



# Learning Page Layout Basics

“*Don't make me think!* —Steve Krug  
(Krug's First Law of Usability)



## This chapter covers

- Understanding web page layout types, technologies, and strategies
- Getting to know the HTML5 semantic page layout elements
- Examining modern, real-world page layouts

The first half of this book served to lay down a solid foundation for creating web pages. When you got past the basics of HTML and CSS, you learned about text tags, fonts, colors, CSS classes, the box model, floating and positioning elements, and images and other media. So congratulations are in order: You've graduated from being able merely to *build* web pages to being able to *design* them.



# Learning Page Layout Basics

Alas, you'll have little time to bask in your newfound glory, because this chapter dives right into the next stage of web design. Here, you take a step back from the "trees" of HTML tags and CSS properties to examine the "forest" of page layout. This refers to the overall structure and organization of a web page, and if that sounds trivial or unimportant, consider this: Every single person who visits your page will, consciously or not, be asking a bunch of questions. What is the page about? Am I interested? Does this page have the information I'm looking for? If so, where can I find it?

All those questions are—or, at least, *should* be—answerable by glancing at your layout. If your structure is wonky or your organization is haphazard, I guarantee you that most people will move on after a few seconds. Avoiding that fate means taking a bit of time to plan and code a layout that shows your content in its best, visitor-friendliest light.

## The Holy-Grail Layout

To help you learn the various web page layout techniques, I'm going to use a version of the so-called *holy-grail* layout that consists of the following parts:

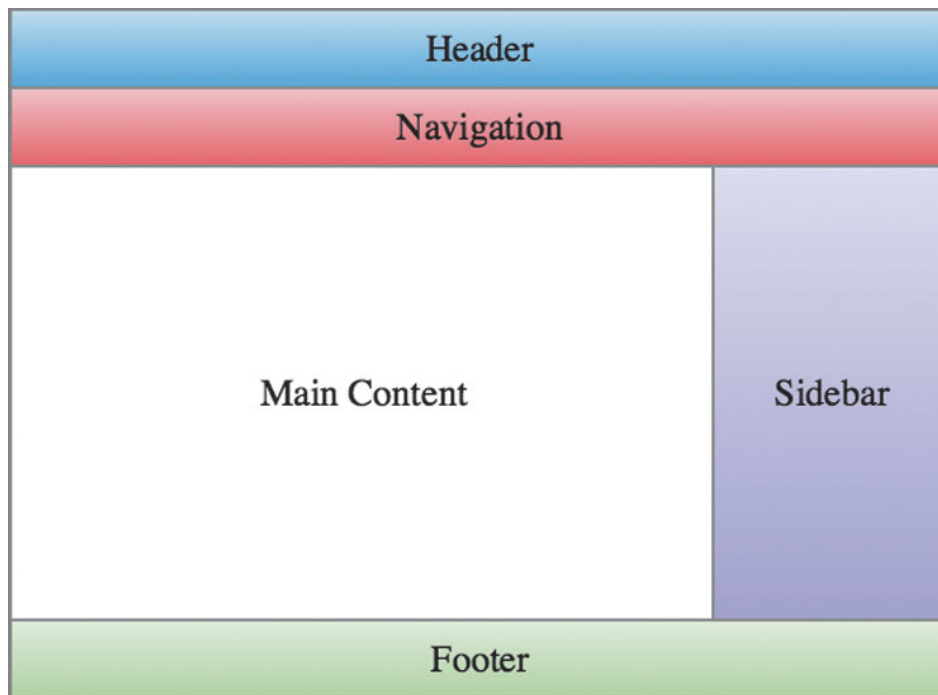
- A header at the top of the page
- A navigation bar below the header
- Two full-height columns consisting of the main page content in the left column and a sidebar of related content in the right (or sometimes the left) column
- A footer at the bottom of the page

### USE IT

*The holy-grail layout is useful for blog posts, articles, essays, how-tos, and similar content-focused pages.*

There are a number of variations on this theme, depending on how strictly you want to define the layout. You may want three columns between the navigation bar and the footer instead of two, for example. Another common variation is to have the footer appear at the bottom of the browser window if the content doesn't extend that far.

Figure 11.1 shows a schematic of the layout you're going to build.



► **Figure 11.1**

A version of the holy-grail web page layout

To build this layout, you need to understand the available page layout methods.

## Understanding Web Page Layout Methods

As I mentioned in Chapter 7, by default the web browser lays out HTML content with the blocks stacked in the order in which they appear in the source document. Within each block, the text runs left to right (for languages that read that way). For the simplest web pages (such as the personal home page you built in Chapter 5), that default layout is fine, but at this point in your web-design career, you're already way beyond that. At this level, you need to know how to break out of that default layout to gain some control of how web content appears on the page.

Fortunately, you have no shortage of ways to do that, but you need to know about three main methods:

- **Floats**—As you learned in Chapter 8, you can use the `float` property to break an element out of the normal page flow and send it to the left or right inside its parent container. By doing this with multiple items, you can organize content into columns and other sophisticated page layouts. See "Creating Page Layouts with Floats" later in this chapter.
- **Inline blocks**—The `display: inline-block` declaration takes a block-level element out of the default vertical page flow and adds it to the horizontal (usually, left-to-right) flow of the other



# Learning Page Layout Basics

## **BEWARE**

Another common page layout technology is a front-end library named *Bootstrap* (<https://getbootstrap.com>). The library comes with prefab HTML, CSS, and JavaScript components that enable you to get your projects off the ground quickly. Most modern web designers, however, eschew complex solutions such as *Bootstrap* in favor of writing their web page code themselves.

inline elements. This creates many interesting page layout opportunities, and you learn about some of them in "Creating Page Layouts with Inline Blocks" later in this chapter.

- *Flexbox*—This powerful but underused CSS module enables you to organize page content in containers that can wrap, grow, and shrink in flexible ways. See Chapter 12 to learn how it works.

Which one should you use? I recommend that you *not* use floats or inline blocks. I do recommend that you learn how floats and inline blocks work for layout—which is why I talk about them in this chapter—because you need to understand the techniques used on so many legacy sites, and you may find these techniques handy for small page components. That leaves flexbox, which you learn about in Chapter 12 and put to good use in Chapter 15's project.

“*Flexbox is certainly something you should take seriously. It paves the way for the modern style of laying out content, and it’s not going away anytime soon. It has emerged as a new standard. So, with outstretched arms, embrace it!* —Ohans Emmanuel

## Learning the HTML5 Semantic Page Elements

The last piece of the page layout puzzle you need to know before getting started is the collection of HTML5 elements that enable you to create semantic layouts. Why is this important? Because every page you upload to the web will be read and parsed in some way by automated processes, such as search-engine crawlers and screen readers for the disabled. If your page is nothing but a collection of anonymous `<div>` and `<span>` tags, that software will be less likely to analyze the page to find the most important content.

To help you solve that problem, HTML5 offers a collection of semantic elements that you can use to specify the type of content contained in each area of your page. There are quite a few of these elements, but the following seven are the most important:

```
<header>
<nav>
<main>
<article>
<section>
<aside>
<footer>
```

The next few sections explain each of these elements.



“Proper semantics . . . increase accessibility, as assistive technologies such as screen readers can better interpret the meaning of our content. —Anna Monus

### **<header>**

You use the header element to define a page area that contains introductory content. This content is most often the site title (which should be marked up with a heading element, such as `h1`), but it can also include things such as a site logo. Here's an example:

```
<body>
  <header>
    
    <h1>Semantics Depot</h1>
  </header>
  etc.
</body>
```

### **<nav>**

You use the nav element to define a page area that contains navigation content, such as links to other sections of the site or a search box. This element can appear anywhere on the page but typically appears right after the page's main header element:

```
<body>
  <header>
    
    <h1>Semantics Depot</h1>
  </header>
  <nav>
    <a href="#">Home</a>
    <a href="#">Blog</a>
    <a href="#">Contact</a>
    <a href="#">About Us</a>
  </nav>
  etc.
</body>
```

### **<main>**

The main element is used as a container for the content that's unique to the current page. Whereas the header, nav, aside, and footer elements are often common to all or most of the pages in the site, the main element is meant to mark up the content that's unique. The main element typically appears after the header and nav elements:

```
<body>
  <header>
    ...
```



# Learning Page Layout Basics

```
</header>
<nav>
  ...
</nav>
<main>
  Unique content goes here
</main>
etc.
</body>
```

## **<article>**

The `article` element is used to mark up a complete, self-contained composition. The model here is the newspaper or magazine article, but this element can also apply to a blog entry, a forum post, or an essay. Most pages have a single `article` element nested within the `main` element:

```
<body>
  <header>
    ...
  </header>
  <nav>
    ...
  </nav>
  <main>
    <article>
      Article content goes here
    </article>
  </main>
  etc.
</body>
```

It's perfectly acceptable, however, to have multiple `article` elements within a single `main` element. Note, too, that it's okay to nest a `header` element inside an `article` element if doing so is semantically appropriate:

```
<article>
  <header>
    <h2>Isn't It Semantic?</h2>
    <p>By Paul McFedries</p>
  </header>
  Article content goes here
</article>
```

## **<section>**

You use the `section` element to surround any part of a page that you'd want to see in an outline of the page. That is, if some part of the page consists of a heading element (`h1` through `h6`) followed by some text, you'd surround the heading and its text with `<section>` tags. This typically happens within an `article` element, like so:

```
<article>
  <section>
    <h3>Introduction</h3>
    Introduction text
  </section>
  <section>
```



```
        <h3>Argument</h3>
        Argument text
    </section>
    <section>
        <h3>Summary</h3>
        Summary text
    </section>
</article>
```

### **<aside>**

You use the `aside` element to mark up a page area that isn't directly related to the page's unique content. A typical example is a sidebar that contains the latest site news, a Twitter feed, and so on. The `aside` element can appear anywhere within the `main` element (and, indeed, can appear multiple times on the page), but it's a best practice to have the `aside` appear after the page's `article` element, as shown here:

```
<body>
  <header>
    ...
  </header>
  <nav>
    ...
  </nav>
  <main>
    <article>
      ...
    </article>
    <aside>
      ...
    </aside>
  </main>
  etc.
</body>
```

### **<footer>**

You use the `footer` element to define a page area that contains closing content, such as a copyright notice, address, and contact information.

Here's the semantic layout of a typical HTML5 page:

```
<body>
  <header>
    ...
  </header>
  <nav>
    ...
  </nav>
  <main>
    <article>
      <section>
        ...
      </section>
      <section>
        ...
      </section>
      <aside>
```



# Learning Page Layout Basics

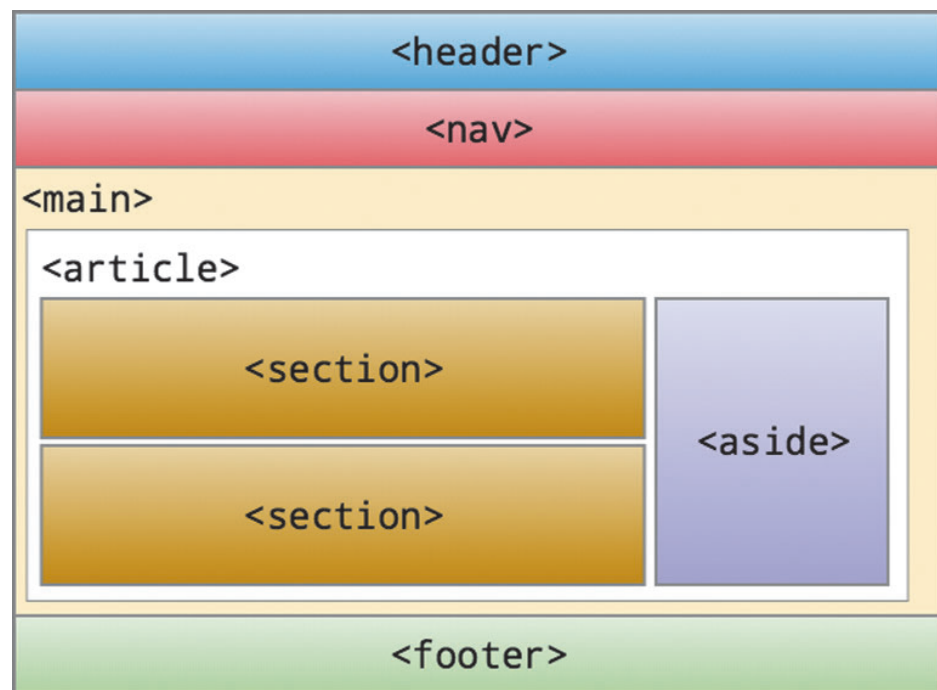
```
        </aside>
      </article>
    </main>
  </footer>
</body>
```

## *The Holy-Grail Layout, Revisited*

Earlier, you learned about the holy-grail layout, which I can reintroduce within the context of the HTML5 semantic page elements. Figure 11.2 shows the same schematic that you saw in Figure 11.1, but with HTML5 semantic layout tags identifying each part.

### ► Figure 11.2

The holy-grail web page layout with HTML5 semantic tags



Here's the bare-bones HTML code for the layout:

```
<header>
  
  <h1>Site Title</h1>
</header>
<nav>
  <ul>
    <li>Item 1</li>
    etc.
  </ul>
</nav>
<main>
  <article>
    <section>
      <h2>Article Title</h2>
```





```

        <p>Article paragraph</p>
        etc.
    </section>
    <aside>
        <p>Sidebar paragraph</p>
        etc.
    </aside>
</article>
</main>
<footer>
    <p>Footer paragraph</p>
    etc.
</footer>

```

## Creating Page Layouts with Floats

I'm simplifying somewhat, but building a page layout with floating elements consists of repeating the following three steps:

- 1 Let the elements flow in the default manner.
- 2 When you need two or more elements to appear beside each other, float them (usually to the left).
- 3 When you want to resume the default page flow, clear the floats.

If you look at the source code for any page that has side-by-side content or content arranged in columns, most of the time that site used floated elements to achieve the effect. That said, floats are losing favor with web designers who yearn for a more straightforward and solid approach to layout. That approach will one day be flexbox combined with a new technology called CSS Grid, but until that day comes, you should be familiar with float-based layouts because they're still used so often.

### LEARN

To get up to speed with CSS Grid basics, see the tutorial “Getting Started with CSS Grid” on the Web Design Playground.  
 ➡ Online: [wdpg.io/grid](http://wdpg.io/grid)

## Lesson 11.1: Creating the Holy Grail Layout with Floats

Covers: Layout with the `float` property

➡ Online: [wdpg.io/11-1-0](http://wdpg.io/11-1-0)

The holy grail includes three instances of side-by-side content:

- In the header, you usually want a site title beside the site logo.
- In the navigation bar, you usually want the navigation items to appear in a row.
- The sidebar must appear to the right of the main content.

All these instances require the use of the `float` property to get the elements out of the default page flow and rendered beside each other.

Begin with the header, as shown in the following example.



# Learning Page Layout Basics

## ► Example

⇒ Online: [wdpg.io/11-1-1](http://wdpg.io/11-1-1)

This example shows you how to use `float` to get the header logo and title side by side.

WEB PAGE

YOUR  
LOGO  
HERE

Site Title

CSS

```
header {  
  border: 1px solid black;  
  padding: .25em 0;  
}  
header img {  
  float: left;  
  padding-left: 1em;  
}  
h1 {  
  float: left;  
  padding-left: .5em;  
}  
.self-clear::after {  
  content: "";  
  display: block;  
  clear: both;  
}
```

Float the `img` element to the left.

Float the `h1` element to the left.

Clearfix to prevent the header from collapsing.

HTML

```
<header class="self-clear">  
    
  <h1>Site Title</h1>  
</header>
```

## REMEMBER

I'm using type selectors (such as `header img`) here to make the code as simple as possible. In practice, it's usually better to assign classes to each element and then select the classes in your CSS.

As you can see, both the `img` element and the `h1` element are assigned `float: left`, which places them beside each other on the left edge of the header element. Because these elements are out of the default page flow, you'll usually have to adjust the padding or margins to get them placed where you want them, as I've done in the example.

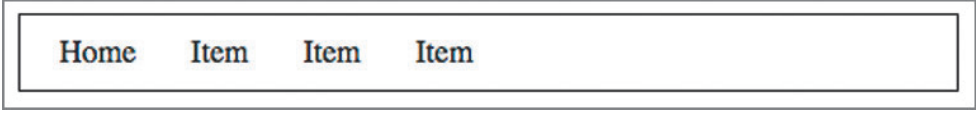
Now float the navigation bar's items, as shown in the following example.



### ► Example

⇒ Online: [wdpg.io/11-1-2](http://wdpg.io/11-1-2)

This example shows you how to use `float` to get the navigation bar items side by side.

WEB PAGE	
CSS	<pre> nav {   border: 1px solid black;   padding: .5em; } nav ul {   list-style-type: none; } nav li {   float: left;   padding-right: 1.5em; } </pre> <p>Float the li elements to the left.</p>
HTML	<pre> &lt;nav class="self-clear"&gt;   &lt;ul&gt;     &lt;li&gt;Home&lt;/li&gt;     &lt;li&gt;Item&lt;/li&gt;     &lt;li&gt;Item&lt;/li&gt;     &lt;li&gt;Item&lt;/li&gt;   &lt;/ul&gt; &lt;/nav&gt; </pre>

In this case, the `li` elements are assigned `float: left`, which places them beside each other on the left edge of the `nav` element. Again, I've used padding to adjust the placement of the elements.

Next, float the main element's `<article>` and `<aside>` tags to create the two-column content layout. The following example shows how it's done.



# Learning Page Layout Basics

## ► Example

⇒ Online: [wdpg.io/11-1-3](http://wdpg.io/11-1-3)

This example shows you how to use `float` to get the article and aside elements side by side in a two-column layout.

WEB PAGE

## Article Title

Article paragraph 1

Article paragraph 2

## Sidebar Title

Sidebar paragraph

CSS

```
article {  
  float: left;  
  width: 75%;  
  border: 1px solid black;  
}  
aside {  
  float: left;  
  width: 25%;  
  border: 1px solid black;  
}
```

Float the article element to the left.

Set the width of the article element.

Float the aside element to the left.

Set the width of the aside element.

HTML

```
<main>  
  <article>  
    <h2>Article Title</h2>  
    <p>Article paragraph 1</p>  
    <p>Article paragraph 2</p>  
  </article>  
  <aside>  
    <h3>Sidebar Title</h3>  
    <p>Sidebar paragraph</p>  
  </aside>  
</main>
```

In this case, both the article element and the aside element are assigned `float: left`, which places them beside each other on the left side of the main element. (You could float the aside element to the right to get the same layout.) You also need to assign a width value to each element to set the size of your columns.

The width of your columns depends on what you'll be using them for. In general, if one of the columns is a sidebar, it shouldn't take up much more than 25 percent of the available width. Note, too, that if you've applied box-sizing: border-box, your column percentages can add up to 100 to fill the width of the main element.



Note, too, that the bottom borders of the two columns don't line up because, in the absence of a CSS height declaration, the browser assigns a height to an element based on the height of its contents. This problem is common with floated columns, but I'll show you a workaround after the next example.

Finally, you're ready to add the footer element, as shown in the example that follows.

## PLAY

How would you modify this layout to display three content columns: a sidebar to the left and to the right of the article element? ➡ Online: [wdpg.io/11-1-7](http://wdpg.io/11-1-7)

### ► Example

➡ Online: [wdpg.io/11-1-4](http://wdpg.io/11-1-4)

This example shows adding the footer to the bottom of the page by clearing the floated columns. Colors have been added to all the elements, but most aren't shown in the code.

WEB PAGE	
CSS	<pre>         footer {             padding: .25em 1em;             background-color: #b6d7a8;         }     </pre>
HTML	<pre>         &lt;footer class="self-clear"&gt;             &lt;p&gt;Footer paragraph&lt;/p&gt;         &lt;/footer&gt;     </pre>

Note, too, that in this example the article and aside columns are the same height. How did I do that? I faked it by using a technique called *faux columns*. Here's how it works:

- 1 Put a wrapper element around both the article and the aside elements.

In this example, the `main` element can serve as the wrapper.

## PLAY

How would you modify this layout to display the sidebar on the left instead of the right? ➡ Online: [wdpg.io/11-1-6](http://wdpg.io/11-1-6)



- 2 Assign the same background color to the wrapper and the `aside` element.

In the example, I assigned the color `#b6d7a8` to both.

- 3 Assign a different background color to the `article` element.

In the example, I assigned `white` to the article background.

Because the wrapper and the `aside` use the same background color, the sidebar appears to reach all the way down to the footer. Here's a skeleton version of the code:

```
<main class="self-clear">
  <article>
  </article>
  <aside>
  </aside>
</main>
<footer class="self-clear">
</footer>
<style>
  main, aside {
    background-color: #b4a7d6;
  }
  article {
    background-color: white;
  }
  .self-clear {
    content: "";
    display: block;
    clear: both;
  }
</style>
```

Using floats to lay out web page content is an old, common CSS trick. As you saw in this section, however, it has some problems. You must remember to clear your floats when needed, for example; margins tend to collapse; and you often have to resort to kludgy tricks such as faux columns to make things look good. You can solve some of these problems by using inline blocks, which I turn to in the next section.

## Creating Page Layouts with Inline Blocks

Building a page layout with inline block elements is similar to using floats:

- 1 Let the page elements flow in the default manner.
- 2 When you need two or more elements to appear beside each other, display them as inline blocks.

Notice that one of the main advantages of using inline blocks is that you don't have to explicitly clear elements.



## Lesson 11.2: Creating the Holy-Grail Layout with Inline Blocks

Covers: Layout with the inline-block property

⇒ Online: [wdpg.io/11-2-0](http://wdpg.io/11-2-0)

As before, the holy grail includes three instances in which you need content side by side: the header, the navigation bar, and the content columns. All these instances require the use of the `display: inline-block` declaration to get the elements out of the default page flow and rendered beside each other. By default, inline blocks are displayed left to right (or according to the default inline orientation), so they're similar to declaring `float: left`.

I begin at the beginning with the header, as shown in the following example.

### REMEMBER

*I'm using type selectors (such as `header img`) here to make the code as simple as possible. In practice, it's usually better to assign classes to each element and then select the classes in your CSS.*

### ► Example

⇒ Online: [wdpg.io/11-2-1](http://wdpg.io/11-2-1)

*This example shows you how to use inline blocks to get the header logo and title side by side.*

WEB PAGE	
CSS	<pre> header {   border: 1px solid black;   padding: .5em 0 .1em 1em; } h1 {   display: inline-block;   padding-left: .5em;   font-size: 2.5em; } </pre> <p>Display the h1 element as an inline block.</p>
HTML	<pre> &lt;header&gt;   &lt;img src="/images/your-logo-here.tif" alt="Our logo"&gt;   &lt;h1&gt;Site Title&lt;/h1&gt; &lt;/header&gt; </pre>



# Learning Page Layout Basics

The `img` element is an inline block by default, and I've declared the `h1` element with `display: inline-block`, which places these two elements beside each other from left to right.

Now convert the navigation bar's items to inline blocks, as shown in the following example.

## ► Example

⇒ Online: [wdpg.io/11-2-2](http://wdpg.io/11-2-2)

*This example shows you how to use inline blocks to get the navigation-bar items side by side.*

WEB PAGE

Home Item Item Item

CSS

```
nav {
  padding: .5em;
  border: 1px solid black;
}
nav ul {
  list-style-type: none;
  padding-left: .5em;
}
nav li {
  display: inline-block;
  padding-right: 1.5em;
}
```

Display the `li` elements as inline blocks.

HTML

```
<nav>
  <ul>
    <li>Home</li>
    <li>Item</li>
    <li>Item</li>
    <li>Item</li>
  </ul>
</nav>
```

In this case, the `li` elements are declared `display: inline-block`, which places them beside each other on the left edge of the `nav` element.

Next, convert the main element's `<article>` and `<aside>` tags to inline blocks, which gives you the two-column content layout. The following example shows how it's done.





## ► Example

⇒ Online: [wdpg.io/11-2-3](http://wdpg.io/11-2-3)

This example shows you how to use inline blocks to get the article and aside elements side by side in a two-column layout.

WEB PAGE	<div> <div> <h2>Article Title</h2> <p>Article paragraph 1</p> <p>Article paragraph 2</p> </div> <div> <h3>Sidebar Title</h3> <p>Sidebar paragraph</p> </div> </div>
CSS	<pre> article {   display: inline-block;   width: 75%;   border: 1px solid black; } aside {   display: inline-block;   vertical-align: top;   width: 25%;   border: 1px solid black; } </pre> <p>Display the article element as an inline block.</p> <p>Display the aside element as an inline block.</p> <p>Align the aside element text with the top.</p>
HTML	<pre> &lt;main&gt;   &lt;article&gt;     &lt;h2&gt;Article Title&lt;/h2&gt;     &lt;p&gt;Article paragraph 1&lt;/p&gt;     &lt;p&gt;Article paragraph 2&lt;/p&gt;   &lt;/article&gt;&lt;aside&gt;     &lt;h3&gt;Sidebar Title&lt;/h3&gt;     &lt;p&gt;Sidebar paragraph&lt;/p&gt;   &lt;/aside&gt; &lt;/main&gt; </pre> <p>No whitespace between the column elements</p>

In this case, both the article element and the aside element are assigned `display: inline-block`, which places them beside each other on the left side of the main element. You also need to assign a width value to each element to set the size of your columns.

Notice, too, that in the HTML code, I crammed the `</article>` end tag and the `<aside>` start tag together so that there's no whitespace between them. This is crucial when working with inline blocks because otherwise, the browser will add a bit of space when it renders the elements, which can mess up your width calculations.



# Learning Page Layout Basics


You've no doubt noticed that as with floats, the bottom borders of the two columns don't line up. You'll use the same workaround to fix that problem.

Finally, add the footer element, as shown in the example that follows.

## ► Example

➡ Online: [wdpg.io/11-2-4](http://wdpg.io/11-2-4)

*This example shows how to add the footer element to the bottom of the page, although in this case, there's no need to clear anything. Colors have been added to all the elements, but most aren't shown in the code.*

WEB PAGE	
CSS	<pre>footer {   padding: 1em;   background-color: #b6d7a8; }</pre> <p><i>No clear is needed with inline blocks.</i></p>
HTML	<pre>&lt;footer&gt;   &lt;p&gt;Footer paragraph&lt;/p&gt; &lt;/footer&gt;</pre>

## PLAY

*How would you modify this layout to display the sidebar on the left instead of the right? ➡ Online: [wdpg.io/11-2-6](http://wdpg.io/11-2-6)*

As I did with the floated layout, I made the article and aside columns appear to be the same height by using faux columns. (Note that these faux columns work properly only as long as the article element is taller than the aside element.)

Using inline blocks to lay out web page content isn't common despite the ease with which you can create fairly sophisticated layouts. Inline blocks have their drawbacks, of course. You have to watch your vertical alignment; you often have to ensure that there's no whitespace between the blocks; and you can't send elements to the right side of the parent as you can with `float: right`. To solve these problems and gain extra power over your layouts, you need to shun these old technologies in favor of the newest layout kid on the block: flexbox. You learn everything you need to know in Chapter 12.



## Summary

- You can make your pages more semantic by using the HTML5 page layout tags: `<header>`, `<nav>`, `<main>`, `<article>`, `<section>`, `<aside>`, and `<footer>`.
- To use a float-based layout, let the elements flow in the default manner; then, when you need two or more elements to appear beside each other, float them (usually to the left). Remember that when you want to resume the default page flow, clear the floats.
- To use an inline block-based layout, let the page elements flow in the default manner; then, when you need two or more elements to appear beside each other, display them as inline blocks.

## PLAY

*How would you modify this layout to display three content columns: a sidebar to the left and to the right of the article element?* ➡ Online: [wdpg.io/11-2-7](http://wdpg.io/11-2-7)