What is HTML Helper in ASP.NET MVC 5?

* HTML Helpers are methods that return a string.
* Helper class can create HTML controls programmatically. HTML Helpers are used in View to render HTML content.
* It is not mandatory to use HTML Helper classes for building an ASP.NET MVC application.
* We can build an ASP.NET MVC application without using them, but HTML Helpers helps in the rapid development of a view.
* HTML Helpers are more lightweight as compared to ASP.NET Web Form controls as they do not use ViewState and do not have event models.
* MVC has built-in Helpers methods
* We can create custom HTML helpers.

Type of HTML Helpers?

HTML Helpers are categorized into three types,

1. Inline HTML helpers
2. Built-in HTML helpers
   1. Standard HTML Helpers
   2. Strongly Typed HTML helpers
   3. Templated HTML helpers
3. Custom HTML helpers

What is Inline HTML Helper in MVC 5?

* Inline HTML Helper is used to create a reusable Helper method by using the Razor @helper tag.
* Inline helpers can be reused only on the same view.
* We can not use Inline helper to the different view Pages.
* We can create our own Inline helper method based on our requirements.

Advantages of using Inline HTML Helper in MVC 5

* It is reusable on the same view.
* It reduces the code repetition
* It is simple to create and easy to use.
* It is easy to customization the method based on the requirement.

**Syntax of Inline Html Helper**

@helper HelperName(Parameters list..)

{

// code.....

}

Example of Inline HTML helpers

After creating the application, we need to create one Index view and add the following code.

**Example 1**

We can create inline HTML Helper method having integer parameter.

@helper AddHelper(**int** a, **int** b)

{

    <label>Addition of two No = @(a + b)</label>

}

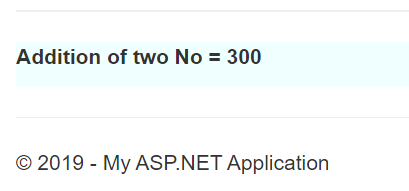
This is how we use inline HTML Helper method in View.

<div style="background-color:azure;">

    <label> @AddHelper(100, 200)</label>

</div>

**Output 1**



**Example 2**

We can create inline HTML Helper method having string parameter.

@helper PrintHelper(**string** message)

{

    <label>@message</label>

}

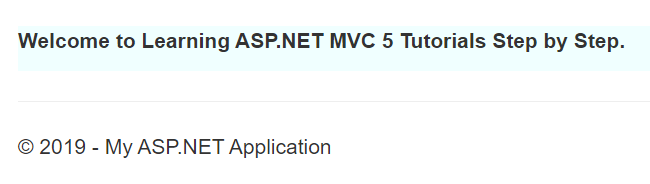
This is how to use inline HTML Helper method in View.

<div style="background-color:azure;">

    <label> @PrintHelper("Welcome to Learning ASP.NET MVC 5 Tutorials Step by Step.")</label>

</div>

**Output 2**



**Example 3**

We can create “ListHelper()” inline HTML Helper method having list as a parameter.

@helper ListHelper(**string**[] strList)

{

    <ol>

        @**foreach** (var item **in** strList)

        {

            <li>@item</li>

        }

    </ol>

}

How to use “ListHelper()” inline Html Helper method in View.

Declare one string array as follow.

@{

**string**[] strBooks = **new** **string**[] { "C#.NET", "ASP.NET MVC", "ASP.NET CORE", "VB.NET", "WEB API" };

}

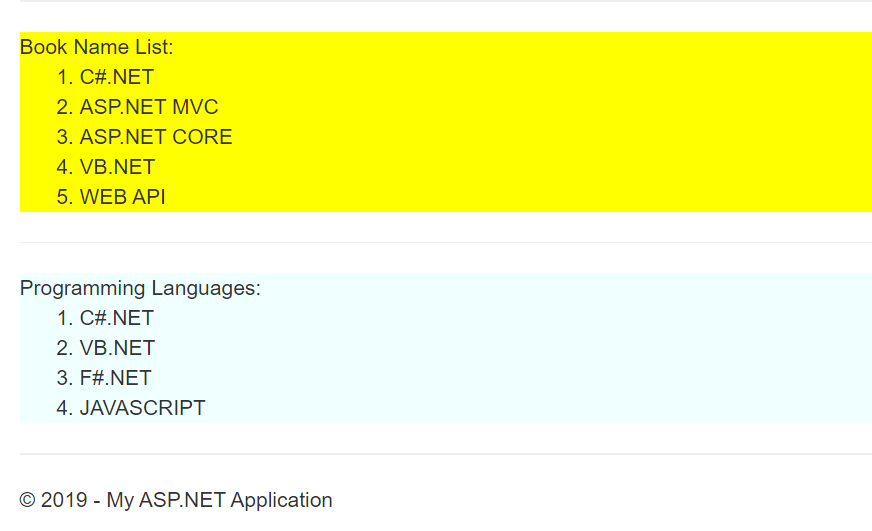
After declaring string array, use the Inline HTML Helper method in View and pass the string array to the “ListHelper()” method.

Print the book list using ListHelper Inline HTML Helper as follow.

<div id="div1" style="background-color:yellow;">

    Book Name List: @ListHelper(strBooks)

</div>



# Different Types of HTML Helpers in ASP.NET MVC

[ASP.NET](https://www.geeksforgeeks.org/introduction-to-asp-net/) provides a wide range of built-in [HTML](https://www.geeksforgeeks.org/html-tutorials/) helpers that can be used as per the user’s choice as there are multiple overrides available for them. There are three types of built-in HTML helpers offered by ASP.NET.

### 1. Standard HTML Helper

The HTML helpers that are mainly used to render HTML elements like text boxes, checkboxes, Radio Buttons, and Dropdown lists, etc. are called Standard HTML helpers.

**List of Standard HTML Helpers** 

@Html.ActionLink() - Used to create link on html page

@Html.TextBox() - Used to create text box

@Html.CheckBox() - Used to create check box

@Html.RadioButton() - Used to create Radio Button

@Html.BeginFrom() - Used to start a form

@Html.EndFrom() - Used to end a form

@Html.DropDownList() - Used to create drop down list

@Html.Hidden() - Used to create hidden fields

@Html.label() - Used for creating HTML label is on the browser

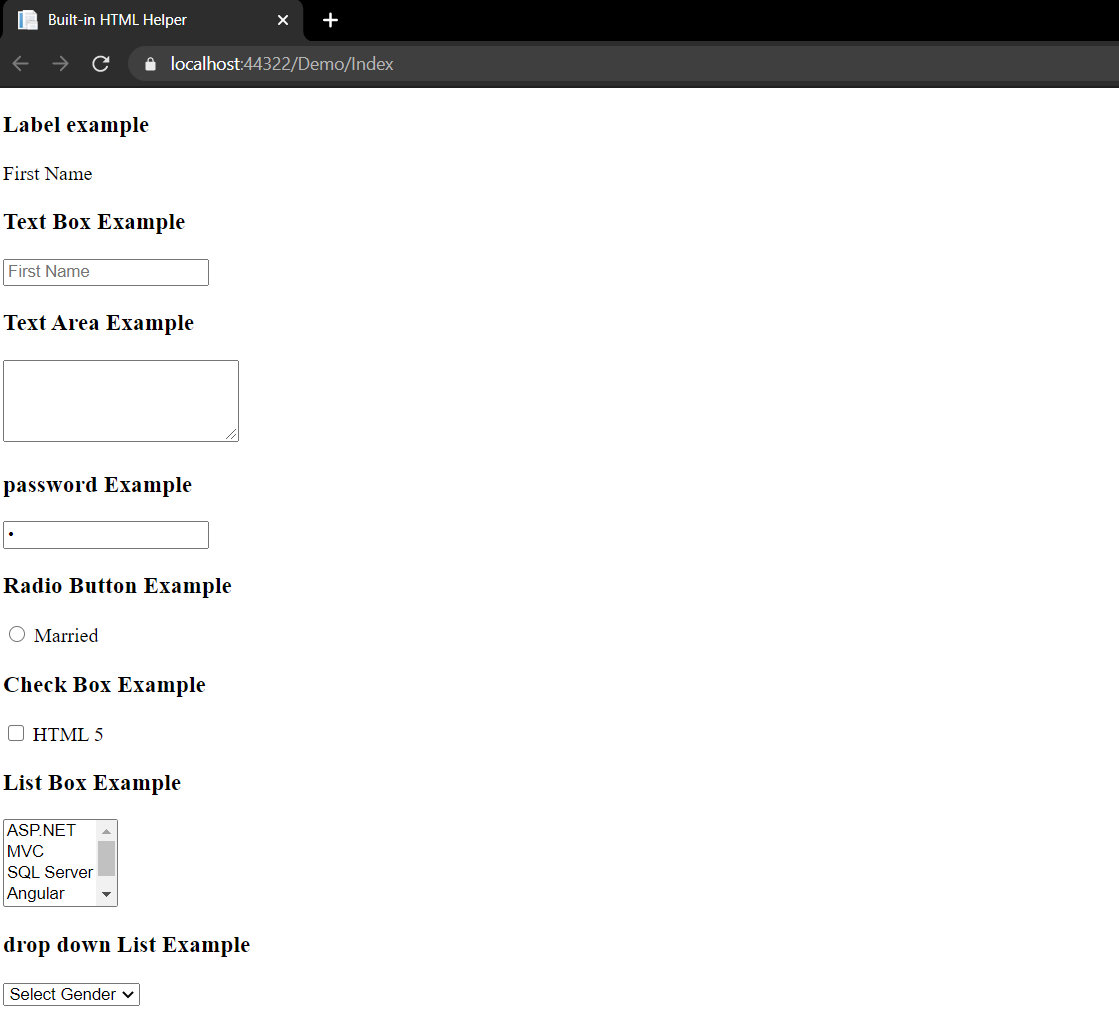
@Html.TextArea() - The TextArea Method renders textarea element on browser

@Html.Password() - This method is responsible for creating password input field on browser

@Html.ListBox() - The ListBox helper method creates html ListBox with scrollbar on browser

* HTML

|  |
| --- |
| @{      Layout = null;  }    <!DOCTYPE html>    <html>  <head>      <meta name="viewport" content="width=device-width" />      <title>Built-in HTML Helper</title>  </head>  <body>      <div>          <h3>Label example</h3>          @Html.Label("firstName", "First Name")            <h3>Text Box Example</h3>          @Html.TextBox("txtFirstName", "", new { @class = "form-control", placeholder = "First Name" })            <h3>Text Area Example</h3>          @Html.TextArea("address", new { @class = "form-control", rows = "5" })            <h3>password Example</h3>          @Html.Password("password", " ", new { @class = "form-control" })            <h3>Radio Button Example</h3>          @Html.RadioButton("MaritalStatus", "Married", new { id = "IsMarried" }) Married            <h3>Check Box Example</h3>          @Html.CheckBox("htmlSkill") HTML 5            <h3>List Box Example</h3>          @Html.ListBox("Skills", new List<SelectListItem> {              new SelectListItem { Text="ASP.NET",Value="1"},              new SelectListItem { Text="MVC",Value="2"},              new SelectListItem { Text="SQL Server",Value="3"},              new SelectListItem { Text="Angular",Value="4"},              new SelectListItem { Text="Web API",Value="5"}          }, new { @class = "form-control" })            <h3>drop down List Example</h3>          @Html.DropDownList("Gender", new List<SelectListItem> {                      new SelectListItem {Text="Select Gender",Value="-1" },                      new SelectListItem {Text="Male",Value="1" },                      new SelectListItem {Text="Female", Value="2" }                      }, new { @class = "custom-select" })        </div>  </body>  </html> |



### 2. Strongly-Typed HTML Helper

The Strongly-Typed HTML helper takes a lambda as a parameter that tells the helper, which element of the model to be utilized in the typed view. The Strongly typed views are used for rendering specific sorts of model objects, rather than using the overall View-Data structure.

**List of strongly-Typed HTML Helper** 

@Html.HiddenFor()

@Html.LabelFor()

@Html.TextBoxFor()

@Html.RadioButtonFor()

@Html.DropDownListFor()

@Html.CheckBoxFor()

@Html.TextAreaFor()

@Html.PasswordFor()

@Html.ListBoxFor()

The functionality of all of these are the same as above but they are used with modal classes. Now, as we know we need a model class to use strongly typed HTML. So firstly we will add a model class as follows

* C#

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Web;    namespace HTML\_Helper\_Demo.Models  {      public class Employee      {          public int EmpId { get; set; }          public string Name { get; set; }          public string Gender { get; set; }          public city city { get; set; }          public skills skills { get; set; }          public string Address { get; set; }          public string Password { get; set; }          public bool AgreeTerm { get; set; }      }  }  public enum city  {      Dehli,      Mumbai,      Kolkata,      Channai,      Bangalore  }  public enum skills  {      HTML5,      CSS3,      Bootstrap,      JavaScript,      JQuery,      Angular,      MVC,      WebAPI  } |

Now write the following code in the controller. And then add a view with the default properties.

* C#

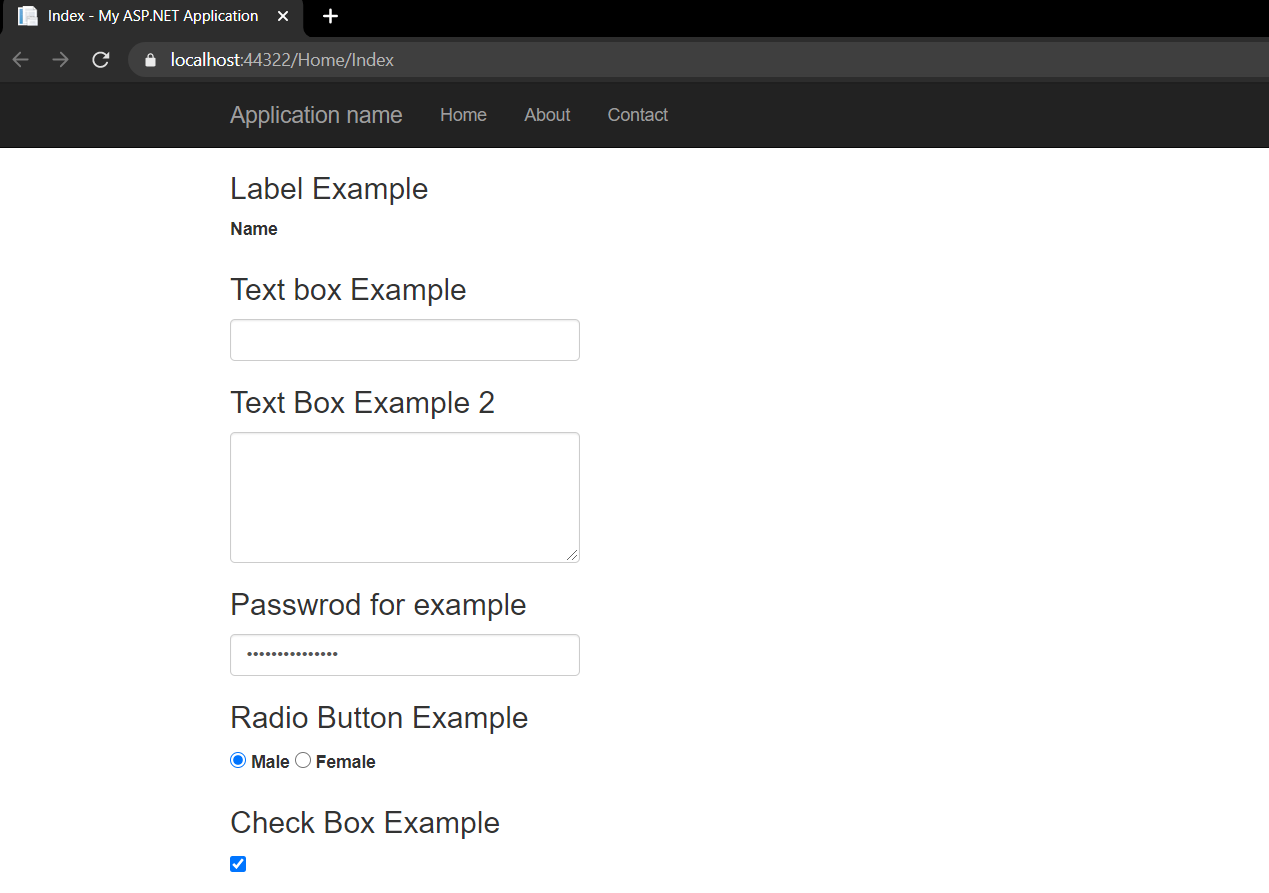
|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Web;  using System.Web.Mvc;  using HTML\_Helper\_Demo.Models;    namespace HTML\_Helper\_Demo.Controllers  {      public class HomeController : Controller      {          public ActionResult Index()          {              return View();          }          [HttpPost]          public ActionResult Index(Employee emp)          {              return View();          }      }  } |

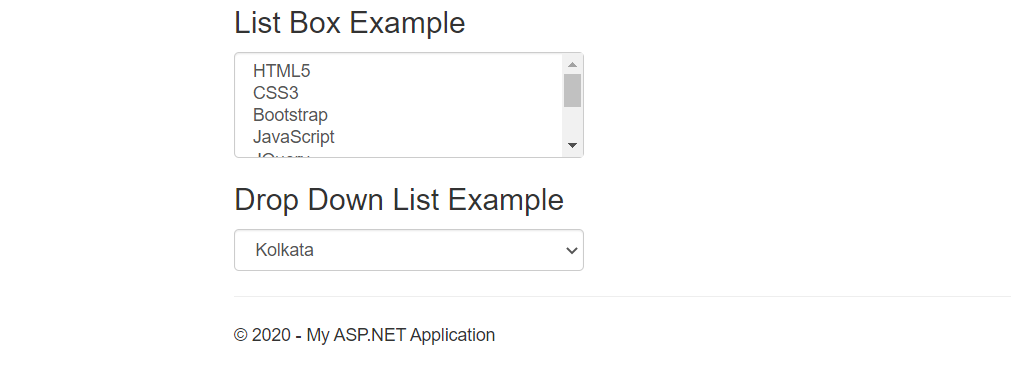
Now write the HTML as follows:

* HTML

|  |
| --- |
| @using HTML\_Helper\_Demo.Models  @model Employee  @{      ViewBag.Title = "Index";  }  <div>      <h3>Label  Example</h3>      @Html.LabelFor(model => model.Name, new { @class = "label-control" })       <h3>Text box Example</h3>      @Html.TextBoxFor(model => model.Name, new { @class = "form-control" })       <h3>Text Box Example 2</h3>      @Html.TextAreaFor(model => model.Address, new { @class = "form-control", rows = "5" })        <h3>Passwrod for example</h3>      @Html.PasswordFor(model => model.Password, new { @class = "form-control" })      <h3>Radio Button Example</h3>      @Html.RadioButtonFor(model => model.Gender, true, new { id = "male-true" })      @Html.Label("male-true", "Male")      @Html.RadioButtonFor(model => model.Gender, false, new { id = "female-true" })      @Html.Label("female-true", "Female")        <h3>Check Box Example</h3>      @Html.CheckBoxFor(model => model.AgreeTerm)        <h3>List Box Example</h3>      @Html.ListBoxFor(model => model.skills, new SelectList(Enum.GetValues(typeof(skills))),                                              new { @class = "form-control" })          <h3>Drop Down List Example</h3>      @Html.DropDownListFor(model => model.city, new SelectList(Enum.GetValues(typeof(city))),                                               "Select City", new { @class = "form-control" })    </div> |

The output will be as follows:





### 3. Templated HTML Helper

The templated HTML Helper is used for data display and input. It generates HTML automatically as per model property and it can generate HTML for a complete model with a single tag.

These are divided into two categories

* Display Template
* Editor Template

**List of Templated HTML Helpers** 

**Display**

@Html.Display()

@Html.DisplayFor()

@Html.DisplayName()

@Html.DisplayNameFor()

@Html.DisplayText()

@Html.DisplayTextFor()

@Html.DisplayModelFor()

**Edit / Input**

@Html.Editor()

@Html.EditorFor()

@Html.EditorForModel()

Now, here we can use the previously created model class and then we should write this code in the controller 

* C#

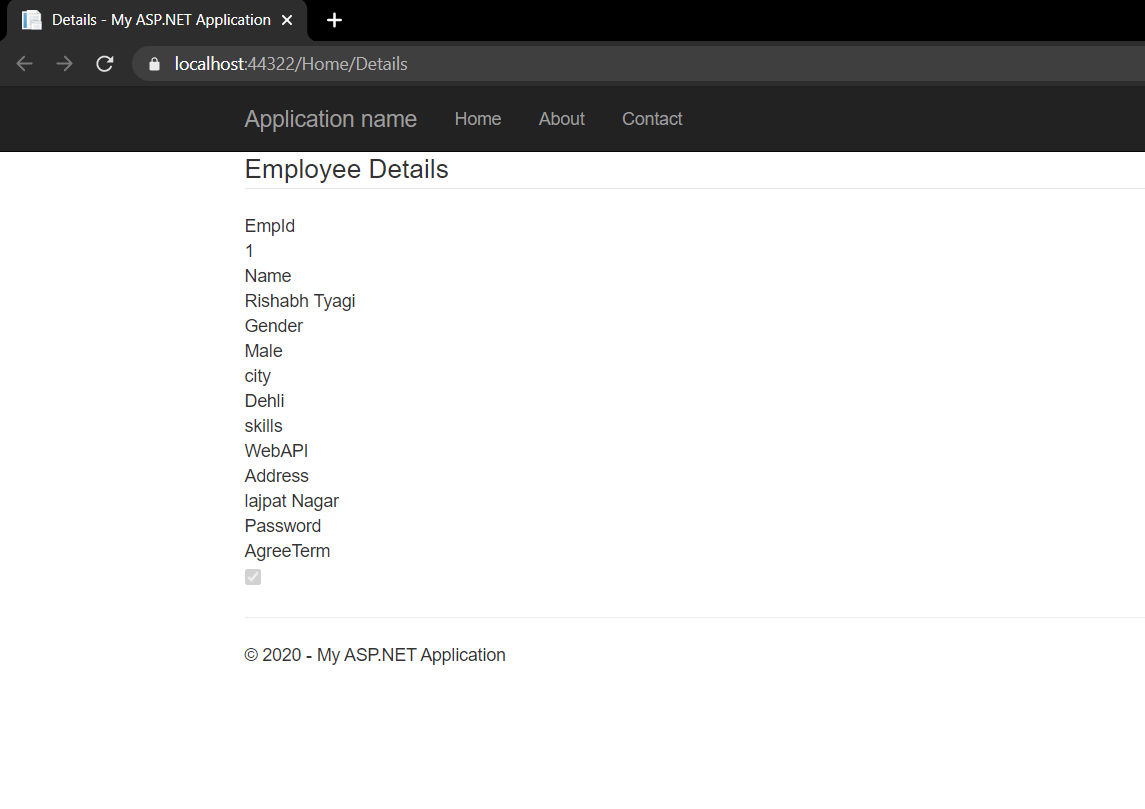
|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Web;  using System.Web.Mvc;  using HTML\_Helper\_Demo.Models;    namespace HTML\_Helper\_Demo.Controllers  {      public class HomeController : Controller      {          public ActionResult Details()          {              //Here we are hardcoded the Employee Details              //In Realtime you will get the data from any data source              Employee employee = new Employee()              {                  EmpId = 1,                  Name = "Rishabh Tyagi",                  Gender = "Male",                  city = city.Dehli,                  skills = skills.WebAPI,                  Address = "lajpat Nagar",                  AgreeTerm = true              };              ViewData["EmployeeData"] = employee;              return View();          }      }  } |

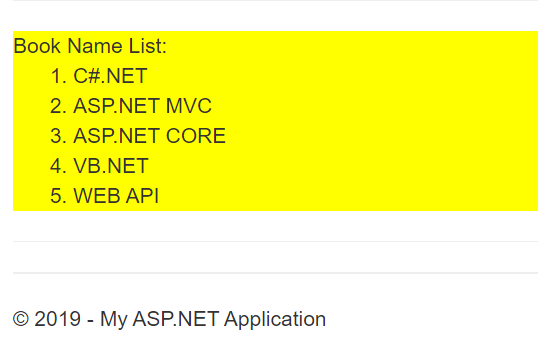
Now add the view with all the default properties and write the following code 

* HTML

|  |
| --- |
| @{      ViewBag.Title = "Details";  }  <fieldset>      <legend>Employee Details</legend>      @Html.Display("EmployeeData")  </fieldset> |

The output will be as follows 





How to reuse the Inline HTML Helper method in the same View

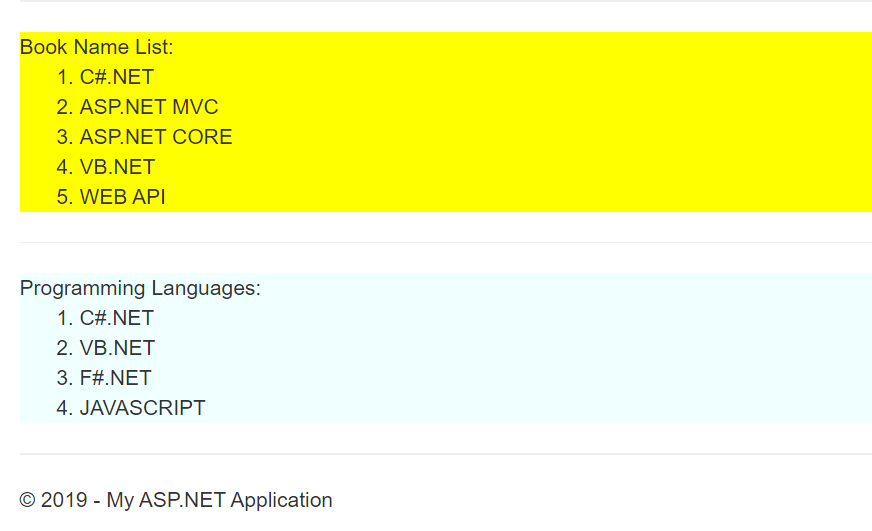
We can declare a string array for list of the programming languages as follows.

1. @{
2. **string**[] strBooks = **new** **string**[] { "C#.NET", "ASP.NET MVC", "ASP.NET CORE", "VB.NET", "WEB API" };
3. **string**[] strLanguages = **new** **string**[] { "C#.NET", "VB.NET", "F#.NET", "JAVASCRIPT" };
4. }

How to reuse the same “ListHelper()” inline HTML Helper method in View.

1. <div id="div1" style="background-color:yellow;">
2. Book Name List: @ListHelper(strBooks)
3. </div>
4. <hr />
5. <div id="div2" style="background-color:azure;">
6. Programming Languages: @ListHelper(strLanguages)
7. </div>

**Output 4**



===========================================================================

**Custom HTML Helpers:**

We can create Custom HTML Helpers in two ways:  
1. Using Static methods.   
2. Using Extension methods.

Using Static methods:  
This is one of the simplest method for creating Custom HTML Helpers. We have added a class (named as CustomHelper ) in CustomHelper folder.

[Graphical user interface, diagram, application

Description automatically generated with medium confidence](https://3.bp.blogspot.com/-X_LOHDpxBxk/V174fQo4uOI/AAAAAAAABLg/Vu3v0b81HSsPpGz6C7Un-n7zS2-30uRCACLcB/s1600/image7.jpg)

|  |
| --- |
| public static class CustomHelper  {          public static MvcHtmlString Image(string source,string altTxt,string width,string height){              //TagBuilder creates a new tag with the tag name specified              var ImageTag = new TagBuilder("img");              //MergeAttribute Adds attribute to the tag              ImageTag.MergeAttribute("src", source);              ImageTag.MergeAttribute("alt", altTxt);              ImageTag.MergeAttribute("width", width);              ImageTag.MergeAttribute("height", height);              //Return an HTML encoded string with SelfClosing TagRenderMode              return MvcHtmlString.Create(ImageTag.ToString(TagRenderMode.SelfClosing));          }  } |
|  |

In the above code, we have created a static method that returns an HTML-encoded string using MvcHtmlString. Now Add namespace reference and call that Custom Helper from the view.

|  |
| --- |
| @{      Layout = null;  }  @using InlineAndCustomHTMLHelper.CustomHelper  <!DOCTYPE html>  <html>  <head>      <meta name="viewport" content="width=device-width" />      <title>Custom Static HTML Helper</title>  </head>  <body>      <div>          @CustomHelper.Image("/Images/UserPic.jpg","UserPic","100","100")      </div>  </body>  </html> |

Preview:

Graphical user interface, text, application

Description automatically generated

Using Extension Methods:  
Extension method enables us to add new methods to an existing class. For creating an Extension method, we have to create a static class and first parameter of an extension method points towards the class that is extended by the method.

|  |
| --- |
| public static MvcHtmlString Image(this HtmlHelper htmlhelper, string source, string altTxt, string width, string height)  {              var ImageTag = new TagBuilder("img");              ImageTag.MergeAttribute("src", source);              ImageTag.MergeAttribute("alt", altTxt);              ImageTag.MergeAttribute("width", width);              ImageTag.MergeAttribute("height", height);              return MvcHtmlString.Create(ImageTag.ToString(TagRenderMode.SelfClosing));  } |

In the above code, we have added Image Custom helper method and extended the HtmlHelper class. Now go to view, add the namespace reference and call Image helper method from HTML Helper class.

[Graphical user interface

Description automatically generated](https://2.bp.blogspot.com/-YpjNEqlr1Zs/V176cbgMSiI/AAAAAAAABL4/31kUBsge96k5hK-h2ruq0Gi46SI1lKTCACLcB/s1600/image9.jpg)

Preview:

[Graphical user interface, text, application

Description automatically generated](https://2.bp.blogspot.com/-Gaq0LhxaI8E/V1763U6m3II/AAAAAAAABMA/yIrNdyqMNQMbmUYvO_yt2wzF2KLprIuGACLcB/s1600/image10.jpg)

Now we can use Image Html Helper in all views. The only thing we have to do is add the reference at the top so that we can call/invoke that custom helper method. If you want to use that custom helper method multiple times, you can add namespace of that custom helper class in the view’s web.config file.

|  |
| --- |
| <add namespace="InlineAndCustomHTMLHelper.CustomHelper"/> |