

Advanced CSS: Layout

Chapter 7

Chapter 7

1 Normal Flow

2 Positioning Elements

3 Floating Elements

4 Constructing Multicolumn Layouts

5 Approaches to CSS Layouts

6 Responsive Design

7 Filters, Transitions, and Animations

8 CSS Frameworks and Preprocessors

Chapter 7

1 Normal Flow

2 Positioning Elements

3 Floating Elements

4 Constructing Multicolumn Layouts

5 Approaches to CSS Layouts

6 Responsive Design

7 Filters, Transitions, and Animations

8 CSS Frameworks and Preprocessors

Normal Flow

To understand CSS positioning and layout, it is essential that we understand this distinction as well as the idea of **normal flow**:

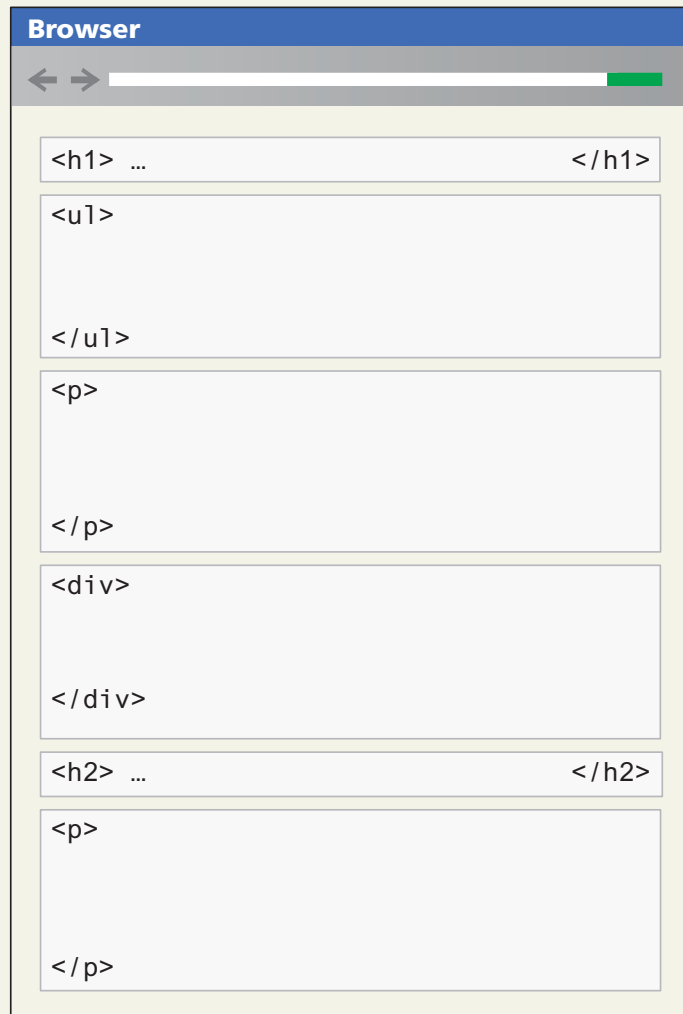
how the browser will normally display block-level elements and inline elements from left to right and from top to bottom

Normal Flow

- **Block-level elements** such as `<p>`, `<div>`, `<h2>`, ``, and `<table>` are each contained on their own line.
- **Inline elements** do not form their own blocks but instead are displayed within lines.

Normal Flow

Block-Level Elements



Each block exists on its own line and is displayed in normal flow from the browser window's top to its bottom.

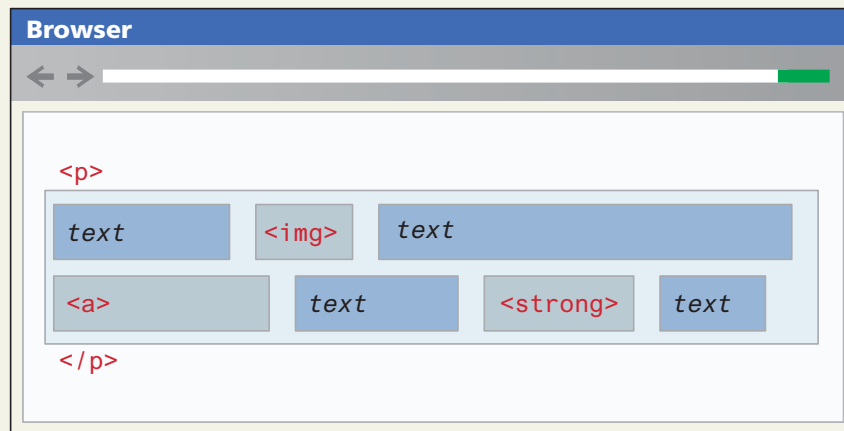
By default each block-level element fills up the entire width of its parent (in this case, it is the `<body>`, which is equivalent to the width of the browser window).

You can use CSS box model properties to customize, for instance, the width of the box and the margin space between other block-level elements.

Normal Flow

Inline Elements

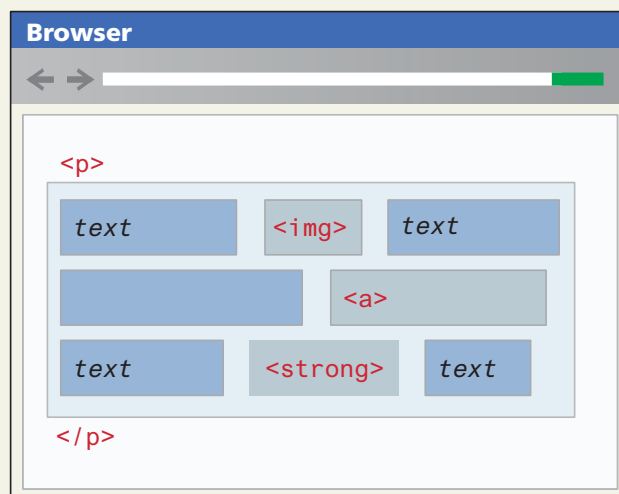
```
<p>  
This photo  of Conservatory Pond in  
<a href="http://www.centralpark.com/">Central Park</a> New York City  
was taken on October 22, 2015 with a <strong>Canon EOS 30D</strong>  
camera.  
</p>
```



Inline content is laid out horizontally left to right within its container.

Once a line is filled with content, the next line will receive the remaining content, and so on.

Here the content of this <p> element is displayed on two lines.

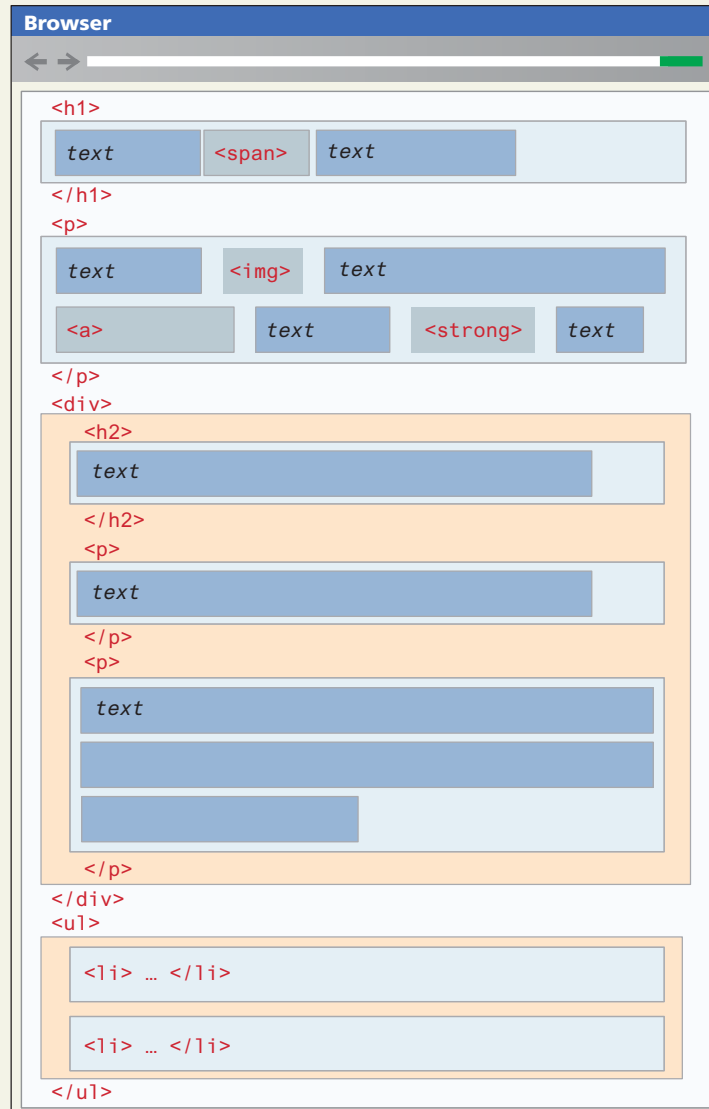


If the browser window resizes, then inline content will be "reflowed" based on the new width.

Here the content of this <p> element is now displayed on three lines.

Normal Flow

Block and Inline Elements



A document consists of block-level elements stacked from top to bottom.

Within a block, inline content is horizontally placed left to right.

Some block-level elements can contain other block-level elements (in this example, a `<div>` can contain other blocks).

In such a case, the block-level content inside the parent is stacked from top to bottom within the container (`<div>`).

Chapter 7

1 Normal Flow

2 Positioning Elements

3 Floating Elements

4 Constructing Multicolumn Layouts

5 Approaches to CSS Layouts

6 Responsive Design

7 Filters, Transitions, and Animations

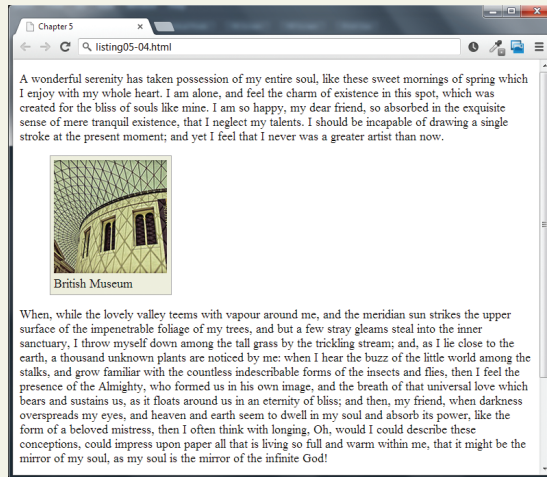
8 CSS Frameworks and Preprocessors

Positioning Elements

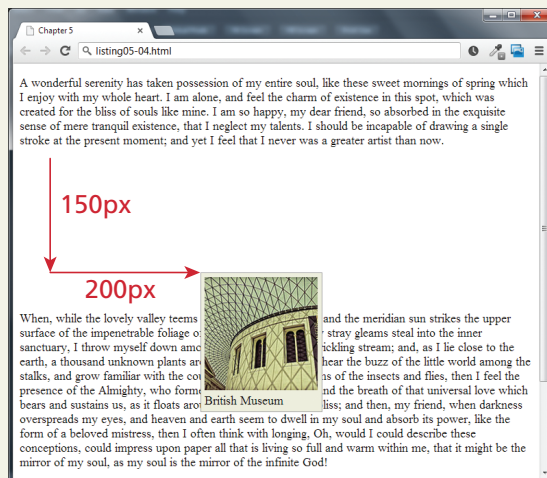
- **absolute** The element is removed from normal flow and positioned in relation to its nearest positioned ancestor.
- **fixed** The element is fixed in a specific position in the window even when the document is scrolled.
- **relative** The element is moved relative to where it would be in the normal flow.
- **static** The element is positioned according to the normal flow. This is the default.

Positioning Elements

Relative Positioning



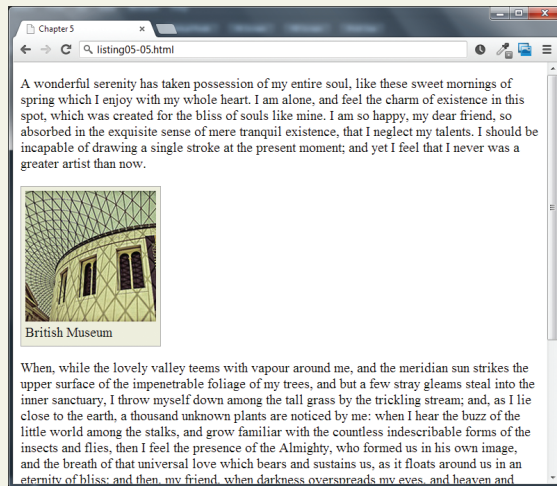
```
<p>A wonderful serenity has taken possession of my ...  
<figure>  
    
  <figcaption>British Museum</figcaption>  
</figure>  
<p>When, while the lovely valley ...
```



```
figure {  
  border: 1pt solid #A8A8A8;  
  background-color: #EDEDDE;  
  padding: 5px;  
  width: 150px;  
  position: relative;  
  top: 150px;  
  left: 200px;  
}
```

Positioning Elements

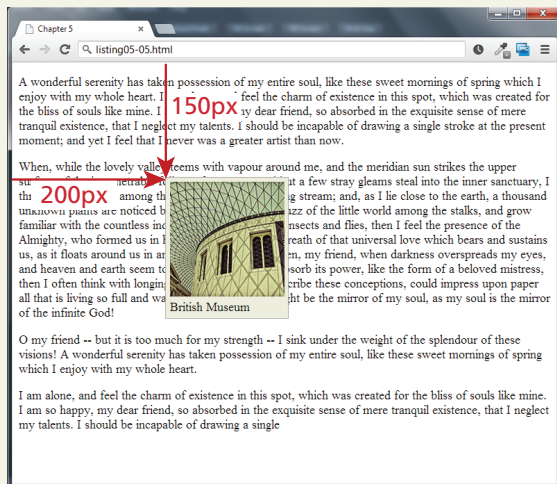
Absolute Positioning



```
<p>A wonderful serenity has taken possession of my ...
```

```
<figure>  
    
  <figcaption>British Museum</figcaption>  
</figure>
```

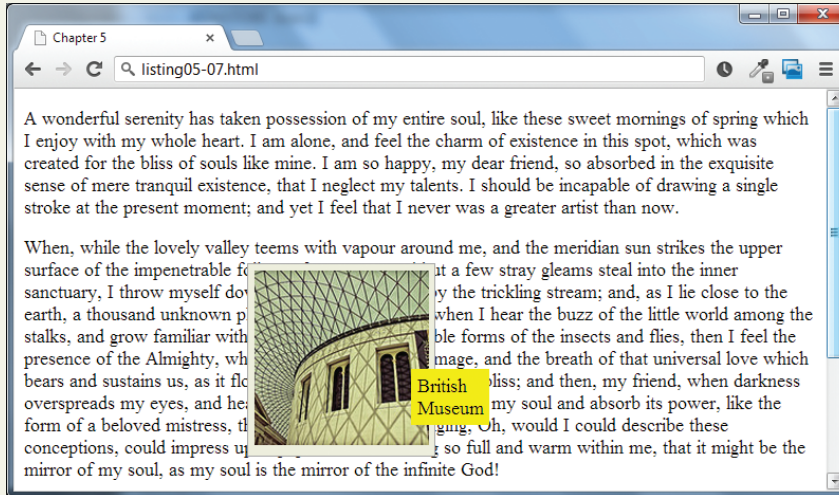
```
<p>When, while the lovely valley ...
```



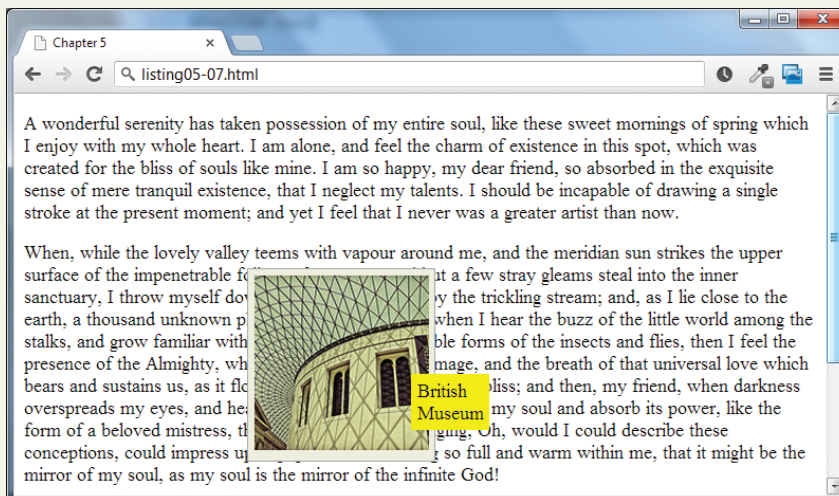
```
figure {  
  margin: 0;  
  border: 1pt solid #A8A8A8;  
  background-color: #EDEDDE;  
  padding: 5px;  
  width: 150px;  
  position: absolute;  
  top: 150px;  
  left: 200px;  
}
```


Positioning Elements

Z-Index



```
figure {  
    position: absolute;  
    top: 150px;  
    left: 200px;  
}  
figcaption {  
    position: absolute;  
    top: 90px;  
    left: 140px;  
}
```

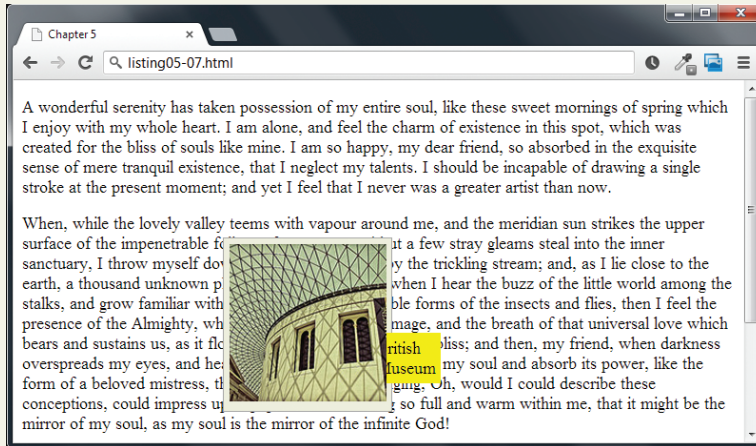


```
figure {  
    ...  
    z-index: 5;  
}  
figcaption {  
    ...  
    z-index: 1;  
}
```

Note that this did **not** move the `<figure>` on top of the `<figcaption>` as one might expect. This is due to the nesting of the caption within the figure.

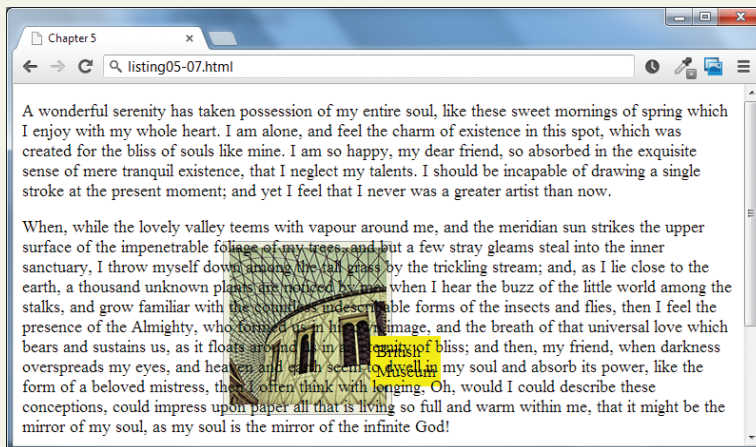
Positioning Elements

Z-Index



```
figure {  
    ...  
    z-index: 1;  
}  
figcaption {  
    ...  
    z-index: -1;  
}
```

Instead the `<figcaption>` `z-index` must be set below 0. The `<figure>` `z-index` could be any value equal to or above 0.



```
figure {  
    ...  
    z-index: -1;  
}  
figcaption {  
    ...  
    z-index: 1;  
}
```

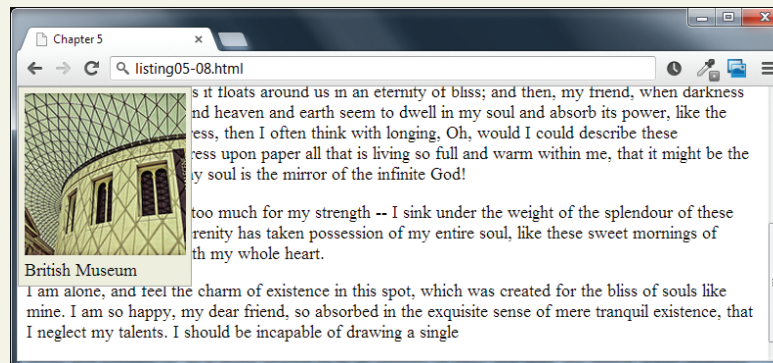
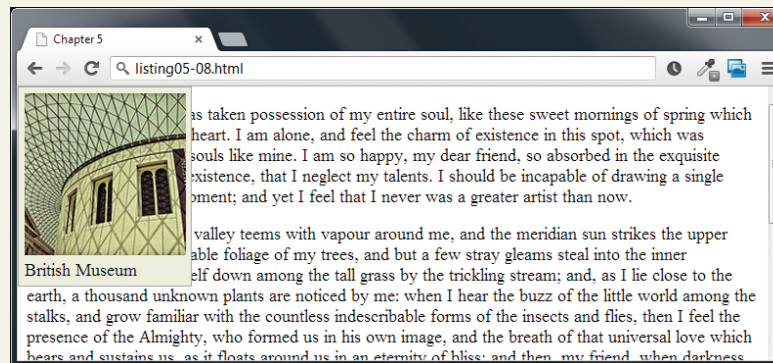
If the `<figure>` `z-index` is given a value less than 0, then any of its positioned descendants change as well. Thus both the `<figure>` and `<figcaption>` move underneath the body text.

Positioning Elements

Fixed Position

```
figure {  
    . . .  
    position: fixed;  
    top: 0;  
    left: 0;  
}
```

Notice that figure is fixed in its position regardless of what part of the page is being viewed.



Chapter 7

1 Normal Flow

2 Positioning Elements

3 Floating Elements

4 Constructing Multicolumn Layouts

5 Approaches to CSS Layouts

6 Responsive Design

7 Filters, Transitions, and Animations

8 CSS Frameworks and Preprocessors

Floating Elements

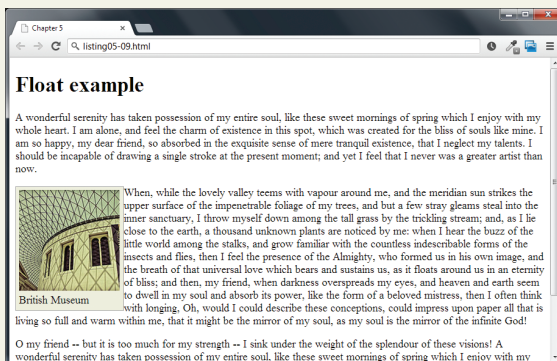
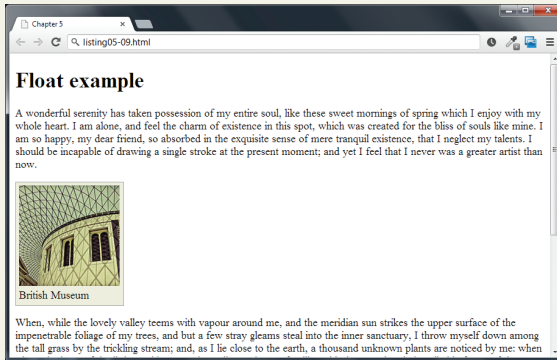
Floating within a Container

Floating Elements

It is possible to displace an element out of its position in the normal flow via the CSS **float** property

- An element can be floated to the left or floated to the right .
- it is moved all the way to the far left or far right of its containing block and the rest of the content is “reflowed” around the floated element

Floating Elements



```
<h1>Float example</h1>
<p>A wonderful serenity has taken ...</p>
<figure>
  
  <figcaption>British Museum</figcaption>
</figure>
<p>When, while the lovely valley ...</p>
```

```
figure {
  border: 1pt solid #A8A8A8;
  background-color: #EDEDDE;
  margin: 0;
  padding: 5px;
  width: 150px;
}
```

Notice that a floated block-level element **should** have a width specified.

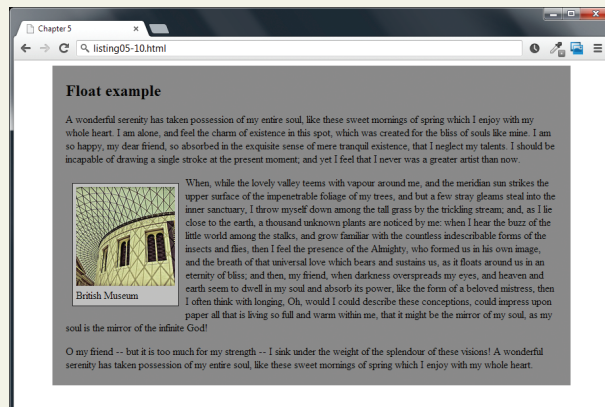
```
figure {
  ...
  width: 150px;
  float: left;
}
```

```
figure {
  ...
  width: 150px;
  float: right;
  margin: 10px;
}
```

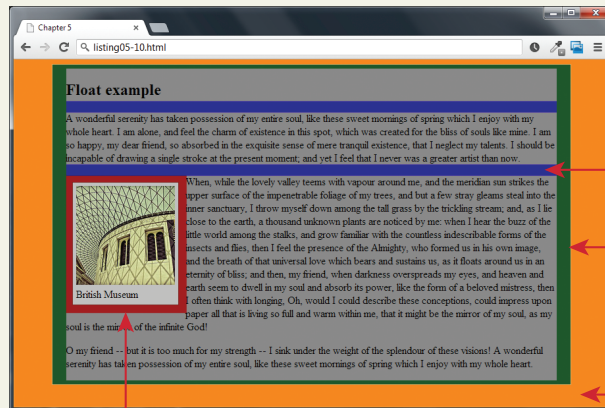

Floating Elements

Floating within a Container

```
<article>
  <h1>Float example</h1>
  <p>A wonderful serenity has taken possession of ... </p>
  <figure>
    
    <figcaption>British Museum</figcaption>
  </figure>
  <p>When, while the lovely valley teems with ...</p>
  <p>O my friend -- but it is too much for my ...</p>
</article>
```

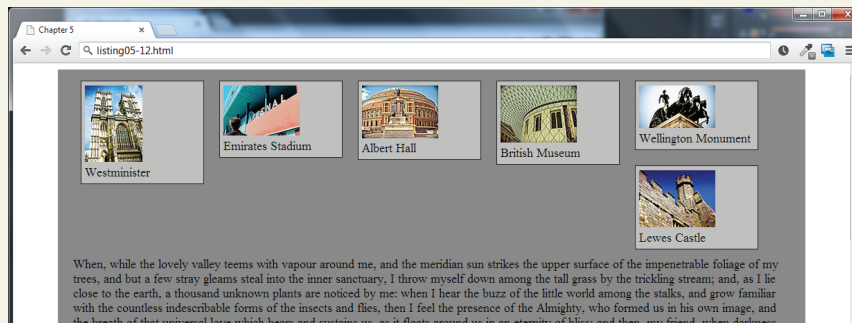
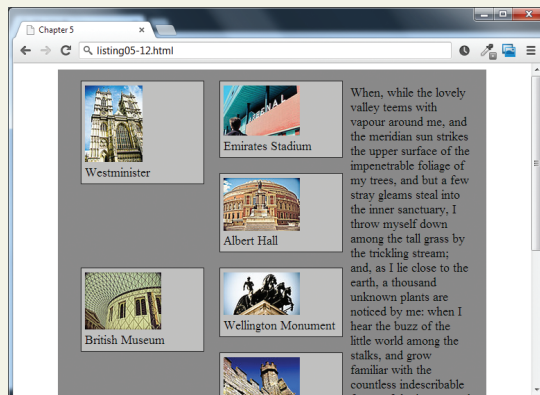
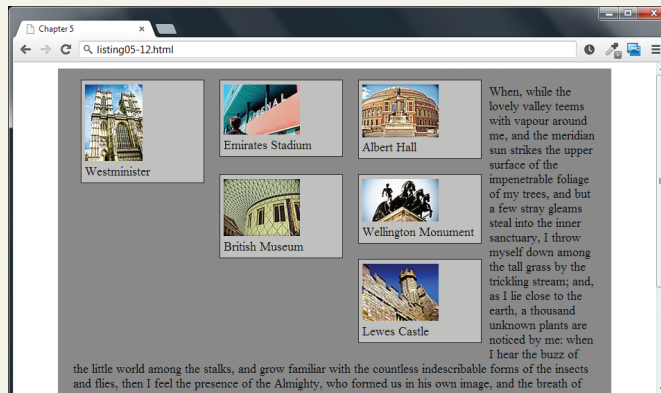


```
article {
  background-color: #898989;
  margin: 5px 50px;
  padding: 5px 20px;
}
p { margin: 16px 0; }
figure {
  border: 1pt solid #262626;
  background-color: #c1c1c1;
  padding: 5px;
  width: 150px;
  float: left;
  margin: 10px;
}
```



Floating Elements

Floating Multiple Items Side by Side



```
<article>
<figure>
  
  <figcaption>Westminster</figcaption>
</figure>
<figure>
  
  <figcaption>Emirates Stadium</figcaption>
</figure>
<figure>
  
  <figcaption>Albert Hall</figcaption>
</figure>
<figure>
  
  <figcaption>British Museum</figcaption>
</figure>
<figure>
  
  <figcaption>Wellington Monument</figcaption>
</figure>
<figure>
  
  <figcaption>Lewes Castle</figcaption>
</figure>
<p>When, while the lovely valley teems ..
</article>
```

```
figure {
  width: 150px;
  float: left;
}
```

As the window resizes, the content in the containing block (the `<article>` element), will try to fill the space that is available to the right of the floated elements.

Floating Elements

Floating Multiple Items Side by Side

Thankfully, you can stop elements from flowing around a floated element by using the **clear** property

```
.first { clear: left; }
```



```
<article>
  <figure>
    
    <figcaption>Westminister</figcaption>
  </figure>
  <figure>
    
    <figcaption>Emirates Stadium</figcaption>
  </figure>
  <figure>
    
    <figcaption>Albert Hall</figcaption>
  </figure>
  <figure class="first">
    
    <figcaption>British Museum</figcaption>
  </figure>
  <figure>
    
    <figcaption>Wellington Monument</figcaption>
  </figure>
  <figure>
    
    <figcaption>Lewes Castle</figcaption>
  </figure>
  <p class="first">When, while the lovely ...
</article>
```

Floating Elements

Clear property

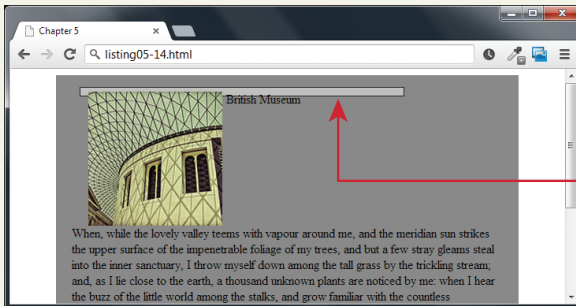
- **left** The left-hand edge of the element cannot be adjacent to another element.
- **right** The right-hand edge of the element cannot be adjacent to another element.
- **both** Both the left-hand and right-hand edges of the element cannot be adjacent to another element.
- **none** The element can be adjacent to other elements.

Floating Elements

Containing Floats

Another problem that can occur with floats is when an element is floated within a containing block that contains only floated content. In such a case, the containing block essentially disappears

```
<article>
  <figure>
    
    <figcaption>British Museum</figcaption>
  </figure>
  <p class="first">When, while the lovely valley ...
</article>
```



Notice that the `<figure>` element's content area has shrunk down to zero (it now just has padding space and borders).

```
figure img {
  width: 170px;
  margin: 0 5px;
  float: left;
}
figure figcaption {
  width: 100px;
  float: left;
}
figure {
  border: 1pt solid #262626;
  background-color: #c1c1c1;
  padding: 5px;
  width: 400px;
  margin: 10px;
}
.first { clear: left; }
```

Floating Elements

Overlaying and Hiding Element

One of the more common design tasks with CSS is to place two elements on top of each other, or to selectively hide and display elements

In such a case, relative positioning is used to create the **positioning context** for a subsequent absolute positioning move.

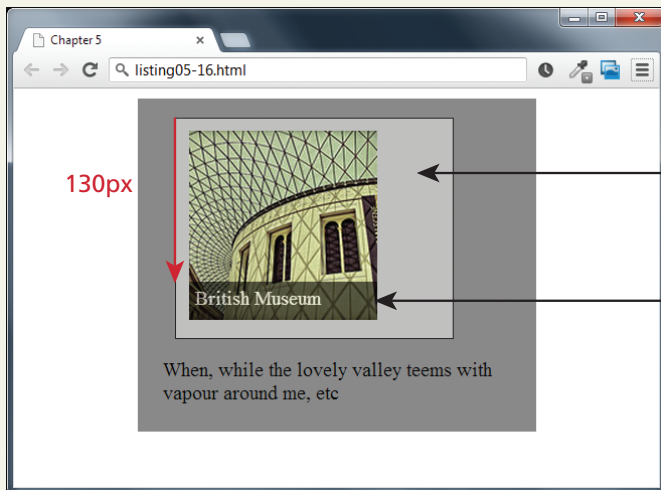
Floating Elements

Overlaying and Hiding Element



```
figure {  
  border: 1pt solid #262626;  
  background-color: #c1c1c1;  
  padding: 10px;  
  width: 200px;  
  margin: 10px;  
}
```

```
figcaption {  
  background-color: black;  
  color: white;  
  opacity: 0.6;  
  width: 140px;  
  height: 20px;  
  padding: 5px;  
}
```

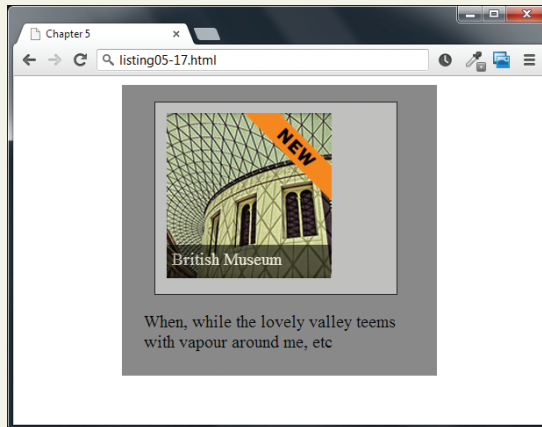


```
figure {  
  ...  
  position: relative; This creates the  
  } positioning context.  
figcaption {  
  ...  
  position: absolute; This does the actual  
  top: 130px; move.  
  left: 10px;  
  }
```

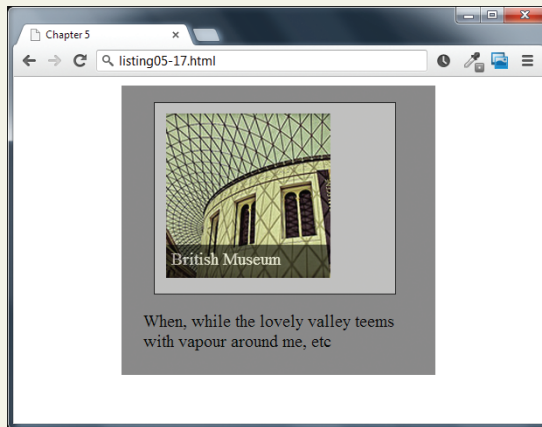
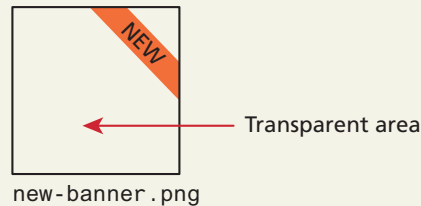
Floating Elements

Using display

```
<figure>
  
  <figcaption>British Museum</figcaption>
  
</figure>
```



```
.overlaid {
  position: absolute;
  top: 10px;
  left: 10px;
}
```



```
.overlaid {
  position: absolute;
  top: 10px;
  left: 10px;
  display: none;
}
```

This hides the overlaid image.

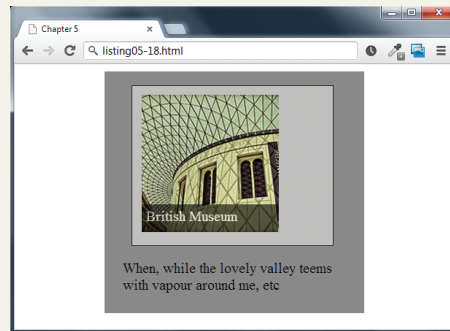
```
.hide {
  display: none;
}
```

This is the preferred way to hide: by adding this additional class to the element. This makes it clear in the markup that the element is not visible.

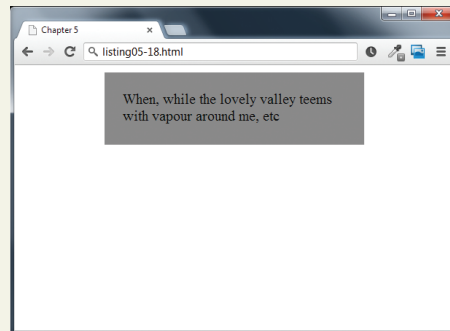
```
<img ... class="overlaid hide"/>
```


Floating Elements

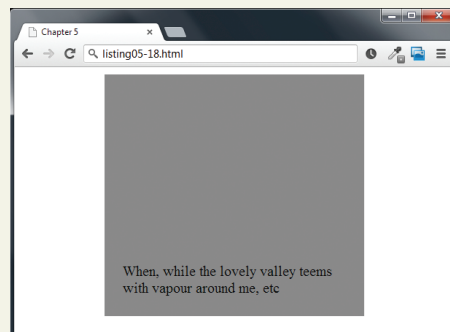
Comparing visibility with display



```
figure {  
  ...  
  display: auto;  
}
```



```
figure {  
  ...  
  display: none;  
}
```

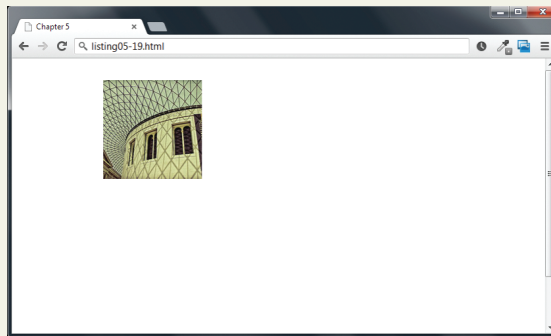


```
figure {  
  ...  
  visibility: hidden;  
}
```

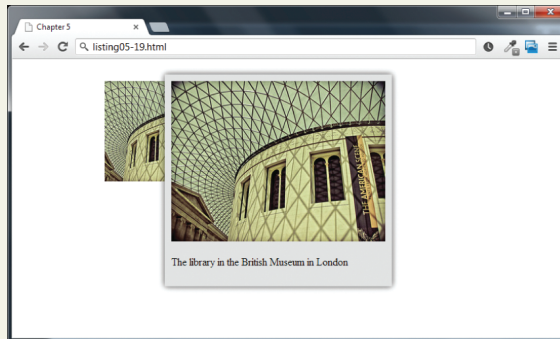
Floating Elements

Using Hover with display

```
<figure class="thumbnail">
  
  <figcaption class="popup">
    
    <p>The library in the British Museum in London</p>
  </figcaption>
</figure>
```



When the page is displayed, the larger version of the image, which is within the `<figcaption>` element, is hidden.



When the user moves/hovers the mouse over the thumbnail image, the `visibility` property of the `<figcaption>` element is set to `visible`.

```
figcaption.popup {
  padding: 10px;
  background: #e1e1e1;
  position: absolute;

  /* add a drop shadow to the frame */
  box-shadow: 0 0 15px #A9A9A9;

  /* hide it until there is a hover */
  visibility: hidden;
}
```

```
figure.thumbnail:hover figcaption.popup {
  position: absolute;
  top: 0;
  left: 100px;

  /* display image upon hover */
  visibility: visible;
}
```

Chapter 7

1 Normal Flow

2 Positioning Elements

3 Floating Elements

4 Constructing Multicolumn Layouts

5 Approaches to CSS Layouts

6 Responsive Design

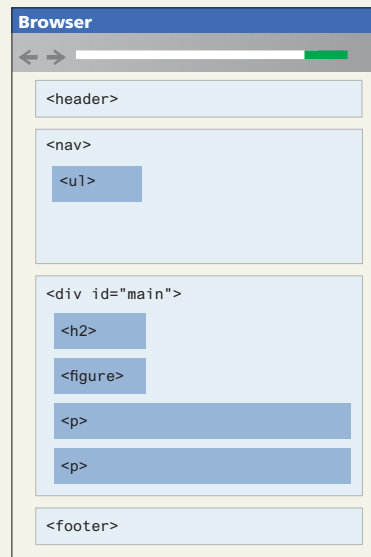
7 Filters, Transitions, and Animations

8 CSS Frameworks and Preprocessors

Constructing Multicolumn Layout

Using Floats to Create Columns

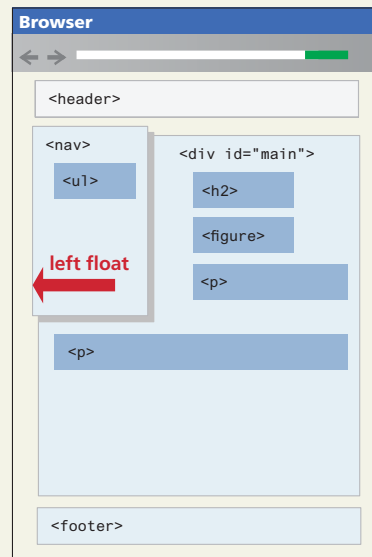
1 HTML source order (normal flow)



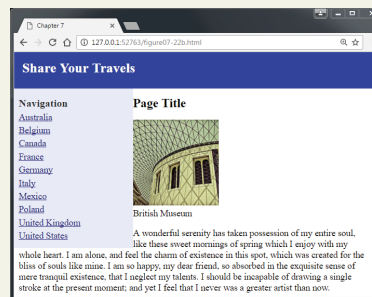
Constructing Multicolumn Layout

Using Floats to Create Columns

2 Two-column layout (left float)



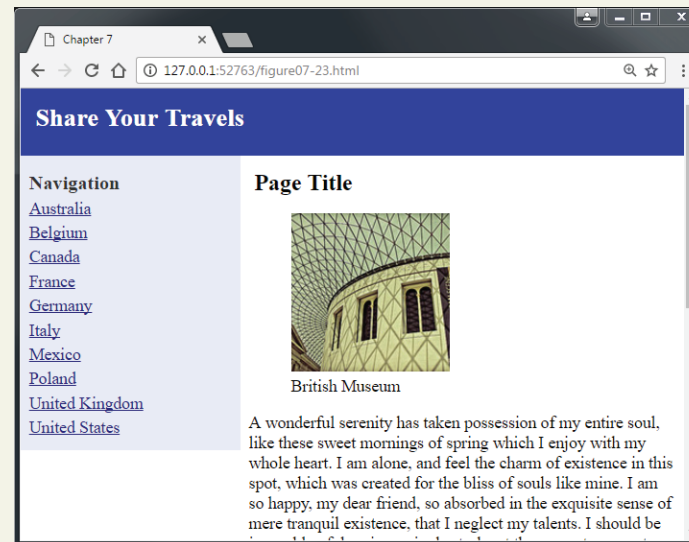
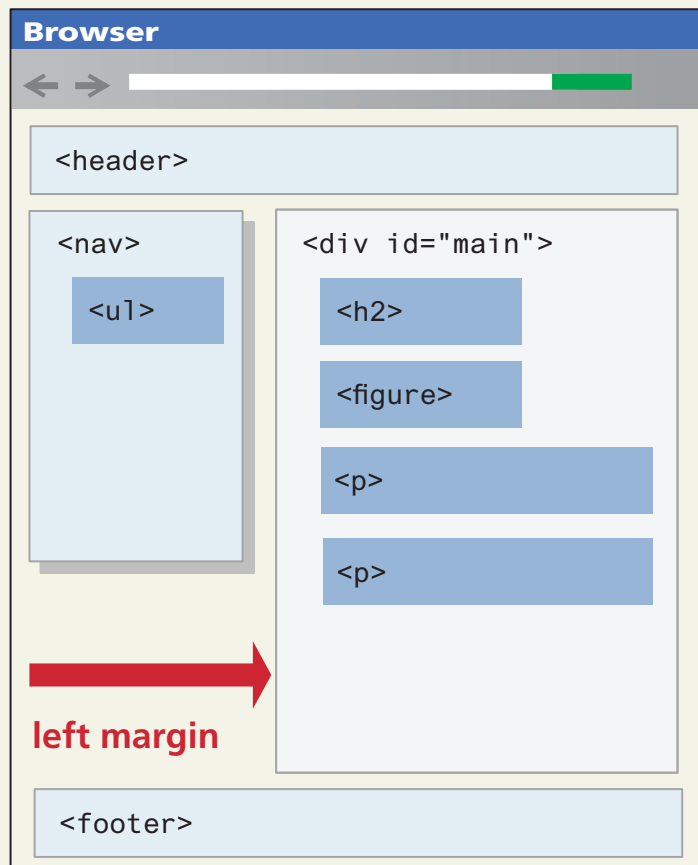
```
nav {  
  ...  
  width: 12em;  
  float: left;  
}
```



Constructing Multicolumn Layout

Using Floats to Create Columns

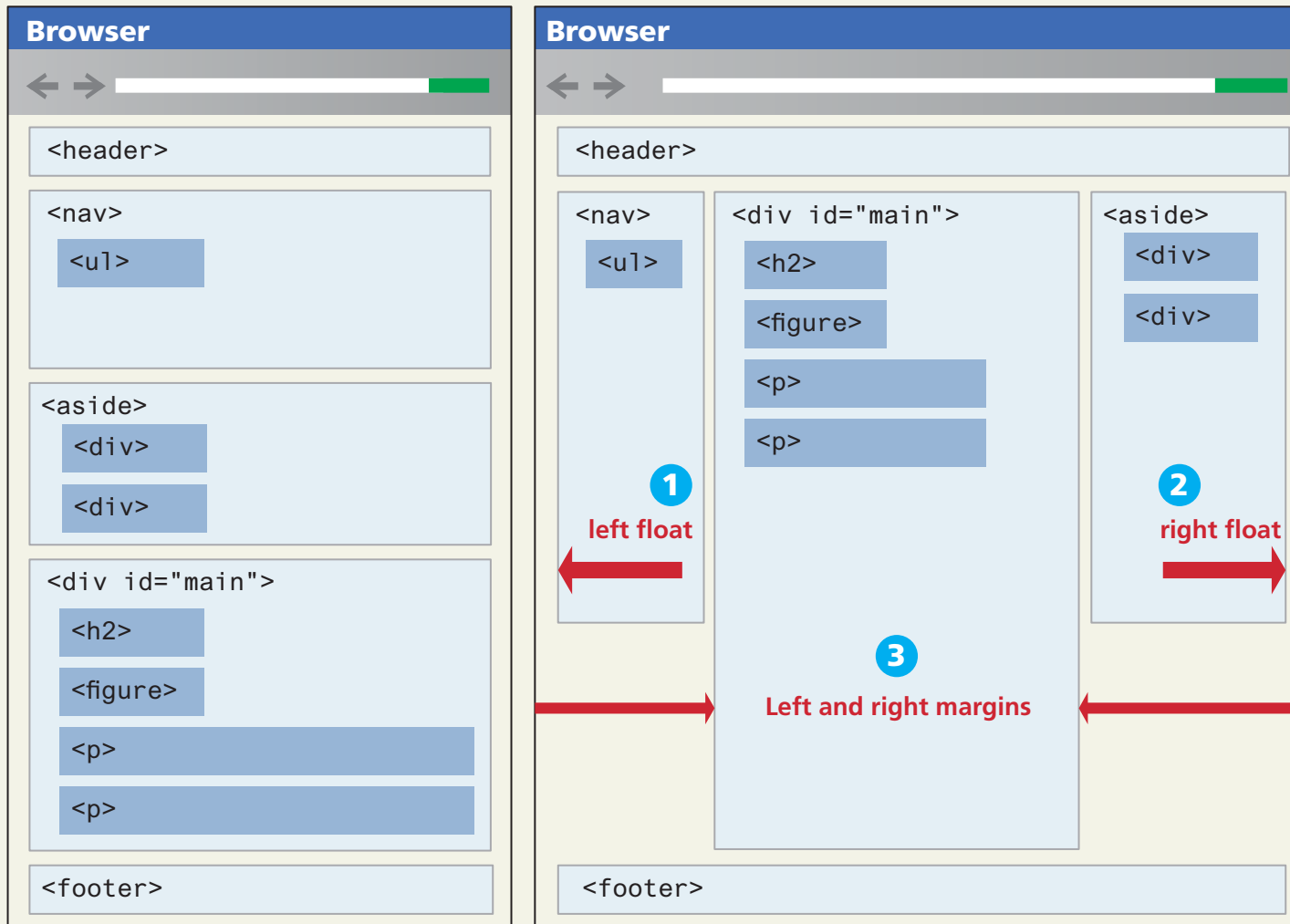
- 3 Set the left margin of non-floated content



```
div#main {
  ...
  margin-left: 13em;
}
```

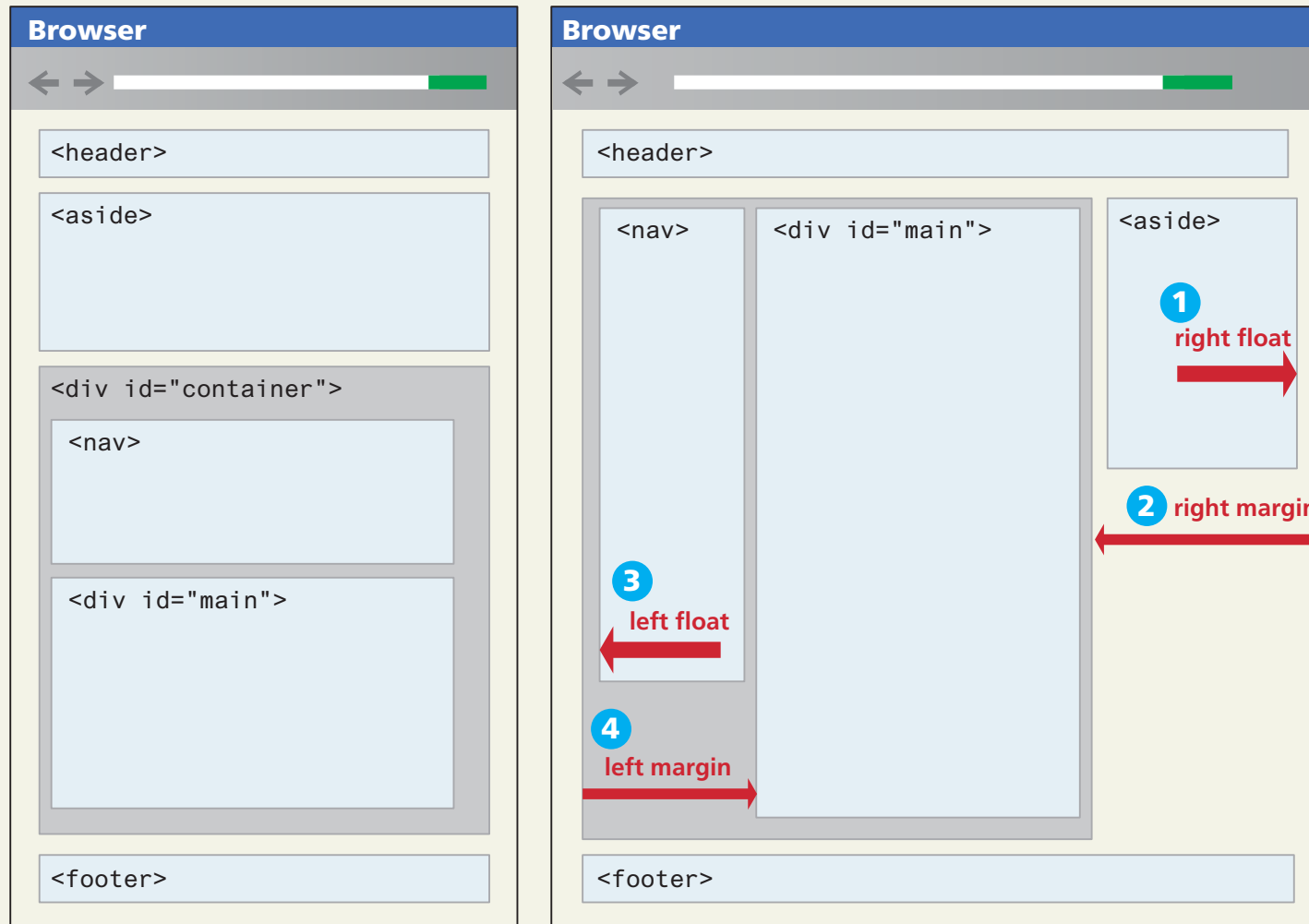
Constructing Multicolumn Layout

3 column example



Constructing Multicolumn Layout

3 column example with nested floats



Constructing Multicolumn Layout

Using Positioning to Create Columns

The image shows two browser window diagrams illustrating the construction of a multicolumn layout using CSS positioning.

Left Browser Window: Shows the initial HTML structure. The content is organized into a container with a main area, a navigation bar, and an aside, all stacked vertically. The HTML code is as follows:

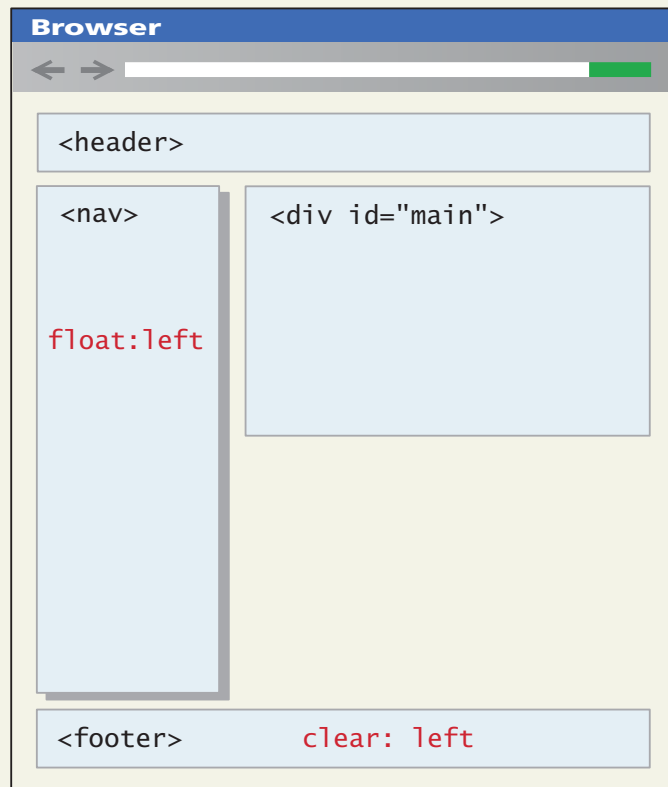
```
<header>  
<div id="container">  
  <div id="main">  
  <nav>  
  <aside>  
</div>  
<footer>
```

Right Browser Window: Shows the same structure with CSS styling applied to create a multicolumn layout. The styling is as follows:

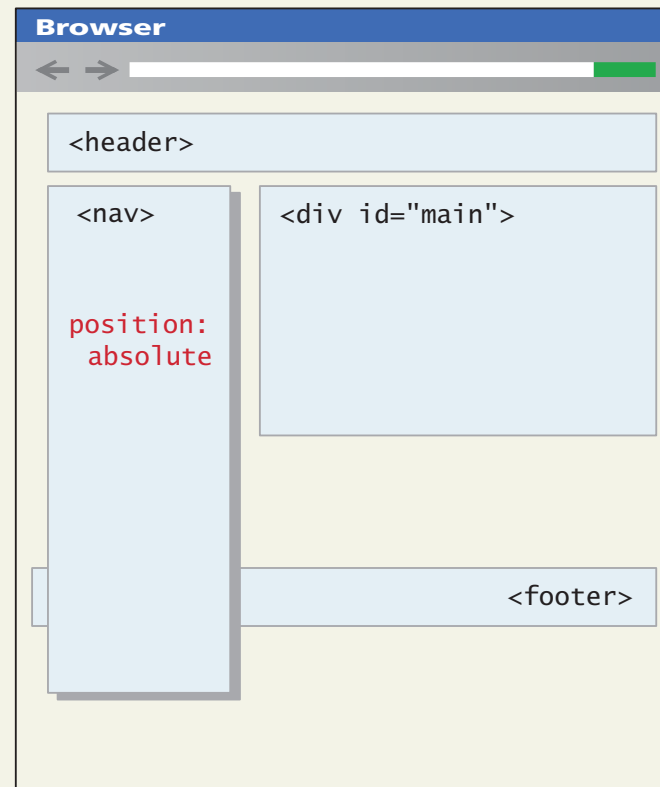
- 1** `position: relative;` (Applied to the container)
- 2** `position: absolute; left: 0; top: 0;` (Applied to the navigation bar)
- 3** `position: absolute; right: 0; top: 0;` (Applied to the aside)
- 4** `left and right margins` (Applied to the main area, indicated by red arrows pointing inward from the navigation bar and aside)

Constructing Multicolumn Layout

Problems with Absolute positioning



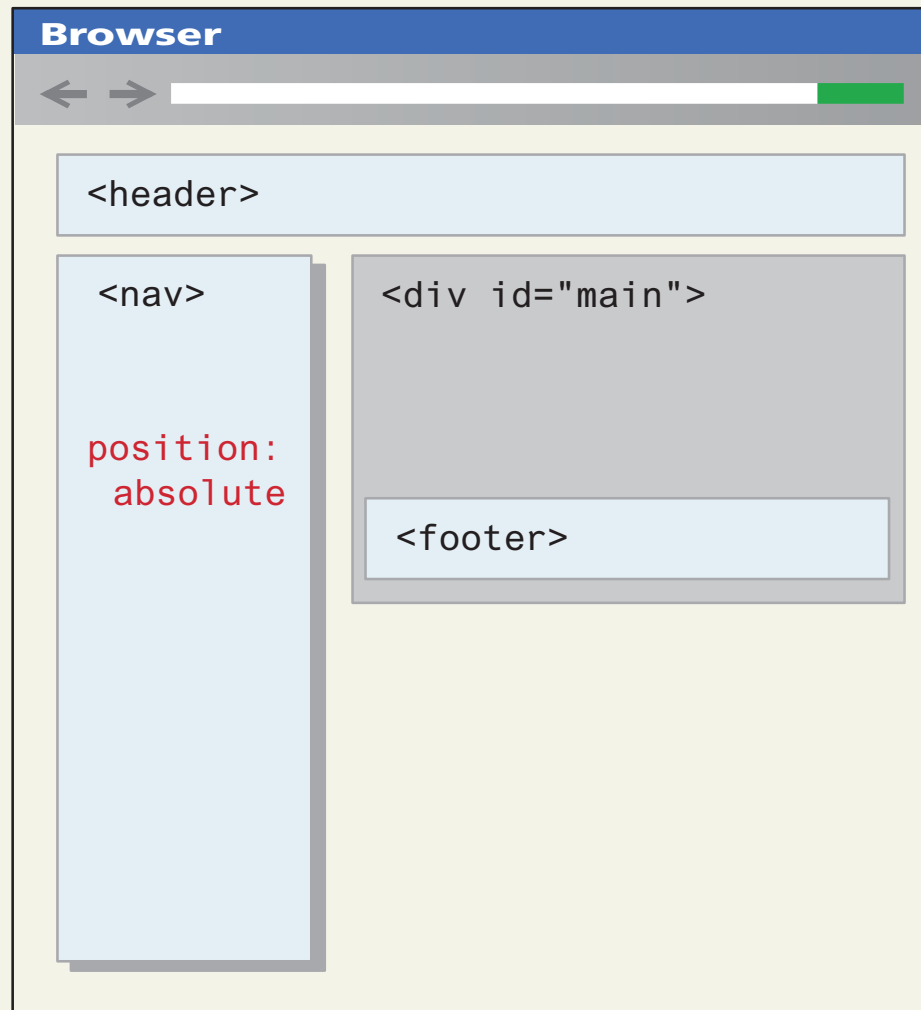
Elements that are floated leave behind space for them in the normal flow. We can also use the `clear` property to ensure later elements are below the floated element.



Absolute positioned elements are taken completely out of normal flow, meaning that the positioned element may overlap subsequent content. The `clear` property will have no effect since it only responds to floated elements.


Constructing Multicolumn Layout

Solution to footer problem



Constructing Multicolumn Layout

Using Flexbox to Create Columns



Fall in Calgary

Nunc nec fermentum dolor. Duis at iaculis turpis. Sed rutrum elit ac egestas dapibus. Duis nec consequat enim.

Mauris porta arcu id magna adipiscing lacinia at congue lacus. Vivamus blandit quam quis tincidunt egestas. Etiam posuere lectus sed sapien malesuada molestie.

Phasellus vel felis purus. Aliquam consequat pellentesque dui, non mollis erat dictum sit amet. Curabitur non quam dictum, consectetur arcu in, vehicula justo.

```
<div class="media">
  
  <div class="media-body">
    <h2>Fall in Calgary</h2>
    <p>Nunc nec fermentum dolor...</p>
    <p>Mauris porta arcu id...</p>
    <p>Phasellus vel felis purus...</p>
  </div>
</div>
```

Prior to flexbox, one would create such a layout within a container using floats plus margins. The problem with this approach is that margins needed to be in pixels and had to exactly match image size. If image size changed (or you wanted same kind of style elsewhere), you had to modify the style.

```
.media-image {
  float: left;
  margin-right: 10px;
}
.media-body {
  margin-left: 160px;
}
```



Fall in Calgary

Nunc nec fermentum dolor. Duis at iaculis turpis. Sed rutrum elit ac egestas dapibus. Duis nec consequat enim.

Mauris porta arcu id magna adipiscing lacinia at congue lacus. Vivamus blandit quam quis tincidunt egestas. Etiam posuere lectus sed sapien malesuada molestie.

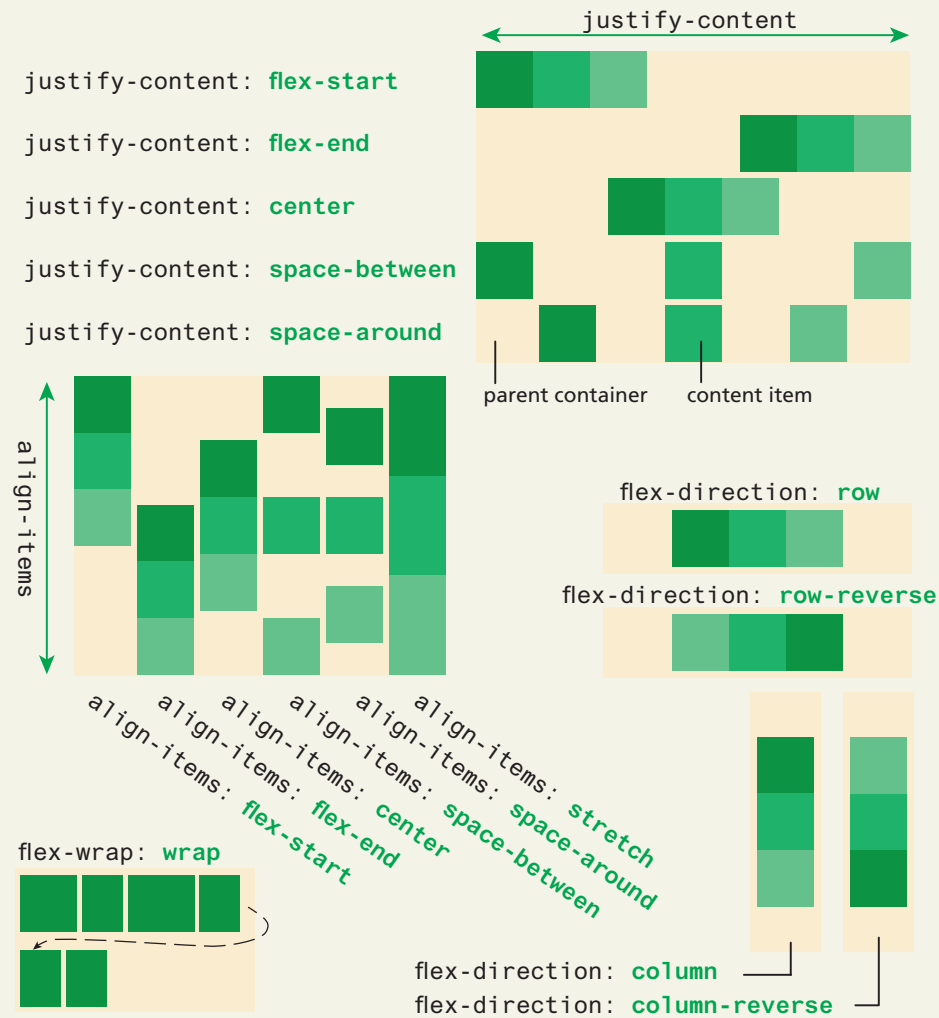
Phasellus vel felis purus. Aliquam consequat pellentesque dui, non mollis erat dictum sit amet. Curabitur non quam dictum, consectetur arcu in, vehicula justo.

```
.media {
  display: flex;
  align-items: flex-start;
}
.media-image {
  margin-right: 1em;
}
```

Using flexbox, we now have a much more generalized (and thus reusable) style.

Constructing Multicolumn Layout

The flexbox parent (container) properties



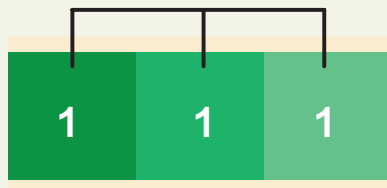
Constructing Multicolumn Layout

The flexbox child (item) properties

flex-grow: 1
flex-shrink: 1
flex-basis: auto

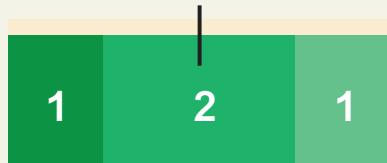
flex: 1 1 auto

These can be combined into the shorthand property instead



When the flex-grow value of each item is greater than 0, then each item will grow equally to fill the parent container.

flex-grow: 2

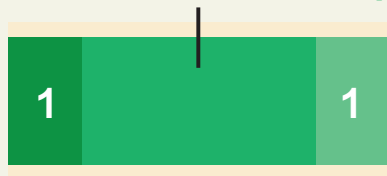


Defines the growth factor of an element relative to the other items.

$\text{width} = n$

$\text{width} = n \times 2$

flex-basis: 200px



Defines the default size of the element before the remaining space is distributed.

Chapter 7

1 Normal Flow

2 Positioning Elements

3 Floating Elements

4 Constructing Multicolumn Layouts

5 Approaches to CSS Layouts

6 Responsive Design

7 Filters, Transitions, and Animations

8 CSS Frameworks and Preprocessors

Approaches to CSS Layout

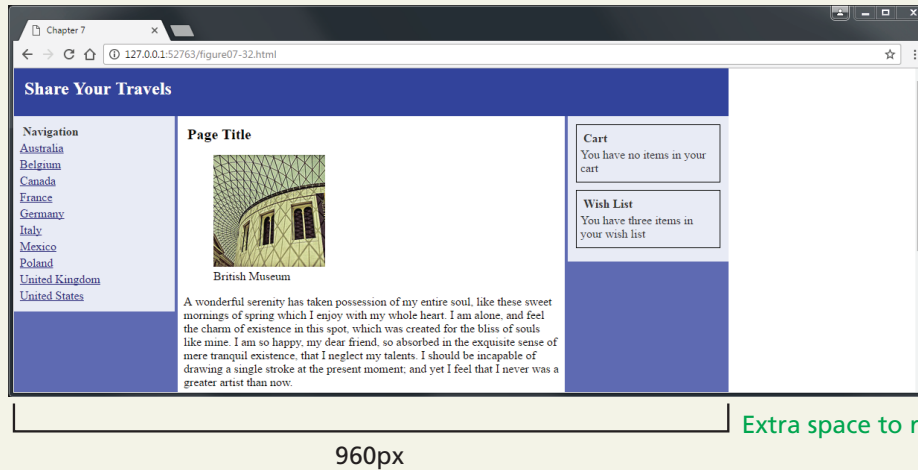
Fixed Layout

In a fixed layout , the basic width of the design is set by the designer, typically corresponding to an “ideal” width based on a “typical” monitor resolution.

The advantage of a fixed layout is that it is easier to produce and generally has a predictable visual result.

Approaches to CSS Layout

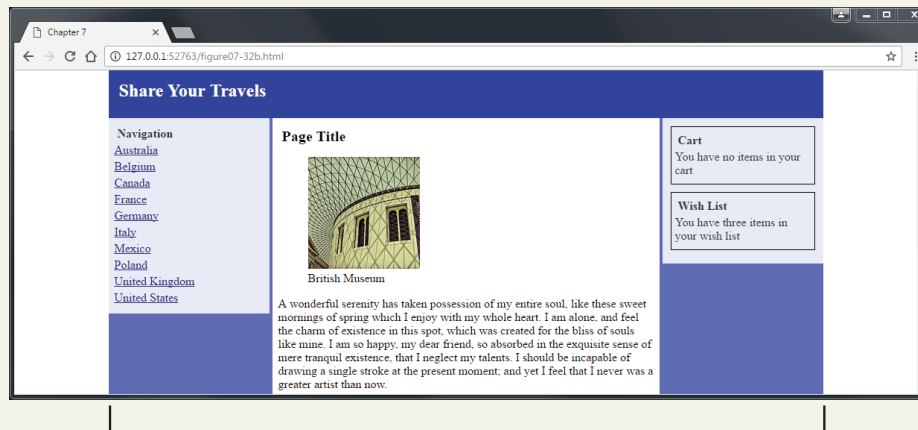
Fixed Layout



Extra space to right

```
div#wrapper {  
  width: 960px;  
  background-color: blue;  
}
```

```
<body>  
  <div id="wrapper">  
    <header>  
      ...  
    </header>  
    <div id="main">  
      ...  
    </div>  
    <footer>  
      ...  
    </footer>  
  </div>  
</body>
```



Equal space to the left and to right

```
div#wrapper {  
  width: 960px;  
  margin-left: auto;  
  margin-right: auto;  
  background-color: blue;  
}
```

Approaches to CSS Layout

Problem with Fixed Layout



The problem with fixed layouts is that they don't adapt to smaller viewports.

Approaches to CSS Layout

Liquid Layout

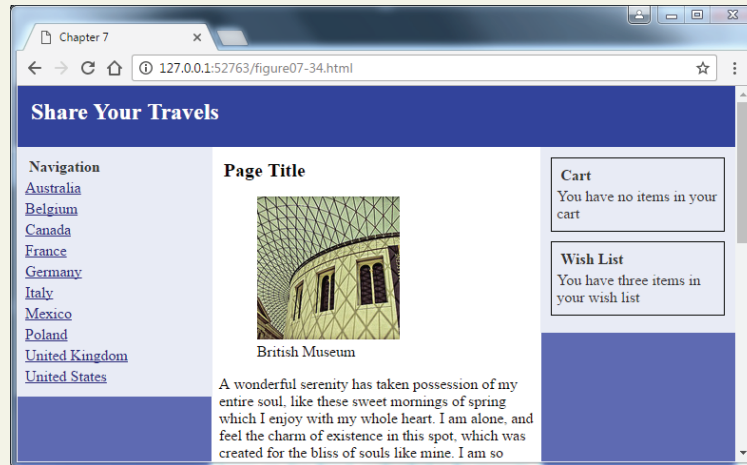
liquid layout (also called a fluid layout) widths are not specified using pixels, but percentage values

advantage of a liquid layout is that it adapts to different browser sizes

creating a usable liquid layout is generally more difficult than creating a fixed layout

Approaches to CSS Layout

Liquid Layout



Fluid layouts are based on the browser window.

However, elements can get too spread out as the browser expands.

