RAZOR

Microsoft ASP.NET ortamına sadece MVC mimarisini taşımakla kalmamış, MVC 3'le birlikte yeni bir görünüm motoru (view engine) olan Razor'u da kazandırmıştır. Razor uygulama kodları (C# veya Visual Basic) ile, html'in iç içe kullanılmasını sağlar. Klasik Asp gibi,php gibi sayfa içinde kodları serpiştirerek uygulama geliştirmek mümkündür.

WebForms ise uzun yıllardır kullanılan ASPX görünüm motorudur. Tüm kodlar <% ve %> işaretleri arasına yazılır.

Razor, yeni bir gramere sahip görünüm motorudur. Yeni gramerde bildiğimiz html ve de C# veya Visual Basic kullanılacağı için, kullanıcılar öğrenmek için zorlanmayacaktır. Razor :

- Sunucu tabanlı kodu web sayfalarına eklemek için kullanılan, az yer kaplayan, anlamlı ve akıcı bir yapıya sahiptir.
- C# veya Visual Basic kullanılacağı için, yeni bir dil öğrenme zorunluğundan kurtarır.
- Öğrenmesi çok kolaydır.
- Visual Studio ile Intellisense desteği mevcuttur.
- Unit-test'lerine uygun bir yapısı mevcuttur.

Razor Yazım Kuralları

Razor kod blokları, **@** işareti ile başlayan blok parantezleri arasına yazılır. **@{……}**

İnline ifadeler (değişkenler ve fonksiyonlar) **@** işareti ile başlamalıdır.

Kod deyimleri **;** (noktalı virgül) işareti ile son bulur.

Değişkenler **var** deyimiyle deklere edilirler.

C# dili kullanıldığında dosya uzantısı **.cshtml**, Visual Basic kullanıldığında **.vbhtml** 'dir.

What is Razor?

Razor is not a programming language. It's a server side markup language.

Razor is a markup syntax that lets you embed server-based code (Visual Basic and C#) into web pages.

Server-based code can create dynamic web content on the fly, while a web page is written to the browser. When a web page is called, the server executes the server-based code inside the page before it returns the page to the browser. By running on the server, the code can perform complex tasks, like accessing databases.

Razor is based on ASP.NET, and designed for creating web applications. It has the power of traditional ASP.NET markup, but it is easier to use, and easier to learn.

Razor Syntax

Razor uses a syntax very similar to PHP and Classic ASP.

Razor:

```
<ul>
@for (int i = 0; i < 10; i++) {
<li>@i</li>
}
</ul>
```

PHP:

```
<ul>
<?php
for ($i = 0; $i < 10; $i++) {
echo("<li>$i</li>");
}
?>
</ul>
```

Classic ASP:

```
<ul>
<%for i = 0 to 10%>
<li><%=i%></li>
<%next%>
</ul>
```

Razor Programming Languages

Razor supports both C# (C sharp) and VB (Visual Basic).

Razor supports both C# (C sharp) and VB (Visual Basic).

Main Razor Syntax Rules for C#

- Razor code blocks are enclosed in @{ ... }
- Inline expressions (variables and functions) start with @
- Code statements end with semicolon
- Variables are declared with the var keyword
- Strings are enclosed with quotation marks
- C# code is case sensitive
- C# files have the extension .cshtml

```
<html>
<body>
<!-- Single statement block -->
@{ var myMessage = "Hello World"; }

<!-- Inline expression or variable -->
<p>The value of myMessage is: @myMessage</p>

<!-- Multi-statement block -->
@{
var greeting = "Welcome to our site!";
var weekDay = DateTime.Now.DayOfWeek;
var greetingMessage = greeting + " Here in Huston it is: " + weekDay;
}

<p>The greeting is: @greetingMessage</p>
</body>
</html>
```

```
The value of myMessage is: Hello World

The greeting is: Welcome to our site! Here in Huston it is: Saturday
```

How Does it Work?

Razor is a simple programming syntax for embedding server code in web pages.

Razor syntax is based on the ASP.NET framework, the part of the Microsoft.NET Framework that's specifically designed for creating web applications.

The Razor syntax gives you all the power of ASP.NET, but is using a simplified syntax that's easier to learn if you're a beginner, and makes you more productive if you're an expert.

Razor web pages can be described as HTML pages with two kinds of content: HTML content and Razor code.

When the server reads the page, it runs the Razor code first, before it sends the HTML page to the browser. The code that is executed on the server can perform tasks that cannot be done in the browser, for example accessing a server database. Server code can create dynamic HTML content on the fly, before it is sent to the browser. Seen from the browser, the HTML generated by server code is no different than static HTML content.

ASP.NET web pages with Razor syntax have the special file extension cshtml (Razor using C#) or vbhtml (Razor using VB).

Working With Objects

Server coding often involves objects.

The "DateTime" object is a typical built-in ASP.NET object, but objects can also be self-defined, a web page, a text box, a file, a database record, etc.

Objects may have methods they can perform. A database record might have a "Save" method, an image object might have a "Rotate" method, an email object might have a "Send" method, and so on.

Objects also have properties that describe their characteristics. A database record might have a FirstName and a LastName property (amongst others).

The ASP.NET DateTime object has a Now property (written as DateTime.Now), and the Now property has a Day property (written as DateTime.Now.Day). The example below shows how to access some properties of the DateTime object:

```
<table border="1">
<tr>
<th width="100px">Name</th>
<td width="100px">Value</td>
</tr>
<tr>
<td>Day</td><td>@DateTime.Now.Day</td>
</tr>
<tr>
<td>Hour</td><td>@DateTime.Now.Hour</td>
</tr>
<tr>
<td>Minute</td><td>@DateTime.Now.Minute</td>
</tr>
<tr>
<td>Second</td><td>@DateTime.Now.Second</td>
</tr>
</td>
</table>
```

If and Else Conditions

An important feature of dynamic web pages is that you can determine what to do based on conditions.

The common way to do this is with the if ... else statements:

Example

```
@{
var txt = "";
if(DateTime.Now.Hour > 12)
  {txt = "Good Evening";}
else
  {txt = "Good Morning";}
}
```

Reading User Input

Another important feature of dynamic web pages is that you can read user input.

Input is read by the Request[] function, and posting (input) is tested by the IsPost condition:

Example

```
@{
var totalMessage = "";
if(IsPost)
   {
   var num1 = Request["text1"];
   var num2 = Request["text2"];
   var total = num1.AsInt() + num2.AsInt();
   totalMessage = "Total = " + total;
   }
}
<html>
<body style="background-color: beige; font-family: Verdana, Arial;">
<form action="" method="post">
<p><label for="text1">First Number:</label><br>
<input type="text" name="text1" /></p>
<p><label for="text2">Second Number:</label><br>
<input type="text" name="text2" /></p>
<p><input type="submit" value=" Add " /></p>
</form>
<p>@totalMessage</p>
</body>
</html>
```

ASP.NET Razor - C# Variables

Variables are named entities used to store data.

Variables

Variables are used to store data.

The name of a variable must begin with an alphabetic character and cannot contain whitespace or reserved characters.

A variable can be of a specific type, indicating the kind of data it stores. String variables store string values ("Welcome to W3Schools"), integer variables store number values (103), date variables store date values, etc.

Variables are declared using the var keyword, or by using the type (if you want to declare the type), but ASP.NET can usually determine data types automatically.

Examples

```
// Using the var keyword:
var greeting = "Welcome to W3Schools";
var counter = 103;
var today = DateTime.Today;

// Using data types:
string greeting = "Welcome to W3Schools";
int counter = 103;
DateTime today = DateTime.Today;
```

Data Types

Below is a list of common data types:

Type	Description	Examples
int	Integer (whole numbers)	103, 12, 5168
float	Floating-point number	3.14, 3.4e38
decimal	Decimal number (higher precision)	1037.196543
bool	Boolean	true, false
string	String	"Hello W3Schools", "John"

Operators

An operator tells ASP.NET what kind of command to perform in an expression.

The C# language supports many operators. Below is a list of common operators:

Operator	Description	Example
=	Assigns a value to a variable.	i=6
+ - * /	Adds a value or variable. Subtracts a value or variable. Multiplies a value or variable. Divides a value or variable.	i=5+5 i=5-5 i=5*5 i=5/5
+= -=	Increments a variable. Decrements a variable.	i += 1 i -= 1
==	Equality. Returns true if values are equal.	if (i==10)
!=	Inequality. Returns true if values are not equal.	if (i!=10)

<
>
<=
>=
	Less than.
	Greater than.
	Less than or equal.
	Greater than or equal.
		if (i<10)
		if (i>10)
		if (i<=10)
		if (i>=10)

+	Adding strings (concatenation).	"w3" + "schools"
.	Dot. Separate objects and methods.	DateTime.Hour

()	Parenthesis. Groups values.	(i+5)
()	Parenthesis. Passes parameters.	x=Add(i,5)
[]	Brackets. Accesses values in arrays or collections.	name[3]
!	Not. Reverses true or false.	if (!ready)
&& ||	Logical AND. Logical OR.	if (ready && clear) if (ready || clear)

Converting Data Types

Converting from one data type to another is sometimes useful.

The most common example is to convert string input to another type, such as an integer or a date.

As a rule, user input comes as strings, even if the user entered a number. Therefore, numeric input values must be converted to numbers before they can be used in calculations.

Below is a list of common conversion methods:

Method	**Description**	**Example**
AsInt() IsInt()	Converts a string to an integer.	if (myString.IsInt()) {myInt=myString.AsInt();}

AsFloat() IsFloat()	Converts a string to a floating-point number.	if (myString.IsFloat()) {myFloat=myString.AsFloat();}
AsDecimal() IsDecimal()	Converts a string to a decimal number.	if (myString.IsDecimal()) {myDec=myString.AsDecimal();}
AsDateTime() IsDateTime()	Converts a string to an ASP.NET DateTime type.	myString="10/10/2012"; myDate=myString.AsDateTime();
AsBool() IsBool()	Converts a string to a Boolean.	myString="True"; myBool=myString.AsBool();
ToString()	Converts any data type to a string.	myInt=1234; myString=myInt.ToString();

ASP.NET Razor - C# Loops and Arrays

Statements can be executed repeatedly in loops.

For Loops

If you need to run the same statements repeatedly, you can program a loop.

If you know how many times you want to loop, you can use a **for loop**. This kind of loop is especially useful for counting up or counting down:

Example

```
<html>
<body>
@for(var i = 10; i < 21; i++)
   {<p>Line @i</p>}
</body>
</html>
```

For Each Loops

If you work with a collection or an array, you often use a **for each loop**.

A collection is a group of similar objects, and the for each loop lets you carry out a task on each item. The for each loop walks through a collection until it is finished.

The example below walks through the ASP.NET Request.ServerVariables collection.

Example

```
<html>
<body>
<ul>
@foreach (var x in Request.ServerVariables)
   {<li>@x</li>}
</ul>
</body>
</html>
```

While Loops

The **while loop** is a general purpose loop.

A while loop begins with the while keyword, followed by parentheses, where you specify how long the loop continues, then a block to repeat.

While loops typically add to, or subtract from, a variable used for counting.

In the example below, the += operator adds 1 to the variable i, each time the loop runs.

Example

```
<html>
<body>
@{
var i = 0;
while (i < 5)
   {
   i += 1;
   <p>Line @i</p>
   }
}
</body>
</html>
```

Arrays

An array is useful when you want to store similar variables but don't want to create a separate variable for each of them:

Example

```
@{
string[] members = {"Jani", "Hege", "Kai", "Jim"};
int i = Array.IndexOf(members, "Kai")+1;
int len = members.Length;
string x = members[2-1];
}
<html>
<body>
<h3>Members</h3>
@foreach (var person in members)
{
<p>@person</p>
}
<p>The number of names in Members are @len</p>
<p>The person at position 2 is @x</p>
<p>Kai is now in position @i</p>
</body>
</html>
```

ASP.NET Razor - C# Logic Conditions

Programming Logic: Execute code based on conditions.

The If Condition

C# lets you execute code based on conditions.

To test a condition you use an **if statement**. The if statement returns true or false, based on your test:

- The if statement starts a code block
- The condition is written inside parenthesis
- The code inside the braces is executed if the test is true

Example

```
@{var price=50;}
<html>
<body>
@if (price>30)
   {
   <p>The price is too high.</p>
   }
</body>
</html>
```

The Else Condition

An if statement can include an **else condition**.

The else condition defines the code to be executed if the condition is false.

Example

```
@{var price=20;}
<html>
<body>
@if (price>30)
  {
  <p>The price is too high.</p>
  }
else
  {
  <p>The price is OK.</p>
  }
</body>
</html>
```

Note: In the example above, if the first condition is true, it will be executed. The else condition covers "everything else".

The Else If Condition

Multiple conditions can be tested with an **else if condition**:

Example

```
@{var price=25;}
<html>
<body>
@if (price>=30)
 {
 <p>The price is high.</p>
 }
else if (price>20 && price<30)
 {
 <p>The price is OK.</p>
 }
else
 {
 <p>The price is low.</p>
 }
</body>
</html>
```

In the example above, if the first condition is true, it will be executed.

If not, then if the next condition is true, this condition will be executed.

You can have any number of else if conditions.

If none of the if and else if conditions are true, the last else block (without a condition) covers "everything else".

Switch Conditions

A **switch block** can be used to test a number of individual conditions:

Example

```
@{
var weekday=DateTime.Now.DayOfWeek;
var day=weekday.ToString();
var message="";
}
<html>
<body>
@switch(day)
{
case "Monday":
   message="This is the first weekday.";
   break;
case "Thursday":
   message="Only one day before weekend.";
   break;
case "Friday":
   message="Tomorrow is weekend!";
   break;
default:
   message="Today is " + day;
   break;
}
<p>@message</p>
</body>
</html>
```

The test value (day) is in parentheses. Each individual test condition has a case value that ends with a colon, and any number of code lines ending with a break statement. If the test value matches the case value, the code lines are executed.

A switch block can have a default case (default:) for "everything else" that runs if none of the cases are true.