

# Building an HTML Form

In order to receive user-inputted data, you will need a form. HTML forms use the `<form>` element. All form components (or widgets) are placed **within** the open and close tags of the `<form>` element. Forms are within the `<body>` of your HTML page. Upon submission of the form, the inputted data is sent to the server where it will be processed by some other means which is beyond the scope of this lesson. The main focus of this lesson is to learn the HTML elements for a form so that you can build one. Form data retrieval and manipulation will be covered in other courses.

## Tip!

As you work on your HTML, you can use a validator as a debugging tool. If you can't figure out what's wrong with your HTML code, run it through a validator (<https://validator.w3.org/>). You can look through the error messages to help get a sense where the problem is. The validator's error message should contain the line number of your code where the error is possibly located. (The validator will state the line number in your code where it *thinks* the error is occurring, but it is sometimes a little off if there are many major errors.)

## **`<form>`**

The form element encloses all form widgets. There are two main attributes: `action` and `method`.

The `action` attribute value should be the page where the data will be sent. For example, if the form is at **`www.example.com/form`** and you wish to process the data at **`www.example.com/result`**, then the action for the form should be `action="<where_you're_submitting_to>"` where "result" is the relative or absolute path to the processing page. You can omit this for now to submit back to the current page (it's no longer required in HTML5), but it's a good habit to include the action. We will not be doing any processing in this course because you need server-side code or a script for that, but you should be aware of the important attributes.

The `method` attribute specifies whether the data should be submitted via GET or POST. This will be covered later in your PHP class because form data processing is more of a server-side thing, but to cut a long story short, GET will display the data in the URL (think Google search queries) while POST is sent to the server (more secure because the data is not in the URL).

Using the action above, your form can begin with:

```
<form action="<where_you're_submitting_to>" method="post">
... [your form components] ...
</form>
```

## **`<input>`**

Input elements have a variety of widget types:

- *text*: This is just a text field. HTML5 includes a new `placeholder` attribute. You can use this to display help text in the textbox. Beware when using the `placeholder` attribute because it is not supported in every browser.
- *checkbox*: Contains a `checked` attribute to indicate if the box is checked. The `value` attribute holds the value that will be sent for the checked item when the form is submitted. Checkboxes holding the same name will be in the same checkbox list.
- *email*: A text field new to HTML5 which checks for a valid email address format. In older browsers, this should fall back to a text input type. (For IE, this is only supported for versions 10 and up.)
- *tel*: A text field new to HTML5 meant to be used for telephone numbers.
- *file*: Allows a user to upload a file. This will open an OS-native file picker for the user to select a file. An **`accept`** attribute allows you to specify the allowed file types.
- *password*: This is a text field but instead of letters as you type, there will be dots to obscure the value.
- *radio*: This is a radio button. The **`value`** attribute defines the value that the checked radio button sends when the form is submitted. Radio buttons holding the same name will be in the same radio button group. Only one radio button in each group can be selected at a time. If you do not give radio buttons which are meant to be in the same group the same name, the user will be able to select multiple radio buttons. This is undesired behaviour. (For example, if you want the user to select male or female as his or her gender, you do not want the user to be able to select both.) This is important so that when the form is submitted and you grab the form data, you can see which radio button was selected by looking at the one piece of form data (identified by the name attribute). To have one button selected by default, use the **`checked`** attribute. Note that in the example on the next page, the checked attribute was set using `checked="checked"`. As of HTML5, you don't need the value, you can just write `checked` alone without a value.

### Names, IDs, and Classes

Form elements usually have a name (`name` attribute) and an ID (`id` attribute). The name is what you will use when you access the submitted form data. It is how you reference the **data for the submitted element**. An ID is used to identify an element (this can be any HTML element in the `<body>`). IDs must be unique which means that you **cannot have the same ID for more than one element in the same page**. Although the user will not see an error, an HTML validator will throw an error and other unusual behaviour may also occur. IDs are usually used when you want to style an element or manipulate it via Javascript (which is why you have unwanted behaviour if more than one element has the same ID). IDs are useful because you can target a specific element. Note that your name and ID can be the same since the uniqueness of IDs only applies to `id` attribute.

Classes (`class` attribute) are used to style more than one element of a class. Classes are similar to IDs except they are used to **classify** an element, so more than one element with the same class can occur per page. For example, you can have a variety of paragraphs with `class="otherinfo"` which you want to style differently from the main text paragraphs. One element can have multiple class names by separating the names with a space.

For example, `class="displaybox fancy"` means that the element has **two** classes: "displaybox" and "fancy".

- *submit*: This is a button to submit the form. You can also use `<button type="submit">`. Using `<button>` is preferred **now** while `<input type="submit">` was preferred for IE7 and under because `<button>` had some bugs for versions older than IE8. The `value` attribute defines the input submit button's text.
- *hidden*: This is used to carry a value to be sent when the form is submitted but remains hidden from the user. This is usually used when you want to carry forward some other information that the user doesn't need to see or input. The hidden value to be forwarded should be placed in the `value` attribute.

Inputs are **self-closing** tags.

### inputmode

In addition to using the appropriate input type, you can also use the `inputmode` attribute to hint at the type of accepted data instead of using type.

[https://developer.mozilla.org/en-US/docs/Web/HTML/Global\\_attributes/inputmode](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/inputmode)

For example, if you wanted the numeric keyboard to pop up in mobile but not see the up/down arrows, you can do instead:

```
<input type="text" id="mynum" name="mynum" inputmode="numeric">
```

## **Examples**

### The components

Your name:

Your email:

### The corresponding HTML

```
<div>
  <label for="name">Your name:</label>
  <input type="text" id="name" name="name">
</div>
<div>
  <label for="email">Your email:</label>
  <input type="email" id="email" name="email">
</div>
```

☒ Yes ☐ No

```
<div>
  <input type="radio" id="yes" name="yesno"
  value="yes" checked="checked">
  <label for="yes">Yes</label>
  <input type="radio" id="no" name="yesno"
  value="no">
  <label for="no">No</label>
</div>
```

### **<label>**

Labels are the text which appear before a form component. It *labels* the widget. There is also a **for** attribute which must be used. The **for** value should be the ID for the element (e.g. `<input>`, `<select>`, etc.) that the label is referring. This is used to ensure screen readers associate the label with the correct

element and also ensures that the widget is activated when the associated label is clicked. This is essential for accessibility.

Example:

```
<label for="name">Name:</label>
<input type="text" id="name" name="user_name">
```

By default, form components are displayed inline. This means that they are displayed side-by-side and not on separate lines. To ensure the next label and form component is on the next line, you can wrap your label and form component pairs in `<div>` elements. This will result in the below (using the example immediately above):

```
<div>
  <label for="name">Name:</label>
  <input type="text" id="name" name="users_name">
</div>
```

### Divs and spans

A `<div>` element defines a division. Before HTML5, divs were used extensively for layouts. The issue with this approach was that a `div` has no semantic meaning. In other words, there was no way to determine what type of content was within the `div`. This is why HTML5 introduced new tags to better describe the content (e.g. `<footer>`, `<header>`, `<main>`, `<article>`, `<section>`, `<aside>`, `<nav>`).

By default, `<div>` elements are block-level elements, which means that they span all the way across their container elements. Essentially, block elements stand alone on one line.

`<div>` elements are generally used where you just want a block of content or you want to group content but the grouping has no semantic meaning. In the above case, we just wanted the label and textbox to be grouped together, so we used a `<div>`. There is no special reason which affects the *meaning* of the content in any way, so `<div>` was most appropriate. Typically, form component elements such as `<input>` and `<label>` are inline elements, meaning that they display side by side.

`<span>` is another generic element with no semantic meaning. It is used for grouping content. The difference is that `<span>` is a generic inline element, so it's meant to group inline content (such as a bit of text within a paragraph).

Notice that the associated ID in the `for` attribute matches the ID of the text field.

Labels are meant to be used as labels for each individual field. For radio buttons, labels are used for each individual button. To label a radio **button group** you should place the button group as a `<fieldset>`. The radio button group label is the `<legend>` for the `fieldset`. (See `<fieldset>` on page 8.)

### **`<textarea>`**

This is a large textbox used to allow users to enter many lines or paragraphs of text. Text areas are usually used for Javascript rich-text editors (think of a Microsoft Word-like experience) or for message boxes in contact forms. This element has an open and close tag. It is not self-closing like `<input>`.

### Example

#### Form component

Your message:



#### Corresponding HTML

```
<div>
  <label for="message">Your message:</label>
</div>
<div>
  <textarea id="message" name="msg"></textarea>
</div>
```

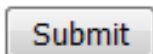
### **<button>**

As mention above, buttons are used to submit data or trigger some action. By default, buttons are taken to be submission buttons to submit a form. If you do not wish for the button to be used as a submit button, you will need to specify `type="button"`. This is especially important in IE, which takes the button to be a submit button even if `type="submit"` is not specified. `<button>` as a submit button is preferable to input of type submit because it semantically makes more sense and it is easier to put content (e.g. an image) within the button because it uses an open and close tag unlike `<input>`. (In old IE versions, using `<input type="submit" ... >` was preferred because IE was buggy when using `<button>`, but using `<button>` for the submit button is now the preferred practice.)

There is a `type="reset"` used for clearing forms.

### Example

#### Form component



#### Corresponding HTML

```
<div>
  <button type="submit"
    name="submit">Submit</button>
</div>
```

### **Practice 1: Build a basic form**

In this example, you will create a basic contact form with a name field, email field, a message box, and a submit button. Since you are only learning how to build the form, it won't do anything when you click the button. You will learn how to manipulate and process the data in other courses. Give your HTML page the title *Contact*.

1. Create a new document in the plain text editor of your choice. Name the file ***form.html***. Remember to save it with UTF-8 encoding and to specify UTF-8 as the character set in the `<head>` section of your page.
2. Type in the basic structure of an HTML page (e.g. DOCTYPE declaration, `<html>`, `<head>`, `<body>`).

3. Type the following within the <head> element:

```
<meta charset="utf-8">
<title>Contact</title>
<meta name="description" content="Contact form">
```

4. Type the following heading in the <body> element:

```
<h1>Contact</h1>
```

5. Add the form open and close tags.

```
<form action="form.html" method="post">
</form>
```

6. Type the following *within* the <form> open and close tags to add the name and email fields.

```
<div>
  <label for="name">Your name:</label>
  <input type="text" id="name" name="user_name" placeholder="Type your name">
</div>
<div>
  <label for="email">Your email:</label>
  <input type="email" id="email" name="user_email" placeholder="john@example.com">
</div>
```

### About Placeholders

Placeholder text as an attribute is new to HTML5. Previously, you could put your placeholder text as the value of the text input then use Javascript or your server-side language to determine that the text field is not empty and is not equal to your placeholder.

The `placeholder` attribute is not supported in every browser. For those browsers, no placeholder text will be visible. This is why it is important to have a label. **Screen readers read the label but not placeholder text.** Without a label, it would be impossible to know what the text field is for. Placeholders are meant to provide an example of valid data for a field and are not meant to replace labels. They both play different roles.

7. Add the message box after the name and email fields.

```
<div>
  <label for="message">Your message:</label>
</div>
<div>
  <textarea id="message" name="user_msg"></textarea>
</div>
```

8. Finally, add the submit button.

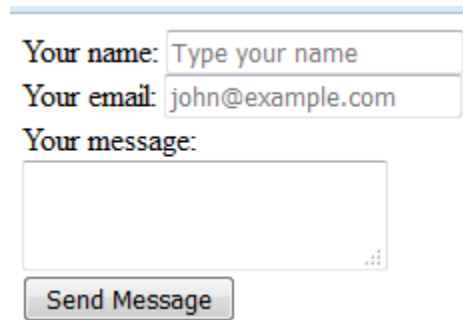
```
<div>
  <button type="submit" name="submit">Send Message</button>
```

</div>

(Alternatively, add the button using `<input type="submit" ... >`.)

9. Save your HTML form page. If you haven't named it yet, save the name as *form.html*.
10. Open the HTML file in a browser.

The form on the page should look like the following:

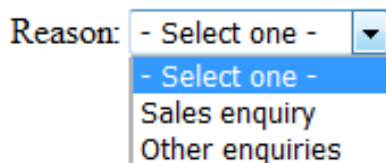


## <select>

You can also allow users to select from a drop-down list. For this, you would need the `<select>` element. Each option in the drop-down list is held by an `<option>` element. These `<option>` elements are placed **between the `<select>` open and close tags**. If you want one option to be selected by default, use the **selected** attribute. You can add the **selected** attribute without a value (in HTML4, you needed to use `selected="selected"`).

### Example

#### Drop-down box



#### Corresponding HTML

```
<div>
  <label for="reason">Reason:</label>
  <select id="reason" name="reason">
    <option value="">- Select one -</option>
    <option value="sales">Sales enquiry</option>
    <option value="other">Other
enquiries</option>
  </select>
</div>
```

## <optgroup>

Within a drop-down box, you can organize or **group** your options, hence `<optgroup>`.

### Example

#### Drop-down box

#### Corresponding HTML

```
<div>
  <label for="reason">Reason:</label>
```

Reason: Sales enquiry ▼

**Category 1**

Sales enquiry

Other enquiries

**Category 2**

Concerns

```
<select id="reason" name="reason">
  <optgroup label="Category 1">
    <option value="sales">Sales
enquiry</option>
    <option value="other">Other
enquiries</option>
  </optgroup>
  <optgroup label="Category 2">
    <option value="concerns">Concerns</option>
  </optgroup>
</select>
</div>
```

## <fieldset>

Use fieldsets to group fields (e.g. address fields or contact info fields). Field components that belong to the <fieldset> go between the open and close tags. To label your fieldset, use a <legend>.

## <legend>

This is the label for a fieldset. This goes immediately after the fieldset's open tag.

## Example

### Radio button group in a fieldset

Gender

☐ Male ☐ Female

### Corresponding HTML

```
<fieldset>
  <legend>Gender</legend>
  <input type="radio" id="male" name="gender"
value="m" >
  <label for="male">Male</label>
  <input type="radio" id="female"
name="gender" value="f" >
  <label for="female">Female</label>
</fieldset>
```

## Practice 2: Add dropdown list to basic form

Add a dropdown box to the form you created in Practice 1. The dropdown box should have the label "Reason for message" and the dropdown box should have the following options:

- Business enquiries
- Media enquiries (make this one the default selected option)
- Other enquiries

Place the drop-down box before the textarea.

1. Type the following directly after the <div> wrapping the email field:

```
<div>
  <label for="reason">Reason for message:</label>
  <select id="reason" name="reason">
    <option value="business">Business enquiries</option>
    <option value="media">Media enquiries</option>
    <option value="other">Other enquiries</option>
```



```
</select>  
</div>
```

2. The drop-down has the second option as the default selected option. Modify the second option to read:

```
<option value="media" selected>Media enquiries</option>
```