**FORMS**

Almost every time you want to collect information from a visitor to your site, you need to use a *form.* Some forms are quite complex, such as those that allow you to book plane tickets or purchase insurance online. Others are quite simple, such as the search box on the homepage of Google. Many of the forms you will fill out online bear a strong resemblance to paper forms you have to fill out. On paper, forms are made up of areas to enter text, boxes to check (or tick), options to choose from, and so on. Similarly, on the Web you can create a form by combining what are known as ***form* *controls****,* such as textboxes (to enter text into), checkboxes (to place a cross in), select boxes (or dropdown) and radio buttons (to choose from different options), and so on. In this chapter, you learn how each of these different types of controls can be combined into a form.

In this chapter then, you’ ll learn the following:

* How to create a form using the <form> element
* The different types of form control you can use to make a form — such as text input

boxes, radio buttons, select boxes, and submit buttons

* What happens to the data a user enters?
* How to make your forms accessible
* How to structure the content of your forms

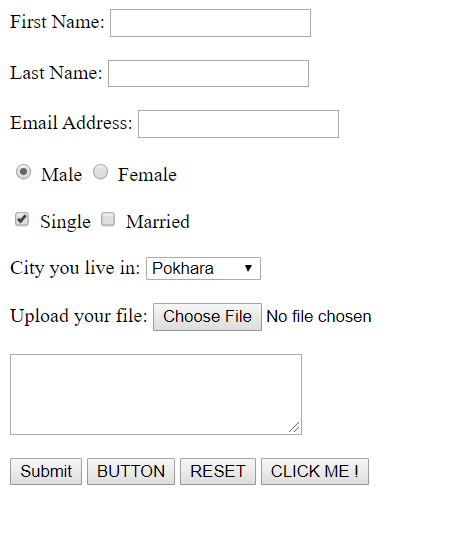
**Introducing Forms:**

Let’ s start by looking at a couple of examples of forms.

Example/Source Code:

****

Output:



**Creating a Form with the <form> Element:**

As you have already seen, forms live inside an element called <form>. The <form> element can also contain other markup, such as paragraphs, headings, and so on, although it may not contain another <form> element.

Providing you keep your <form> elements separate from each other (and no <form> element contains another <form> element), your page may contain as many forms as you like. For example, you might have a login form, a search form, and a form to subscribe to a newsletter, all on the same page. If you do have more than one form on a page, users will be able to send the data from only one form at a time to the server.

Every <form> element should carry at least two attributes:

* action
* method

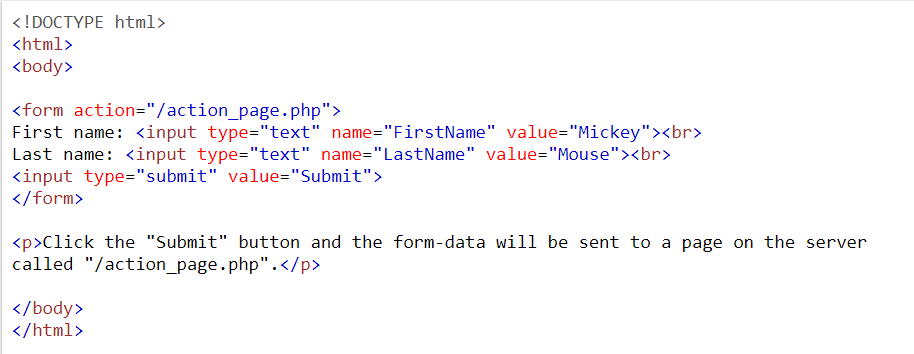
A <form> element may also carry all of the universal attributes, the UI event attributes, and the following attributes:

* enctype
* accept
* accept-charset
* onsubmit
* onreset

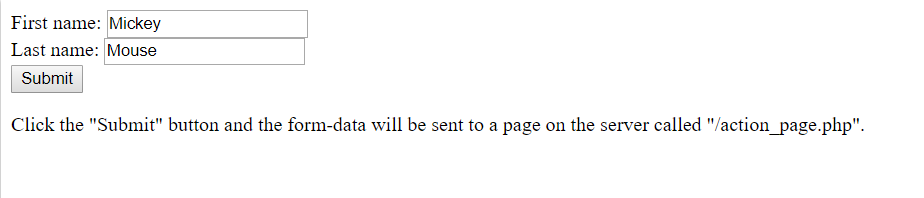
**The action attribute:**

The action attribute indicates what happens to the data when the form is submitted. Usually, the value of the action attribute is a page or program on a web server that will receive the information. The action attribute specifies where to send the form-data when a form is submitted.

Example:



Output:



**The method Attribute:**

Form data can be sent to the server in two ways, each corresponding to an *HTTP method:*

* The get method, which sends data as part of the URL
* The post method, which hides data in something known as the HTTP headers

**The id Attribute:**

The id attribute allows you to uniquely identify the <form> element within a page, just as you can use it to uniquely identify any element on a page.

It is good practice to give every <form> element an id attribute, because many forms make use of style sheets and scripts, which may require the use of the id attribute to identify the form.

The value of the id attribute should be unique within the document. Some people start the value of id and name attributes for forms with the characters frm and then use the rest of the value to describe the kind of data the form collects — for example, frmLogin or frmSearch.

**The name Attribute:**

As you have already seen through its use on other elements, the name attribute is the predecessor to the id attribute, and as with the id attribute, the value should be unique to the document.

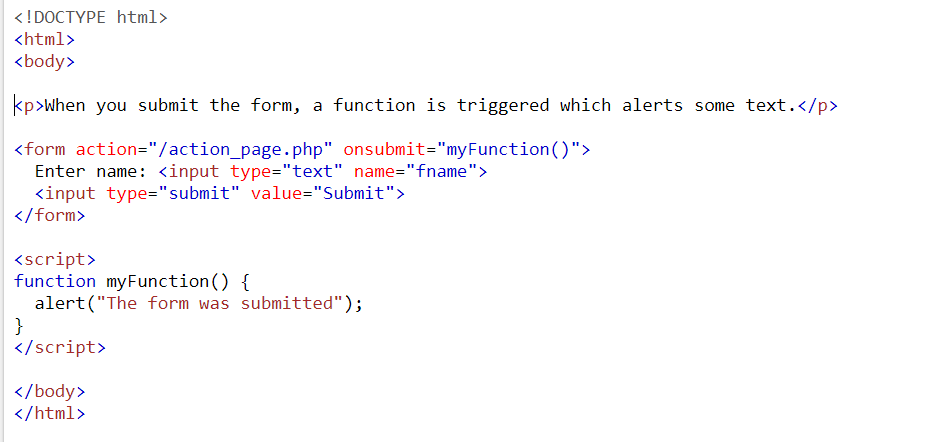
Generally, you will not need to use this attribute, but when you do use it, you can give it the same value as the id attribute. You will often see the value of this attribute begin with the characters frm followed by the purpose of the form (such as frmLogin or frmSearch).

**The onsubmit Attribute:**

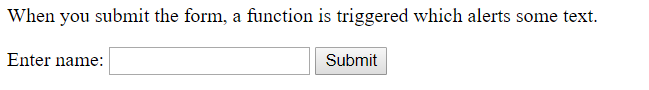
At some point, you have probably filled in a form on a web site, and then, as soon as you have clicked the button to send the form data (even before the page is sent to the server), been shown a message telling you that you have missed entering some data, or entered the wrong data. When this happens, the chances are you have come across a form that uses the onsubmit attribute to run a script in the browser that checks the data you entered before the form is sent to the server.

When a user clicks a submit button, something called an *event* fires. It is rather like the browser raising its hand and saying, “ Hey, I am sending this form data to the server. ” The idea behind these events is that a script (usually written in JavaScript) can be run before the data is sent to the server to check that users have filled in the necessary parts of the form in a format the server expects. The value of the onsubmit attribute should be a script function that would be used when this event fires.

Example/Source Code:



Output:

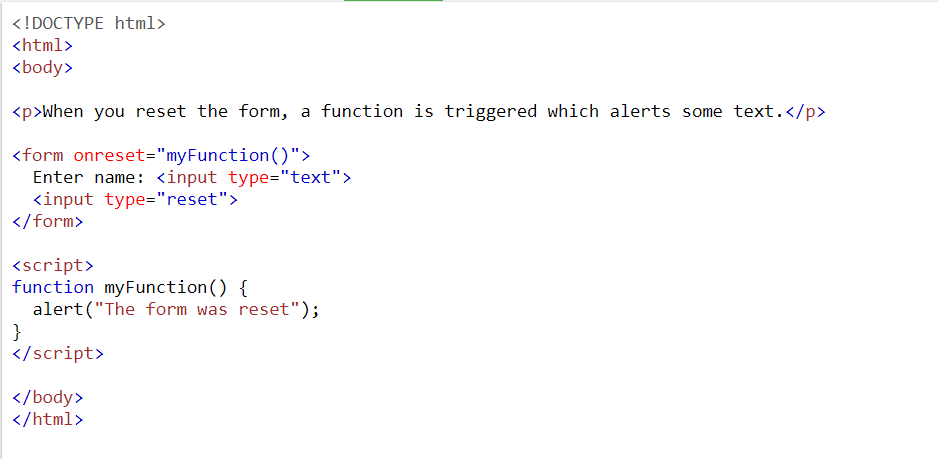


**The onreset Attribute:**

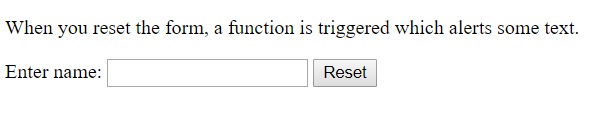
Some forms contain a reset button that empties the form of all details, although the button might say something like clear form instead; when this button is pressed, an onreset event fires and a script can be run.

When the onreset attribute is used, its value is a script (as with the onsubmit attribute) that is executed when the user clicks the button that calls it.

Example/Source Code:



Output:



**The enctype Attribute:**

If you use the HTTP post method to send data to the server, you can use the enctype attribute to specify how the browser encodes the data before it sends it to the server. Browsers tend to support two types of encoding:

* **application/x-www-form-urlencoded**, which is the standard method most forms use.

Browsers use this because some characters, such as spaces are converted to the plus sign(+), and some other non - alphanumeric characters cannot be sent to the web server. Instead, they are replaced by other characters which are used to represent them.

* **multipart/form-data**, which allows the data to be sent in parts, where each consecutive part corresponds to a form control, in the order it appears in the form. It is commonly used when visitors have to upload files (such as photos) to a server. Each part can have an optional content - type header of its own indicating the type of data for that form control. No characters are encoded. This value is required when you are using forms that have a file upload control.

Example:

<form enctype = “application/x-www-form-urlencoded”>

<form enctype = “multipart/form-data”>

**The accept - charset Attribute:**

Different languages are written in different *character sets* or groups of characters. However, when creating web sites, developers do not always build them to understand all different languages. The idea behind the accept - charset attribute is that it specifies a list of character encodings that a user may enter and that the server can then process. Values should be a space - separated or comma - delimited list of character sets.

For example, the following indicates that a server accepts UTF - 8 encodings:

<form accept-charset = ”UTF-8”>

**The accept Attribute:**

The accept attribute is similar to the accept - charset attribute except it takes a comma - separated list of content types (or file types) that the server processing the form can handle. Unfortunately, none of the main browsers supports this feature. The idea is that a user would not be able to upload a file of a different content type other than those listed. Here, you can see that the only types intended to be uploaded are images that are GIFs or JPEGs:

accept = ”image/gif, image/\*”

**Note:** Since the main browsers currently ignore this attribute, if you were to use it visitors would still be able to upload any file.

**The target Attribute:**

The target attribute is usually used with the <a> element to indicate which frame or browser window the link should be loaded into. It can also be used with a form to indicate which frame or window the form results open in when the user has submitted a form.

**White Space and the <form> Element:**

You should also be aware that when a browser comes across a <form> element it often creates extra white space around that element. This can particularly affect your design if you want a form to fit in a small area, such as putting a search form in a menu bar. If CSS will not cure this problem in the browsers you are targeting, the only way to avoid the problem is through careful placement of the <form> element.

To avoid the extra space created, you can try either placing the <form> element near the start or end of the document, or, if you are using tables for layout purposes in a Transitional XHTML 1.0 document, between the <table> and <tr> elements. (You should be aware that this latter approach is a cheat, and therefore it might cause an error if you tried to validate the page. However, most browsers will still display the table and form as you intended.)

**Assignment:**

**Form Controls: (Explain These in details with attributes)**

You’ ve met the <form> element, so this section goes on to cover the different types of form controls that live inside the <form> element to collect data from a visitor to your site. You will see:

* Text input controls
* Buttons
* Checkboxes and radio buttons
* Select boxes (sometimes referred to as drop - down menus and list boxes)
* File select boxes
* Hidden controls