***PART I : Razor Pages ( 1st semester)***

***Who this tutorial is for***

This tutorial is for students having an intermediate knowledge of C# and are looking to throughout understanding of developing web applications using ASP.NET Core RazorPages framework. This tutorial will take you through a long voyage, full of details, step by step with no fear to get lost on the way. Mr GitHub is all the time at your help. Mr GitHub can provide you with all the code examples used throughout the tutorial. Fasten your belt. You are going through the following chapters:

***Chapter 1: HTML***

**I**n this chapter, you will get a good understanding of HTML. Indeed, we will cover more than what is necessary to work with in this tutorial. We focus on the very important HTML elements such as tables, forms, images, links…etc.

***Chapter 2: CSS***

In this chapter, you will learn the basics of CSS. You will get the necessary to understand what happens behind the scene when working with Bootstrap which is the subject of the next chapter. You will learn how to create and apply a CSS file to a page.

***Chapter 3: Bootstrap***

This chapter covers a big part of Bootstrap. You will learn the most important Bootstrap classes including container and navbar classes. You will also learn bootstrap applied to buttons, forms , inputs, form validation, tables, images …etc.

***Chapter 4: Your First Razor Pages Application***

After getting a correct knowledge of these fundamental technologies for building web applications, you are going to build your first application. You will first be introduced to the ASP.NET Core framework in general, its benefits ...etc. You will build the first Razor Pages application and explore the default file structure. You will also get the opportunity to deeply understand where the application bootstraps and where it ends by investigating the code execution sequence from the time it starts*.*

***Chapter 5: Razor pages architecture***

Before starting coding a web app, it is crucial to understand the architecture of the framework you are working with and how the user request is processed. This is the purpose of this chapter. In this chapter, you will get a good understanding of which class/method is intercepting, handling the request, how the request is routed and how the response is returned to the user.

**Chapter 6 : EventMakerRazorPages application- GetAllEvents**

After getting the hang of how a request is processed in a Razor Pages application, you will start implementing the EventMakerRazorPages application. It is about managing events in Denmark. In this chapter, you will implement the GetAllEvents method to display all the events from a list. At this stage, you are not going to work with a real data storage (i.e. SQL Database ). For the moment, a List data structure can do the job.

**Chapter 7 : Create a new event : CreateEvent**

In this chapter, you will continue implementing the CRUD operations. In this chapter, you implement the CreateEvent method for adding new events to the list. Having some issues with the implementation, you will be introduced to the Singleton design pattern to solve the issues related to implementing the CreateEvent functionality.

**Chapter 8: Data Validation and Singleton design pattern**

In this chapter, you will implement the Singleton design pattern to solve the issues related to implementing the CreateEvent functionality. You will also address the question “***what if the user enters invalid data?***”. To answer this question, you will learn and apply validation.

**Chapter 9: Routing - EditEvent**

In this chapter, you will implement the EditEvent method. To edit a specific event, data about this specific event is passed from a page to another page. This requires knowledge about routing. You will be introduced to routing in RazorPages before implementing this functionality. You will also implement “filtering the displayed events based on the city”. As an exercise, you are supposed to implement “Delete an event” and “display the details of an event”.

**Chapter 10: Dependency injection**

In this chapter, you will get rid of the code used to implement the Singleton Design pattern. Thanks to the way that ASP.NET Core implements Dependency Injection , a singleton service is defined, configured, and then used across the application. You will see how easy it is to implement this pattern*.*

***Chapter 11: Repository Design Pattern - Json file storage***

In this chapter, you will implement another data access layer using a Json file as data storage. To abstract the data access layer, the Repository Design Pattern is used. At the end of this chapter, you will realize how easy it is to incorporate another data access layer with any change to the existing code, which enhances maintainability.

**Chapter 12: Testing**

**I**n this chapter, we will test the application that you have just built; we are mainly going to look at Unit Testing.

**Chapter 13: Building a Real application: 1- \* relationship using the Fake Repository with a list**

**I**n this chapter, we are pushing our implementation further towards real application behaviour. We are going to add another class and implement a **1-many** relationship. This time , we are still using the FakeEvent repository data access layer using a list

**Chapter 14: Building a Real application: 1- \* relationship using a json file with a dictionary**

**I**n this chapter, we are still implementing the **1-many** relationship. However , this time we are using the json file as data storage along with a dictionary as the data structure.

***Chapter 1: Introduction to HTML 5***

**Introduction**

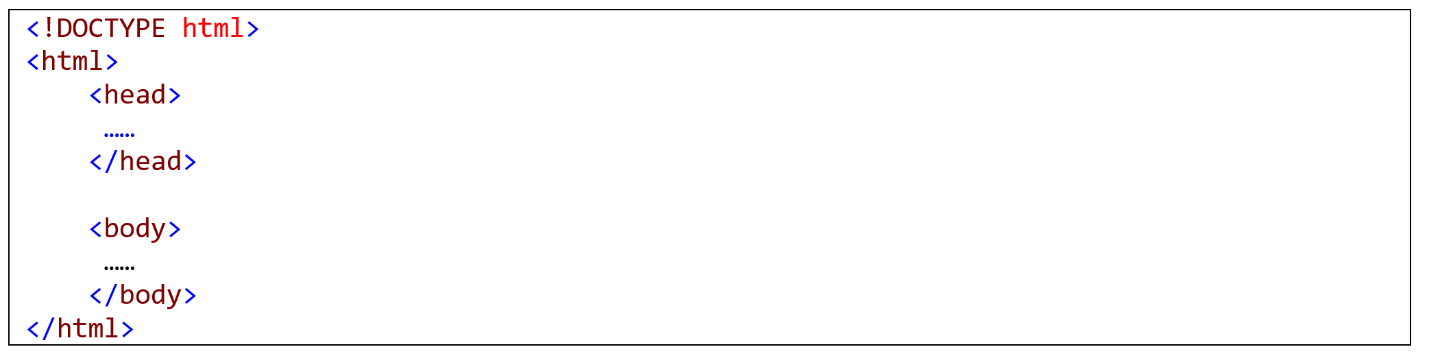
In this chapter, you will be introduced to HTML. HTML stands for  **H**yper **T**ext **M**arkup **L**anguage. The word “Markup” means that the language is a declarative; a human-readable language using tags and attributes to define the different elements of the page and their content. So HTML is not a programming language, rather it is a standard for structuring the page content of the Web. We can modify the page content and its appearance by setting attributes on the different tags. In general, attributes are expressed as name-value pairs.

An example of an HTML page is given below.



**My First page**

Let us look at the page´s HTML code in the figure above. As you can see, the <!DOCTYPE html> <html> , <head> and <body> tags represent the foundation of an HTML page. The way these tags are structured is shown below.



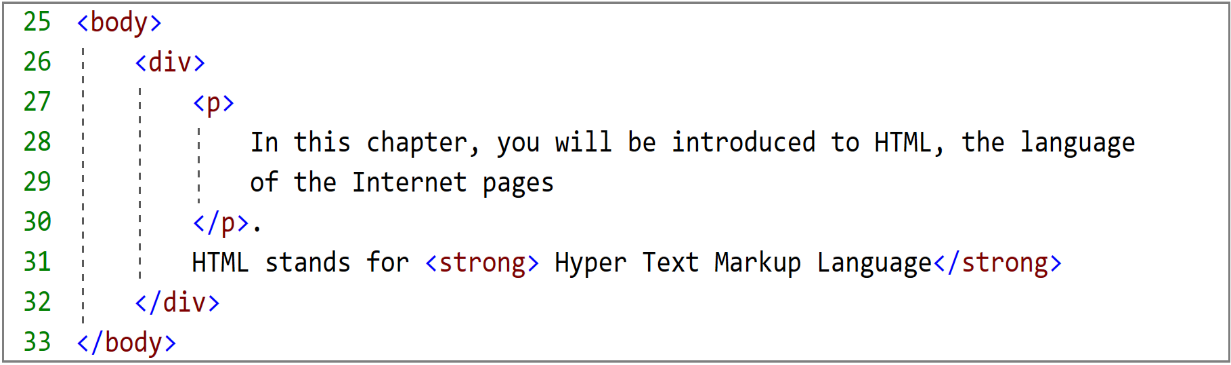
|  |
| --- |
| HTML5 elements are marked up using start tags and end tags. Tags are delimited using angle brackets with the tag name in between. The difference between start tags and end tags is that the latter includes a slash before the tag name. |

Let us explore this structure one tag at time.

* <!DOCTYPE html>. This statement states that the document type is HTML 5. It should be placed at the top of the HTML page.
* <html>. This element represents the root element of your html page. Notice that, as it is true for most html elements, this element has an opening <html> and a closing tag </html>. This <html> element contains 2 important elements: the <head> and the <body> elements:
  + <head>. As we will see later on, this element contains metadata about the document like the title, styles, scripts and other meta information. Meta tags should also go in the <head> tag. The code below is an example of a <head> element content.



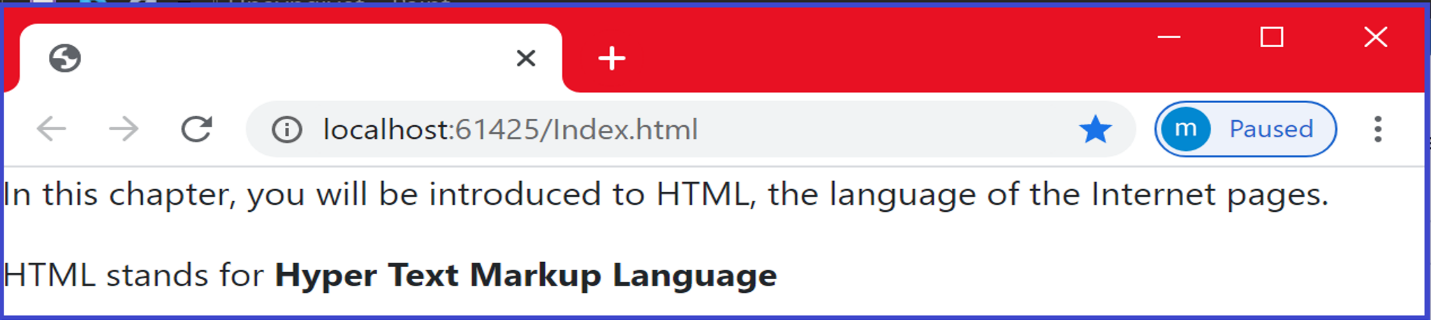
* + <body> This element defines the document body, that is, the visible part of the page. The code snippet below shows an example of a very simple body content.



At the moment, do not worry if you do not understand what is a <div> or a <strong> element. We will cover most of the HTML elements in details in the next sections. Now, you need to understand the <div> element as a container and the <strong> element a way to transform the text inside the element as bold.

**Note** that html elements are not case sensitive and some have no content ( empty elements)

Thanks to the browser (the engine that interprets the html code), the content of the page is read and rendered in very informative laid-out pages that the user can understand. The rendering on the browser of the previous html code is shown in the figure below.



As can be seen, browsers do not display the html tags, but uses them to determine how to display the document. HTML5 has a broad browser support. It is compatible with the latest version of many browsers such as Apple safari, Google Chrome, Mozilla Firefox, Internet Explorer …etc.

**HTML elements and attributes**

HTML elements represent the components of an html page. An html elements may contain attributes that help add information and properties about the element. We can define global attributes that can be applied to any element. We can also define attributes that can be applied to specific elements. They are generally expressed as name-value pairs. The following example illustrates how to mark up an image element with an attribute named src using a value of "image1.jpg".

*< img  src="image1.jpg">*

**Basic elements and attributes**

<**p**>: The <p> element defines a paragraph <**/p**>

<**div**> </**div**> : the div element general purpose is to play the role of a container.

It helps set the different parts of a page and achieve good page layout.

<**br**>:The <br> element defines a line break ( a new line)

**Comment** : **<!-- This is a comment -->**

**Note**: Comments are very useful for debugging HTML (search for errors)

<**h1**>This element defines a heading, it is generally a title or subtitle</h1>

**Note** : We can apply a range of headings from  **h1**  to **h6** tags in descending order of importance, where **h1** defines the most important heading.

<**p**  **style** ="color: red"> <**/p**>

The style attribute is used to style the element . In the example above, we make this paragraph red.<**/p**>

**Note**: You can also use the style attribute to define background color, font family, font size, text alignment…etc. In chapter 2, we will see how to define sophisticated style that can be applied to the whole page or specific element.

The example below is an example of applying the font family with the value “Calibri” to a paragraph.

**<**p **style="**font-family:Calibri;">**This is a paragraph.</p>**

**HTML Links**

<a> element: The <a> HTML element , called anchor element, defines an hyperlink. It has many attributes:

href, target attributes

Consider the following hyperlink element

<a href="https://www.Zealand.com/corona/" target=”\_blank”> About Corona</a>

href=https://www.Zealand.com/corona/. This part defines the address (URL) of the linked page.

target=”\_blank” This part indicates where to open the target page. We want to open the “Zealand.com/corona” page in a new page. You can also use the following values for the target attribute: \_self , \_parent and \_top.

The text “About Corona” represents the text of the link. Instead of a text, you can also use an image.

**HTML images element**

<img> element: The <img> HTML element inserts an image into your page . You can specify the location of the image, its dimension (width, height) , alternative text to be displayed if the image cannot be found. Let us look at some of its attributes.

src **,** alt**,** style attributes

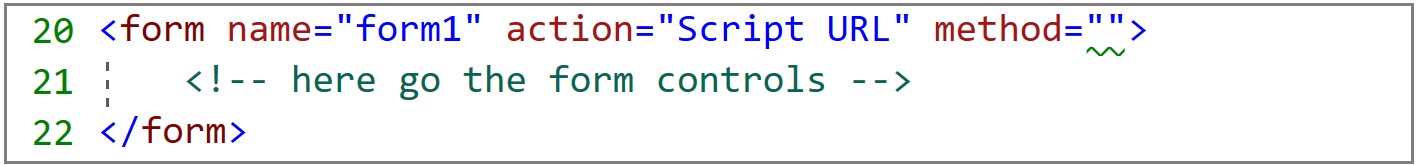
Consider the following HTML image element

<img src="image1.jpg” alt="Tour Eiffel" style="width:42px;height:42px;>

* + <img> element defines an image .
  + src="image1.jpg" The src attribute specifies the path of the image.
  + alt="Tour Eiffel**"** In case the image cannot be displayed, you can specify an alternative text to be displayed using the alt attribute.
  + style="width:42px; height:42px;> The style attribute is used to define the width and the height of the image.

**HTML Form**

HTML forms are mainly used to collect user inputs. A very simple example is a user registration to create accounts. A form is used to collect info such as name, email address, username, password …etc. Once we click on the “Submit” button, the entered data is sent to the back-end application for processing. The form may encapsulate controls such as text fields, text area, dropdown menus , radio buttons, checkboxes, buttons…etc. The code below shows the structure of a <form> element.



**The <**form**>** element defines the template of the form.

The name attribute is used to identify the form.

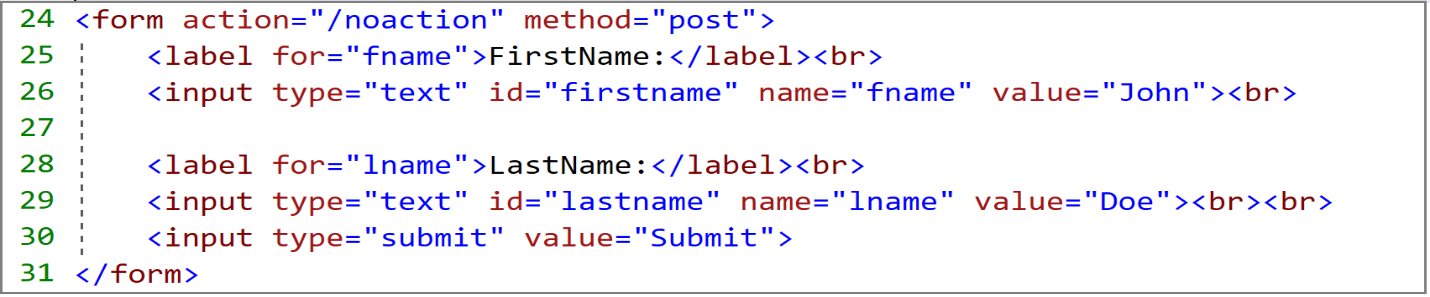
The actionattribute specifies the event handler that will handle the submitted data.

The method attribute defines the type of the http method used to handle the form . For form submission, the POST method is commonly used. We can also use the **Get** method. However, the main difference between these two methods is that, with “get” method, the data is displayed in the page´s address field. So if you want to send sensitive data, it is recommended to use the “**post**” method.

Let us look at the different elements that could be incorporated in a form.

**<label> element**

Let us consider the following html code.



The <label> elements are used to add a caption or a label to each input data you are collecting. Labels help the user enter the right data. The forattribute is used to specify the property for which you want to assign the input data.

**<Input> element**

We continue working with the code shown above.

<input type=”Text”> elements are used to catch the data entered by the user. The data can be text. In this case we use an <input> element whose type attribute has “Text” value.

<input type="submit"> defines a button that is susceptible of firing events. The value attribute defines the text displayed on the button. By clicking on this button, the form data will be sent to back-end.

**TextArea**

Sometimes the user is willing to enter many lines of text. The <textarea> element is used for this purpose. In the following example, we defined a text area which is 14 columns wide and 6 rows high.

|  |  |  |
| --- | --- | --- |
| <body>  <form method="post" style="margin-left:12px; margin-top:12px;">  <textarea cols="14" rows="6"></textarea>  <br />  <input type="submit" value="Submit " />  </form>  </body> | | |
|  |  | *Notice the use of the style attribute to set the left and the top margins. We used the “*cols*” attribute to define the width and the “*rows*” attribute to define the hight of the text area.* |

**Radio buttons**

Radio buttons allow you to set up a list of options, from which the user can pick just one. In this scenario, radio buttons should be used to force you to select only one option. The following example shows the use of radio buttons.

|  |
| --- |
| <form method="post" style="margin-left:12px; margin-top:12px;">  <legend>What is Your Favorite Movie?</legend>  <input type="radio" name="favorite\_movie" value="Star Wars" checked>Star wars<br>  <input type="radio" name="favorite\_movie" value="Fast & Furious">Fast & Furious<br>  <input type="radio" name="favorite\_movie" value="Bad Boys">Bad Boys<br>  <br>  <input type="submit" value="Submit ">  </form> |

|  |  |
| --- | --- |
|  | ***Notice*** *that we define a name for the radio button. We need to identify the control when posting the form. This helps know what has been selected by the user.* ***Notice*** *also that we pre-checked the first option using the “****checked****” attribute. This will force the user to select one of the available options.* |

**Note** that you should use labels to tie your radio button and the descriptive text together, to allow the user to click a larger area when manipulating the radio button. We cover this aspect in the next section.

**Radio buttons and labels**

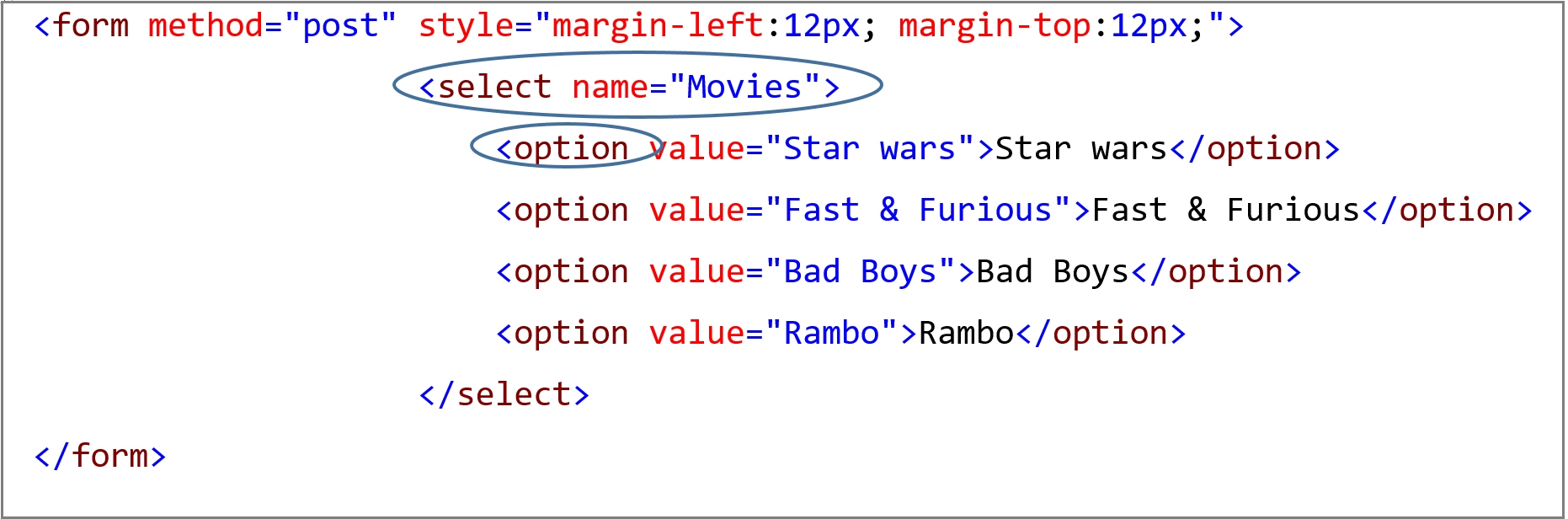
In the previous example, you have to click on the radio button itself. This may difficult if you are using a small device (smartphone). This can be solved using labels to tie the radio button to the attached text, which allows you click on the text as well.

|  |
| --- |
| <form method="post" style="margin-left:12px; margin-top:12px;">  <legend>What is Your Favorite Movie?</legend>  <input type="radio" name="favorite\_movie" value="Star Wars" id="wars" >  <label for="wars">Star wars</label></br>    <input type="radio" name="favorite\_movie" value="Fast & Furious" id="fast" >  <label for="fast">Fast & Furious</label></br>    <input type="radio" name="favorite\_movie" value="Bad Boys" id="boys" >  <label for="boys">Bad Boys</label><br>    <input type="submit" value="Submit ">  </form> |

**Notice** the use of the “**id**” attribute in the input element and the use of the “**for**” attribute in the label element to assign the label to the corresponding input element.

**Dropdown List**

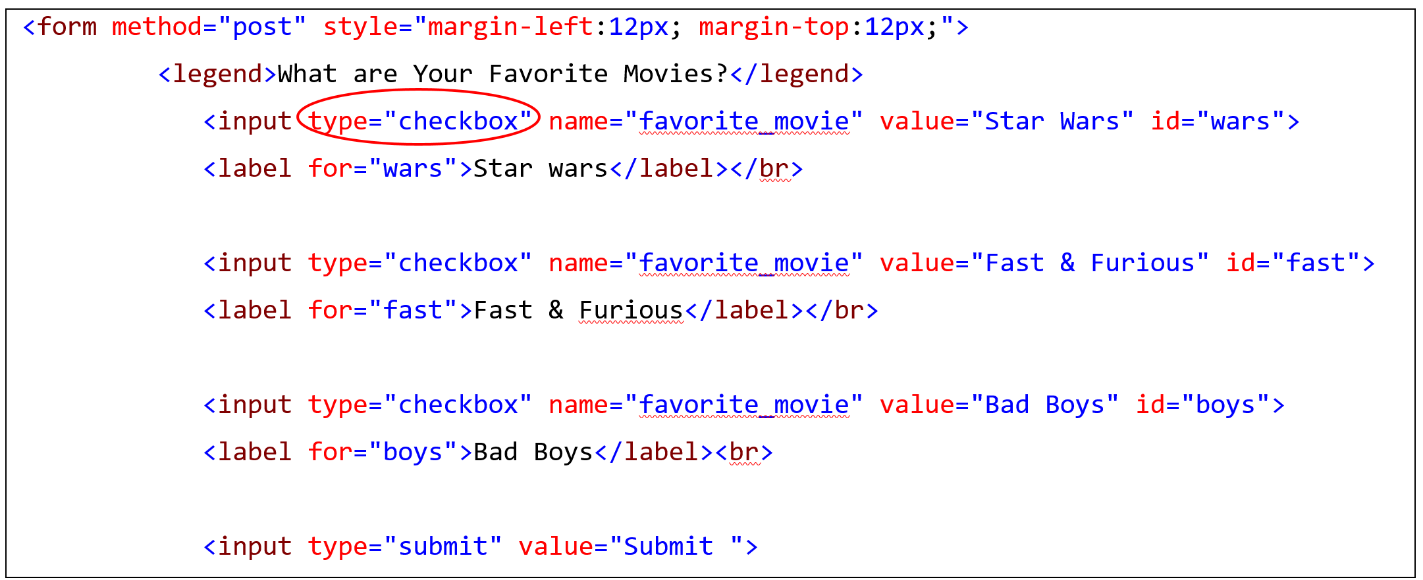
Another way to allow the user pick only one option is the use of dropdown list. There is a small difference between using radio buttons and dropdown list. The dropdown list forces the user to select an option , whereas the radio button does not force the user to pick one. **This is commonly used when ????**

****

|  |  |
| --- | --- |
|  | *Notice the use of the <****select****> element to create the dropdown-list.*  *The <****option****> element defines each option.*  *By default, the first option in the list is selected.* |
|  | <select name="Movies">  <option value="Star wars">Star wars</option>  <option value="Fast & Furious">Fast & Furious</option>  <option value="Bad Boys" selected="selected"> Bad Boys </option>  <option value="Rambo">Rambo</option>  </select>  ***Note*** *that If you want to select a specific option , use the “****selected****” attribute* | |

**Checkboxes**

As opposite to the radio button, which allows you to pick only one option from a list of options, checkboxes allow you choose several options from a group of options. Let us consider the previous example that used radio buttons. Instead of radio buttons, let us use checkboxes .



|  |  |
| --- | --- |
|  | ***Notice*** *that the only thing I did change is the type of the input element from radio to chakcbox.*  ***Also Notice*** *as with all input elements, you need to define a name to be able to identify the checked elements when posting the form.*  ***Notice****, as done with radio buttons before, the use of labels to attach the checkbox to text, which allows a vast area of selection.* |

**Submit & Reset Buttons**

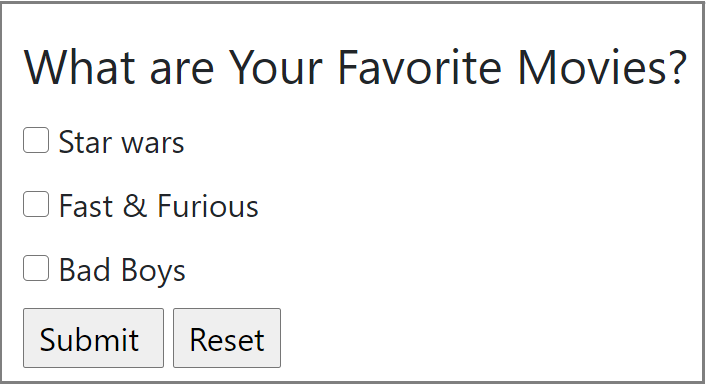
You might notice that in the previous form examples , we have been using the “submit” button to submit the form. The “**submit**” button is also an input element whose type is “submit”.

<input type="submit" value="Submit ">

Another button that we did not talk about is the “**reset**” button. The “Reset” button is also an input element whose type is “reset ” and it is used to clear all inputs.

<input type="reset" value="Reset" />

Adding the Reset button to the previous example is illustrated in the figure below.



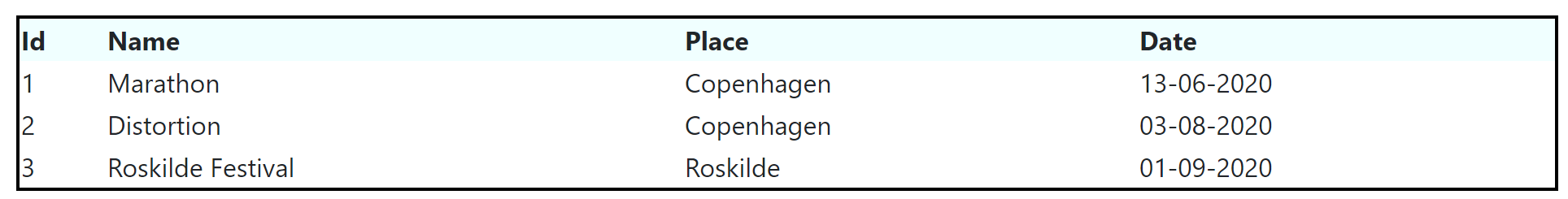
**HTML Tables**

HTML tables are used to display data into rows and columns. You create a table using the <**table**> element. This is the first thing you start with. Then you need to create the table header using the <**th**> element. Table rows are defined using the <**tr**> (stands for table row) tag , while the <**td**> (stands for table data) tag defines a table cell. The following table contains a header, three rows and four columns.

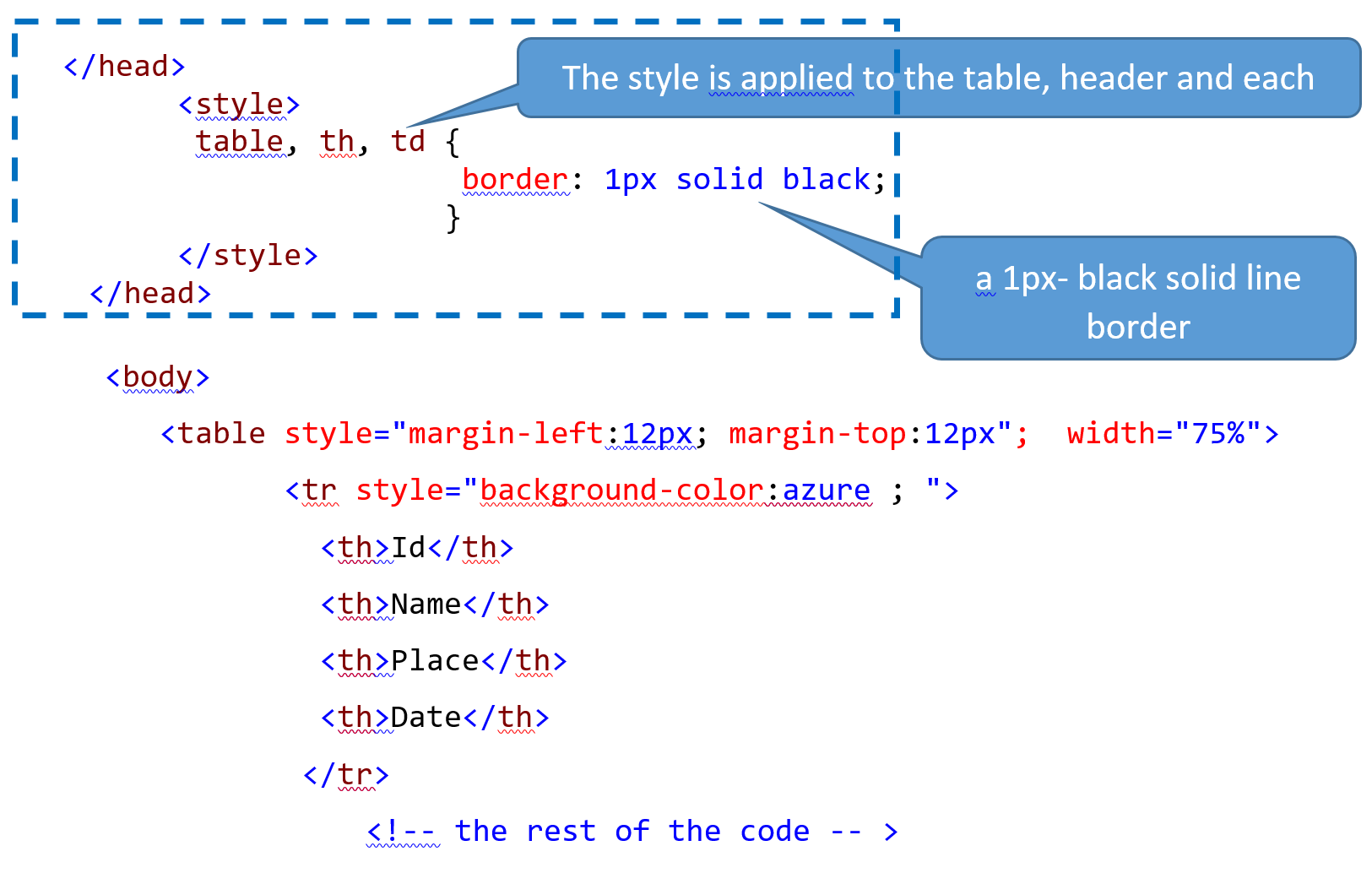


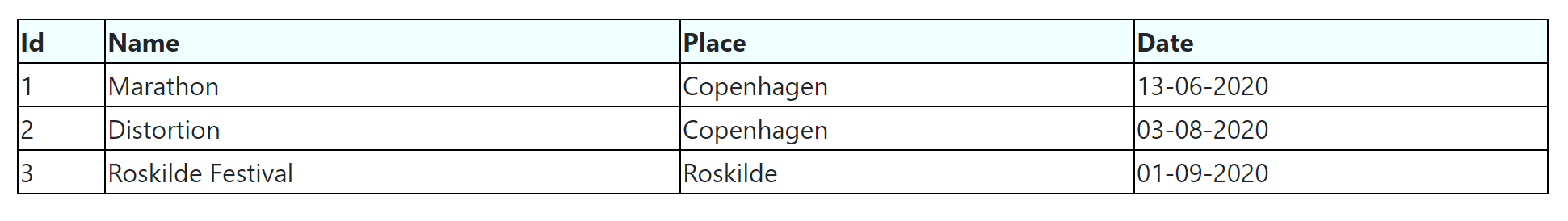
* Notice that we have also added of a border of the table using the border attribute with the value 2px solid black"
* We have also applied a background color to the first row only using the style attribute with the value background-color:azure.

The result is shown below.

****

You probably want to show data in a kind of a Grid with borders between rows and between columns. You can add borders for each row and for each column using the style attribute. This is probably going to be a cumbersome process. A very practical way is to define the style in the header and apply it to any element. So let us define a border that should be applied to the <**table**>, <**tr**> and <**td**> elements. You just need to the following style code in the header section





Notice that we removed the border styling from the <table> element and we applied to the <table>, <th> and<td> elements.

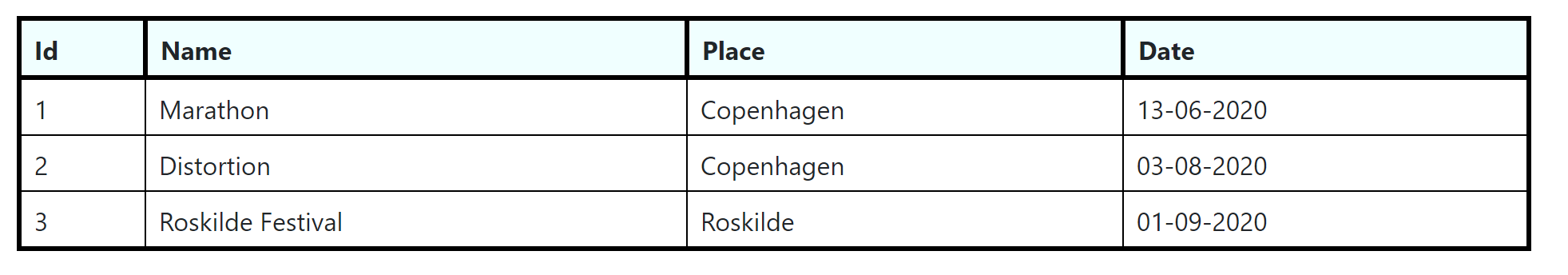
Looking at the table above, you may notice that the text inside each cell is left aligned by default and you may want to add some padding (a padding is a space between the text and the border) or you want to center-align the text. As we did previously, we can easily define a style that does the job , apply it to the **concerned** elements and place it in the header.

The code below applies the padding style 6px 4px 4px 8px to the <th> and <td> elements. This padding corresponds respectively to the top, right, bottom and left padding.

|  |
| --- |
| th , td{  padding: 6px 4px 4px 8px ;  } |

Let us incorporate this style into the previous one. The code below shows the final style. Notice that instead of applying the same border property with value “ 1px solid black” to the <table>, <th> and <td>,we applied a border property with value “ 3px solid black” to table and the header element while a “1px solid black” border is applied to the <td> element.

|  |
| --- |
| <style>  td {  border: 1px solid black;  }    table, th{  border: 3px solid black;  }    th, td {  padding: 6px 4px 4px 8px ;  }  </style> |



Some words before closing this chapter : You might notice that we already used the <style> element to add styling to our page. The question is: what if I want to apply the same style to many pages ? Should I add it to every page ? won´t be tedious ? The next chapter is going to answer these questions. We are going to cover CSS (Cascading Style Sheet). It is going to be relatively short, so let us move to the next chapter.