat for as long as a particular condition is true

C# LearningASP\CS\Loops.aspx

```
<%@ Page Language="C#" %>
⋮
<script runat="server">
  void Page_Load()
  {
    int counter = 0;
    while (counter <= 10)
    {
      messageLabel.Text = counter.ToString();
      counter++;
    }
  }
</script>
⋮
```

Loops–while loop (Cont.)


Loops - Windows Internet Explorer
http://localhost:56363/VB/Loops.aspx
Google
Loops

```
10
```

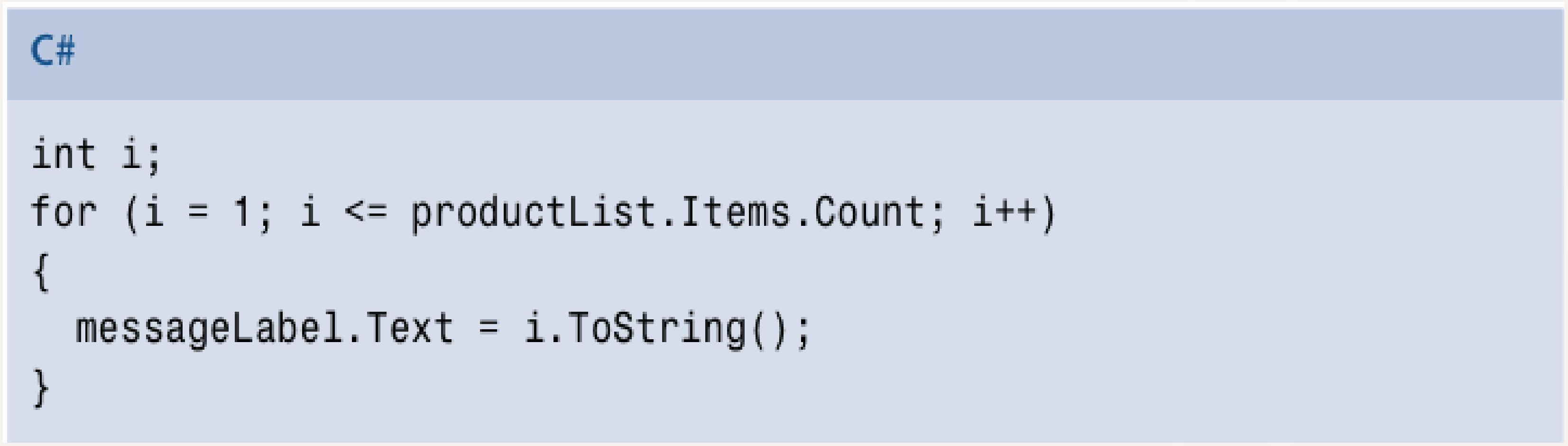
Loops–do while loop

C#

```
void Page_Load()
{
  int counter = 0;
  do
  {
    messageLabel.Text = counter.ToString();
    counter++;
  }
  while (counter <= 10);
}
```

Loops–for loop

- A For loop is similar to a While loop, but we typically use it when we know in advance how many times we need it to execute. The following example displays the count of items within a DropDownList control called productList:

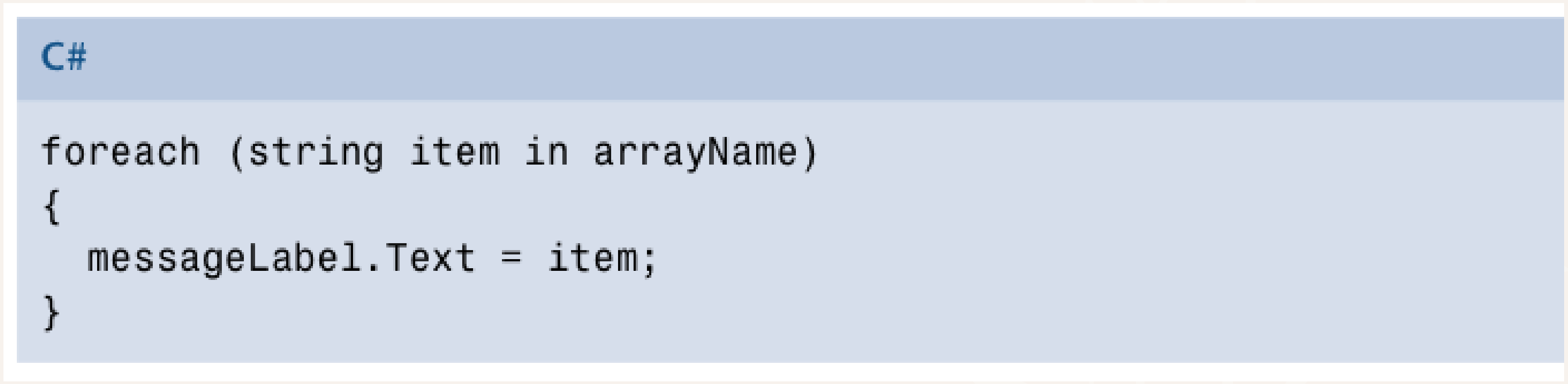


C#

```
int i;
for (i = 1; i <= productList.Items.Count; i++)
{
  messageLabel.Text = i.ToString();
}
```

Loops–for each loop

- The other type of For loop is For Each (foreach in C#), which loops through every item within a collection. The following example loops through an array called arrayName:



C#

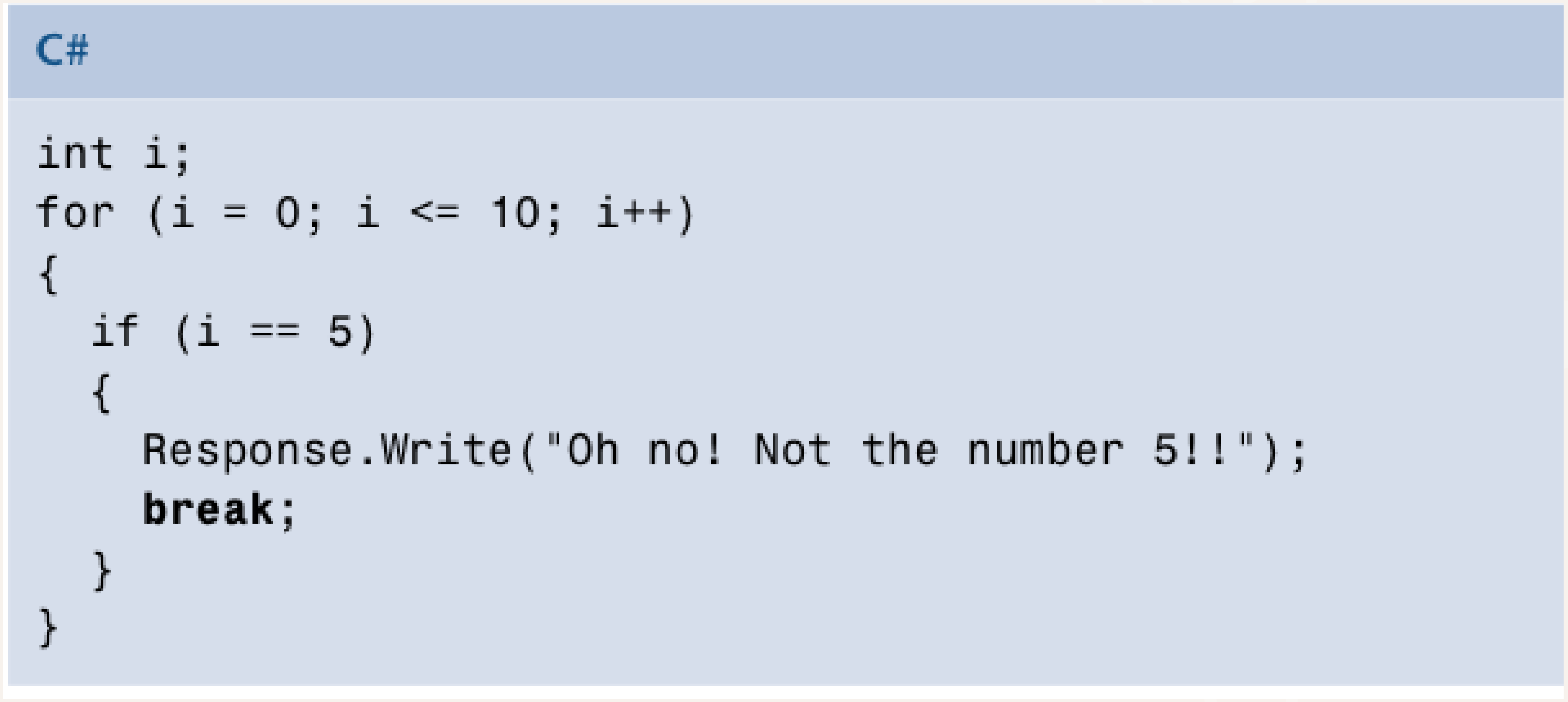
```
foreach (string item in arrayName)
{
  messageLabel.Text = item;
}
```

For vs. foreach

- The important difference between a For loop and a For Each loop involves what happens to the variable we supply to the loop.
- In a For loop, the variable represents a counter—a number which starts at a predefined initial value and is incremented until it reaches a predefined maximum value.
- The counter is incremented every time the code in the For loop is executed.
- In a For Each loop the variable represents the current object from the given collection.
- The variable can be any kind of object, including a string, a date, or a custom object that you created (more about these just a bit later!).
- The object reference changes to the next item in the collection each time the code in the For Each loop executes.

Loop Termination–break

- You may also come across instances in which you need to exit a loop. In these cases, you can use the break statement in C# to terminate the loop:



C#

```
int i;
for (i = 0; i <= 10; i++)
{
  if (i == 5)
  {
    Response.Write("Oh no! Not the number 5!!");
    break;
  }
}
```