Razor Syntax in Blazor

Razor Syntax in Blazor

Razor is a **templating syntax** that allows you to combine **HTML markup** with **C# code** inside .razor files.
Blazor uses Razor to build **interactive UI components**.

A Razor component is compiled into a C# class at build time.

---

1. The @ Symbol (Core of Razor)

The @ symbol tells the Razor engine:

"Switch from HTML to C#"

Example

```
<h3>Hello @UserName</h3>


@code {
   string UserName = "San";
}
```

Here:

- HTML → <h3>
- C# → @UserName

---

2. Inline Expressions

Use @ to evaluate a **single C# expression**.

```
<p>Current Year: @DateTime.Now.Year</p>

<p>Sum: @(10 + 20)</p>
```

Use parentheses () for complex expressions.

---

3. Code Blocks (@{ })

Use @{ } for **multiple C# statements**.

```
@{
   var message = "Welcome to Blazor";
   var year = DateTime.Now.Year;
}

<p>@message - @year</p>
```

---

4. @code Block (Component Logic)

All **component logic, state, and methods** go inside @code.

```
<button @onclick="Increment">Click Me</button>
<p>Count: @count</p>

@code {
   int count = 0;

   void Increment()
   {
      count++;
   }
}
```

This is equivalent to a **code-behind class**.

---

5. Conditional Rendering (@if, @else)

```
@if (isLoggedIn)
```

```
{
   <p>Welcome, User!</p>
}
else
{
   <p>Please log in.</p>
}

@code {
   bool isLoggedIn = true;
}
```

Razor allows **C# control flow + HTML together**.

---

6. Looping (@foreach, @for)

@foreach

```
<ul>
@foreach (var name in Names)
{
   <li>@name</li>
}
</ul>

@code {
   List<string> Names = new() { "A", "B", "C" };
}
```

@for

```
@for (int i = 1; i <= 5; i++)
{
   <p>Item @i</p>
}
```

---

7. Data Binding

One-way Binding

```
<p>@Message</p>

@code {
   string Message = "Hello";
}
```

Two-way Binding (@bind)

```
<input @bind="UserName" />
<p>Hello @UserName</p>

@code {
   string UserName = "";
}
```

By default, Blazor updates on onchange (when focus is lost).

Immediate update:

```
<input @bind="UserName" @bind:event="oninput" />
```

---

8. Event Handling

```
<button @onclick="SayHello">Click</button>
```

```
@code {
    void SayHello()
    {
        Console.WriteLine("Hello");
    }
}
```

With Lambda

```
<button @onclick="() => DeleteItem(1)">Delete</button>
```

---

9. Attribute Binding

```
<button disabled="@isDisabled">Submit</button>


@code {
    bool isDisabled = true;
}
```

Conditional attributes:

```
<button class="@GetCss()">Save</button>


@code {
    string GetCss() => "btn btn-primary";
}
```

---

10. Rendering Raw HTML (MarkupString)

By default, Blazor **HTML-encodes** output.

```
<p>@rawHtml</p>
```

```
@code {
    string rawHtml = "<b>Bold Text</b>";
}
```

To render as HTML:

```
<p>@((MarkupString)rawHtml)</p>
```

---

11. Comments in Razor

Razor Comment (not rendered)

```
@* This is a Razor comment *@
```

HTML Comment (sent to browser)

```
<!-- HTML Comment -->
```

---

12. Mixing HTML and C# (Important Rule)

❌ This is invalid:

```
@if (true)
    <p>Hello</p>
```

✅ Correct:

```
@if (true)
{
    <p>Hello</p>
}
```

---

13. Directive Syntax (@page, @using, @inject)

@page

```
@page "/counter"
```

@using

```
@using System.Net.Http
```

@inject

```
@inject HttpClient Http
```

---

14. Razor File Structure (Mental Model)

```
@page "/example"
@using Some.Namespace
@inject SomeService Service

<h3>UI Markup</h3>

<button @onclick="Action">Click</button>

@code {
   // State
   // Methods
   // Lifecycle methods
}
```