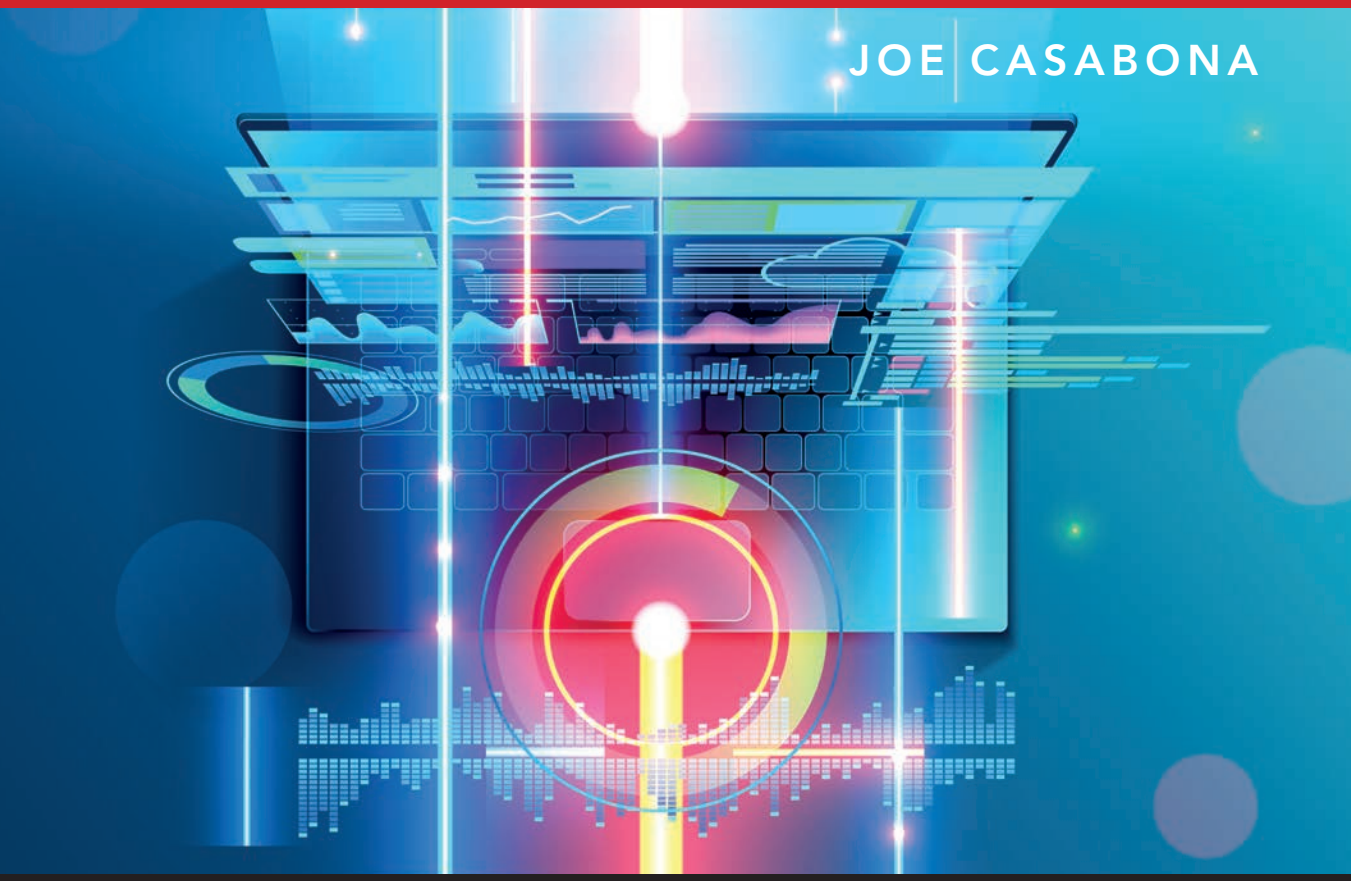
VISUAL QUICKSTART GUIDE

HTML and CSS

Ninth Edition

JOE CASABONA



INCLUDES WEB EDITION WITH FREE VIDEO

VISUAL QUICKSTART GUIDE

HTML and CSS

9th Edition

JOE CASABONA

Peachpit Press

9

Web Forms

So far, everything you've learned about webpages and HTML has taught you how to build a one-way street. That is, you can post information to a webpage, but the website visitors have no way of interacting with you. That's where web forms come in.

Web forms are the primary way users interact with websites. From contact forms to Google's search box, forms drive engagement and make the web a more interactive place.

In This Chapter

Interacting with Webpages

Forms allow users to submit information to your website. You then have the option to store the data or otherwise do something with it. Some examples of popular web forms:

- Contact forms
- Comments
- Forums
- Login boxes
- Post boxes (on social media websites)
- Search boxes
- Checkout pages, Add To Cart buttons, and payment submissions for online stores
- Chatbots
- Popup opt-in boxes

You've no doubt seen lots of forms (**FIGURES 9.1**, **9.2**, and **9.3**).

VIDEO 9.1
Interacting with Forms

There are lots of different forms you can build, as well as many different ways users can interact with those forms. In this video you'll see some unique web forms and learn how they work.

FIGURE 9.1 Google's iconic minimalist home page, with only a search box

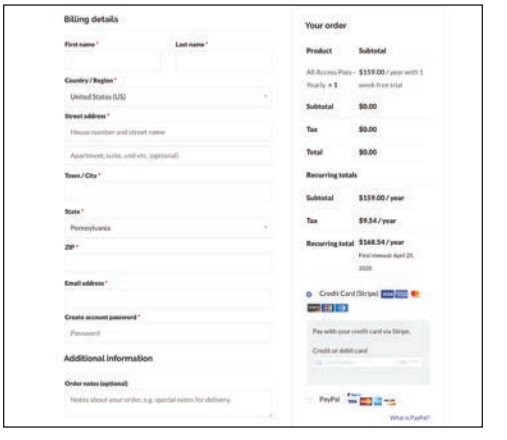


FIGURE 9.2 Checkout form on an ecommerce site

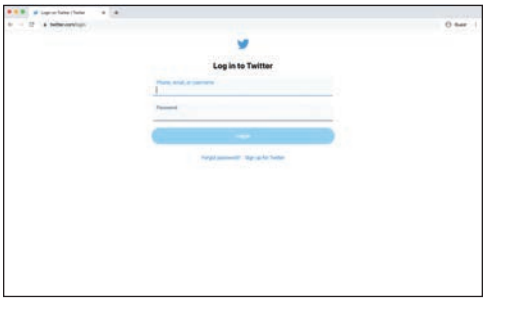


FIGURE 9.3 Twitter's login page

How a Web Form Works

There are several steps to building and processing a web form (**FIGURE 9.4**):

1. Build the form using HTML.
2. Validate the form to make sure all data is submitted properly.
3. Submit and process the form.

   Processing can happen in several ways. You could simply email the contents of the form somewhere, you could store it in a database, you could use it to change your site in real time, and much more.
4. Display confirmation to the user.

Keep this in mind as you read this chapter, though: you're learning how to build the form in HTML, and you're using HTML elements to perform basic validation of the input data.

But with regard to submitting the data, you can't do as much with just HTML. Submission often requires the use of another programming language, which goes beyond the scope of this book. That said, you will learn basic form processing to email the form contents. In Chapter 10, you'll even learn a simple technique for storing the data.

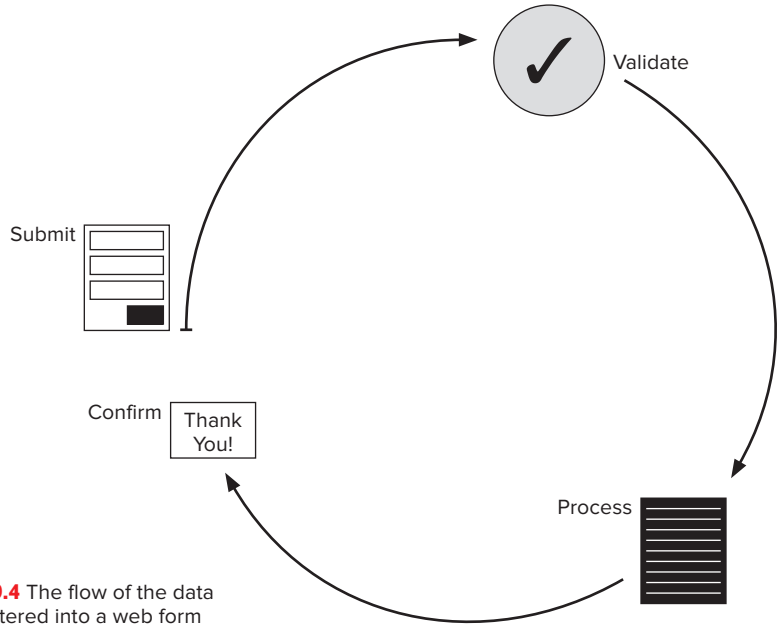


FIGURE 9.4 The flow of the data that is entered into a web form

Components of an HTML Form

Every form is wrapped in a `<form>` element, which consists of the opening `<form>` tag and the closing `</form>` tag.

The bulk of the form itself is made up of *fields* that can accept data from users. Most of these elements are created with `<input>` tags, though there are others you'll also learn about.

TIP Webpages can include more than one form, so placing fields inside the opening and closing `form` tags signals to a browser that all the fields belong to the same form.

The `<form>` Element

Every form needs an opening `<form>` tag and a closing `</form>` tag, which together define a `form` element. The `form` element requires an `action` attribute, and it should have `method` and `name` attributes as well:

```
<form name="search-form"
→ method="GET" action="process.php">
```

The `name` attribute is a simple way to uniquely identify the form (each form should have a unique name). Webpages can contain more than one form, and the `name` attribute allows you to easily reference the form in both CSS and JavaScript.

The `method` attribute determines how the form data should be sent, and can take one of two values.

`GET` is the default value. This method transmits data from one page to another in a URL as *name-value pairs* (**FIGURE 9.5**).

Attributes are a good example of name-value pairs. They have a name (like the `role` attribute) and a value (like "`main`"). In a URL, name-value pairs appear in this format: `role=main`.

When a user fills out your form and clicks Submit, the browser takes all the data from the form and then inserts it into the URL. In the above example, the URL would look something like this: `process.php?name_of_field=value_the_user_input`.

The other `method` value is `POST`. In this method, data is transmitted in the HTTP request and is not shown in the URL. The data is sent, in this instance, to `process.php` via the server.

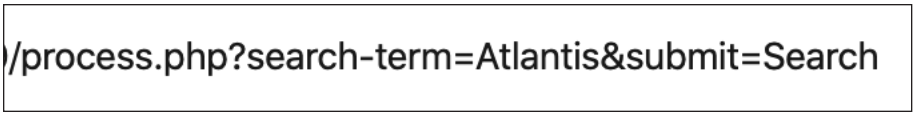


FIGURE 9.5 Using the `GET` method, you can see the results of a form in the URL.

VIDEO 9.2

Comparing the GET and POST Form Submission Methods

To get a better idea of how each action behaves, here you'll see what happens with a GET method compared to a POST method.

The `action` attribute tells the browser *where* to send the form information. It's what does the form processing, whether that's emailing its contents or storing it in a database. If you do not include the `action` attribute, modern browsers will assume the current page will also process the form.

TIP Forms are often processed using a server-side language like PHP, Python, or C#. While that's outside the scope of this book, you can download the `process.php` file from the Github *repo* (short for *repository*, or a place where we can store our code for others to download) for this book (see "Code" in the Introduction).

TIP Privacy and data storage laws are becoming stricter around the world. Depending on where you live, you may need to alert the user to how you're using the data or ask them to explicitly give your website permission to let you store it.

Deciding Between GET and POST

Both GET and POST have pros and cons. With GET, the data entered by the user is visible in the URL, so you should *never* use it to pass sensitive data, like passwords. But using GET (and therefore allowing form data to be visible) is great if you want to make the results shareable or if you want to let the user save them. An example that uses GET is Google search results.

On the other hand, the length of a GET request is limited, ranging from 2000 to 8000 characters, depending on the server and browser configurations.

There is no limit on results when using POST, and it's more secure, but the results pages cannot be shared, nor can the results of a specific form submission be saved.

Form Fields

There are several form fields that you can define in HTML, and each allows the user to interact with the form in a different way.

Input fields

The most common tag you'll find between the opening and closing `<form>` tags is `<input>`. This element creates a field into which users can insert data. It looks something like this:

```
<input type="text" name="search"
→ value="" />
```

Let's take each attribute one by one.

The `type` attribute determines the kind of data a user can input. While `text` is the most common (and the default when the `type` attribute is not defined), there are lots of values the `type` attribute can have. You'll see most of them throughout this chapter.

TIP For a complete (and regularly updated) list, check out developer.mozilla.org/en-US/docs/Web/HTML/Element/input.

The `name` attribute assigns a name to the input field. Remember those name-value pairs from earlier? The *name* is derived from the `name` attribute. It should be unique to prevent overriding data.

The *value* part of the name-value pairs is derived from the `value` attribute. Notice that the `value` attribute in the above example is blank. You can add one, but whatever the user inputs will overwrite it (**FIGURE 9.6**).

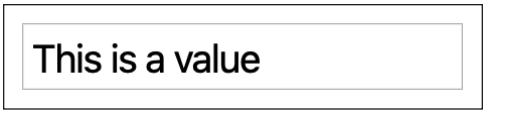


FIGURE 9.6 A text field with the `value` attribute