Advanced CSS: Layout

Chapter 7

Randy Connolly and Ricardo Hoar

Fundamentals of Web Development

http://www.funwebdev.com

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>, , <h2>, <ul>, 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.

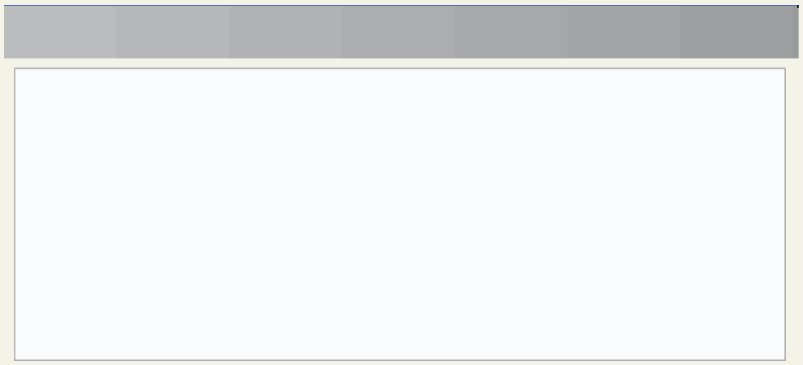
By default each block-level element fills up the entire width of its parent (in this case, it is the <body>, which is equivalent to the width of the browser window).

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

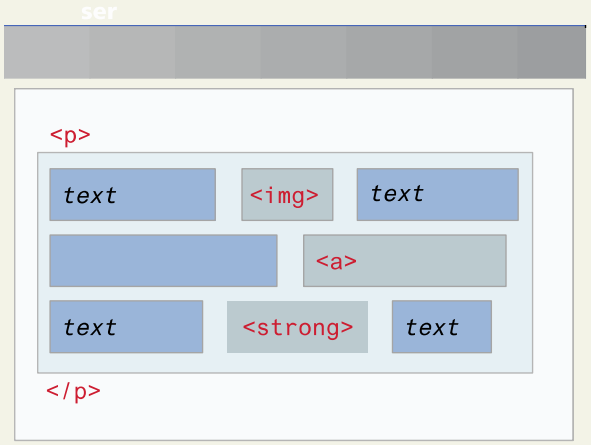
```
<p>
This photo <img src="photo-con.png" alt="..." /> of Conservatory Pond in
<a href="http://www.centralpark.com/">Central Park</a> New York City
was taken on October 22, 2015 with a <strong>Canon EOS 30D</strong>
camera.
</p>
```



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.

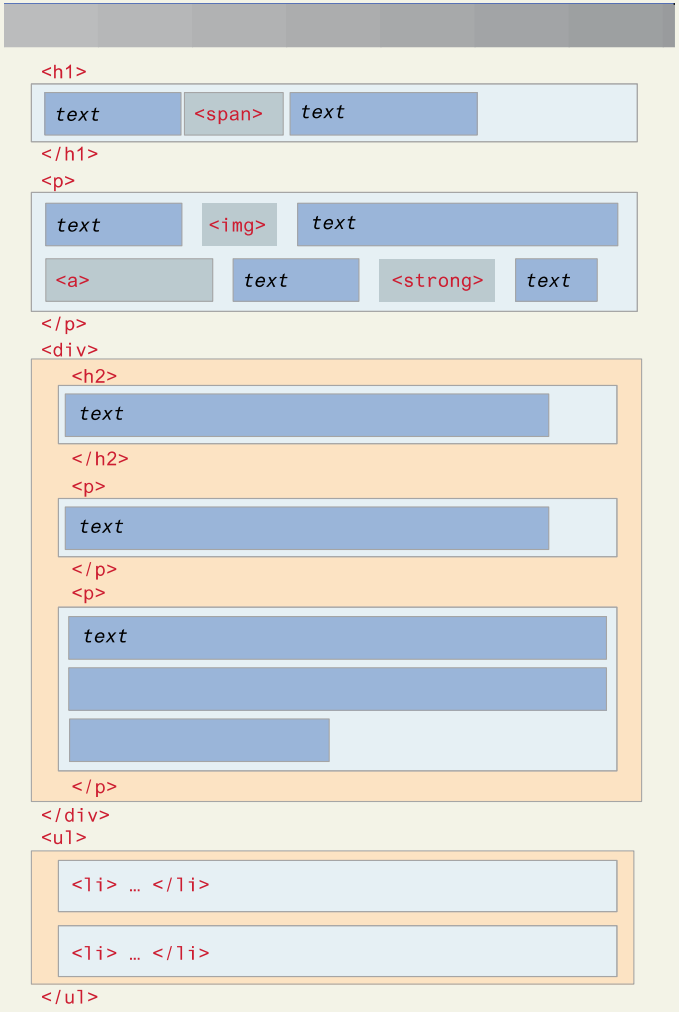


If the browser window resizes, then inline content will be "reflowed" based on the new width.

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

Normal Flow

Block and Inline Elements



( ).

Chapter 7

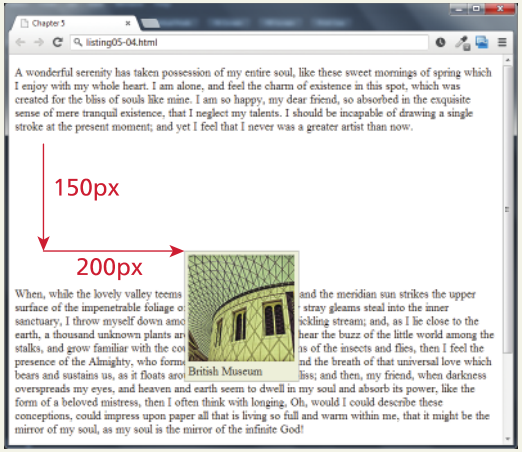
1. Normal Flow
2. Positioning Elements
3. Floating Elements
4. Constructing Multicolumn Layouts
5. Approaches to CSS Layouts
6. Responsive Design
7. Filters, Transitions, and Animations
8. CSS Frameworks and Preprocessors

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.

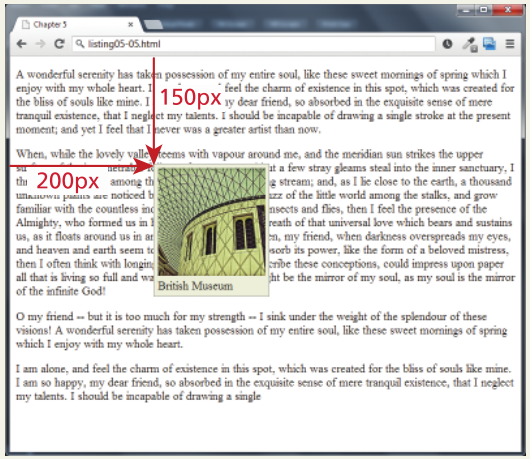
Positioning Elements

Relative Positioning



Positioning Elements

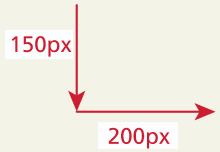
Absolute Positioning



Positioning Elements

Absolute Positioning is relative to nearest positioned ancestor


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

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



Positioning Elements

Z-Index

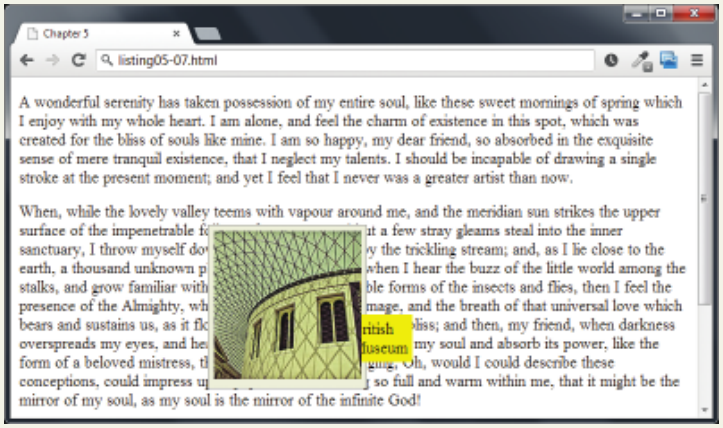
```
figure {
   position: absolute;
   top: 150px;
   left: 200px;
}
figcaption {
   position: absolute;
   top: 90px;
   left: 140px;
}
```

```
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.

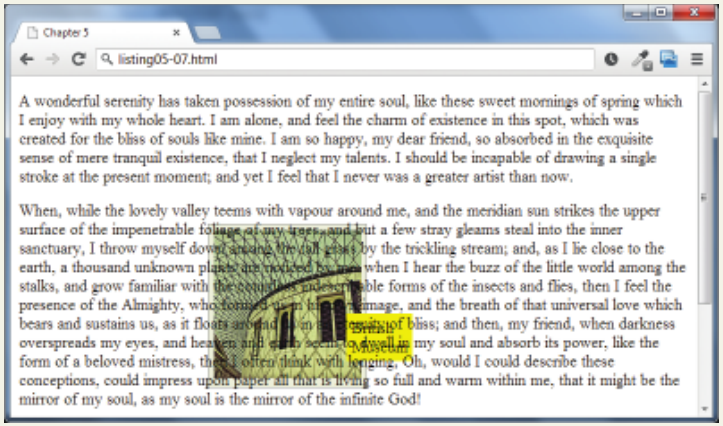
Positioning Elements

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.

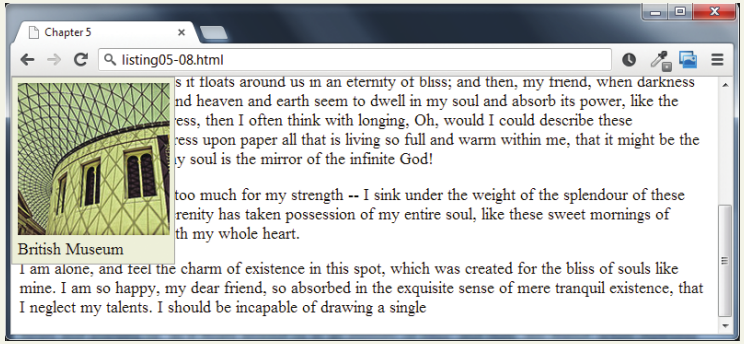



```
figure {
    …
    z-index: -1;
}
figcaption {
    …
    z-index: 1;
}
```

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



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

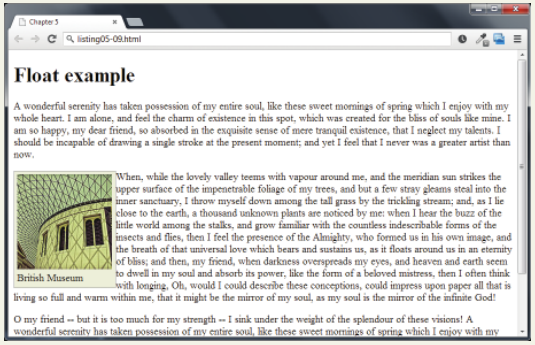
- An element can be floated to the left or floated to the right .
- it is moved all the way to the far left or far right of its containing block and the rest of the content is "reflowed" around the floated element

Floating Elements

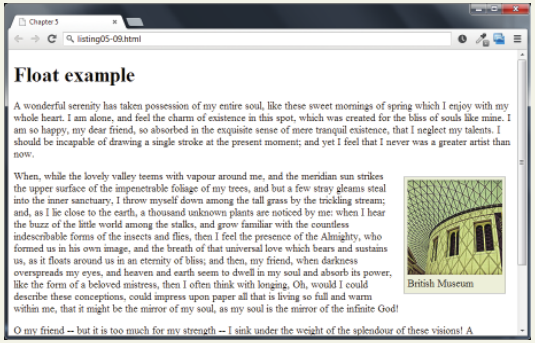
```
<h1>Float example</h1>
<p>A wonderful serenity has taken ...</p>
<figure>
   <img src="images/828.jpg" alt="..." />
   <figcaption>British Museum</figcaption>
</figure>
<p>When, while the lovely valley …</p>
```

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

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



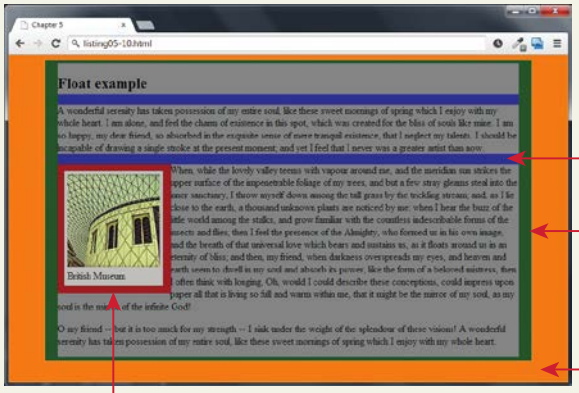

```
figure {
   …
   width: 150px;
   float: left;
}
```




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

Floating Elements

Floating within a Container

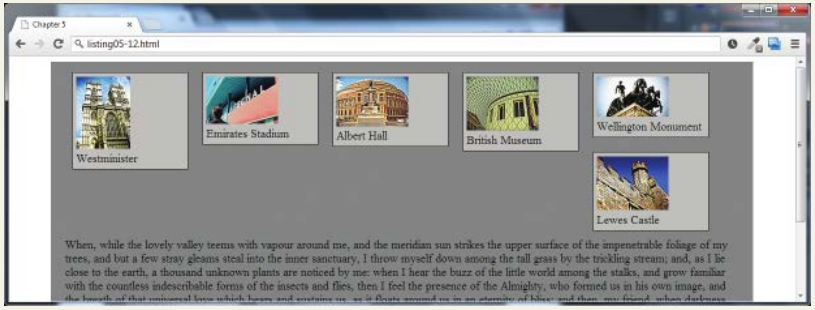



```
  { margin: 16px 0; }
figure {
   border: 1pt solid #262626;
   background-color: #c1c1c1;
   padding: 5px;
   width: 150px;
   float: left;
   margin: 10px;
}
```

Floating Elements

Floating Multiple Items Side by Side

```
<article>
  <figure>
     <img src="images/tiny/275.jpg" alt="..." />
     <figcaption>Westminister</figcaption>
  </figure>
  <figure>
     <img src="images/tiny/700.jpg" alt="..." />
     <figcaption>Emirates Stadium</figcaption>
  </figure>
  <figure>
     <img src="images/tiny/537.jpg" alt="..." />
     <figcaption>Albert Hall</figcaption>
  </figure>
  <figure>
     <img src="images/tiny/828.jpg" alt="..." />
     <figcaption>British Museum</figcaption>
  </figure>
  <figure>
     <img src="images/tiny/464.jpg" alt="..." />
     <figcaption>Wellington Monument</figcaption>
  </figure>
  <figure>
     <img src="images/tiny/224.jpg" alt="..." />
     <figcaption>Lewes Castle</figcaption>
  </figure>
  <p>When, while the lovely valley teems ..
</article>
```



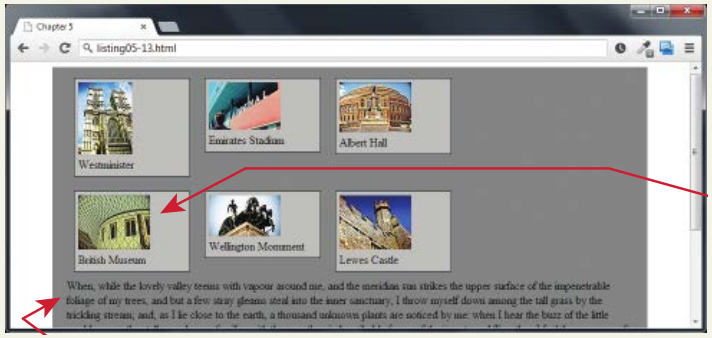

```
figure {
    ...
    width: 150px;
    float: left;
}
```

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




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

Floating Elements

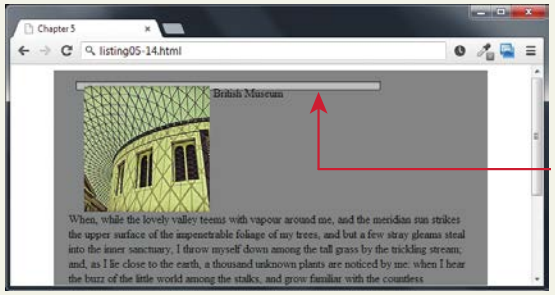
Clear property

- **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




```
ear: left; }
```

Floating Elements

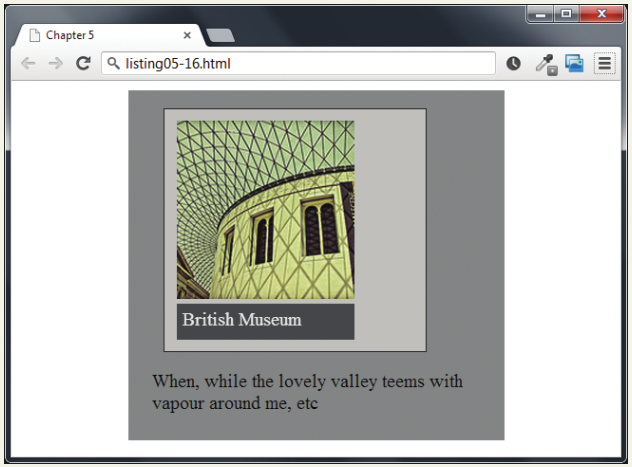
Overlaying and Hiding Element

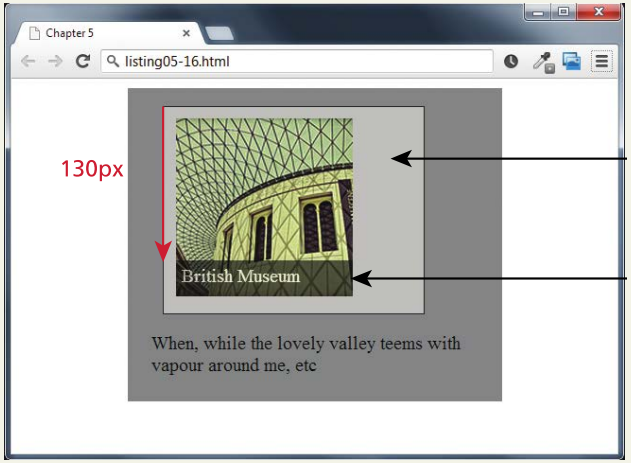
One of the more common design tasks with CSS is to place two elements on top of each other, or to selectively hide and display elements

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

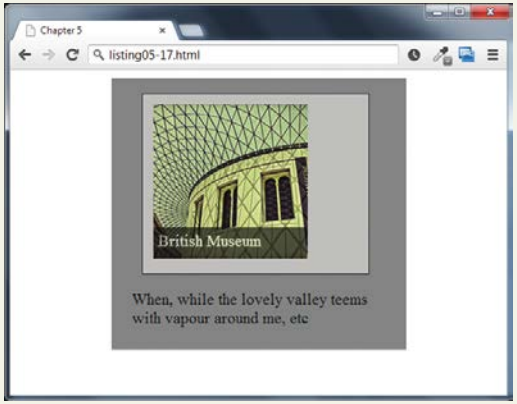




Floating Elements

Using display

NEW



This makes it clear in the markup that the element is not visible.

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

Floating Elements

Comparing visibility with display


```
    display: auto;
}
```

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

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

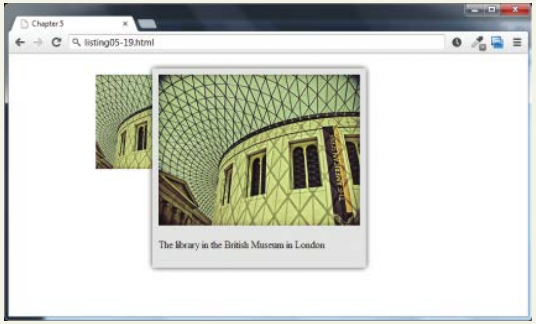
Floating Elements

Using Hover with display

```
<figure class="thumbnail">
  <img src="images/828.jpg" alt="..." />
  <figcaption class="popup">
     <img src="images/828-bigger.jpg" alt="..." />
     <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.

```
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;
}
```



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

```
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

Constructing Multicolumn Layout

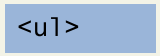
Using Floats to Create Columns

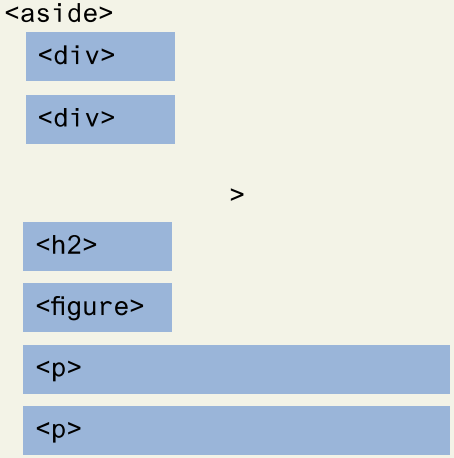
Constructing Multicolumn Layout

Using Floats to Create Columns

Constructing Multicolumn Layout

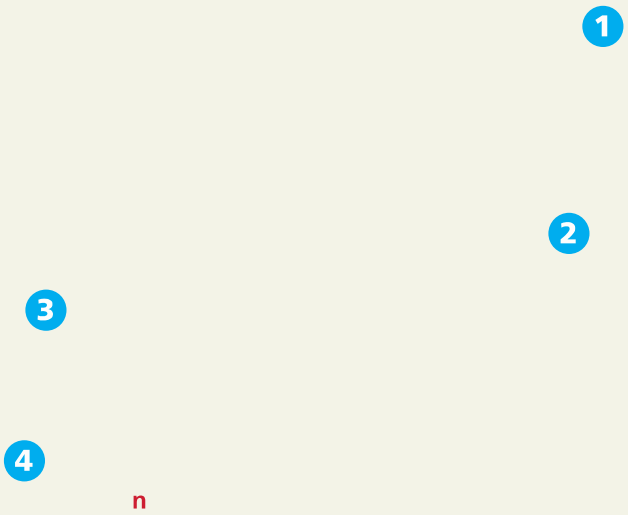
3 column example





Constructing Multicolumn Layout

3 column example with nested floats



Constructing Multicolumn Layout

Using Positioning to Create Columns

1. `position: relative`

2.

3.
```
position:
 absolute;
right: 0;
top: 0;
```

4.

s

Constructing Multicolumn Layout

Problems with Absolute positioning

```
osition:
absolute
```

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

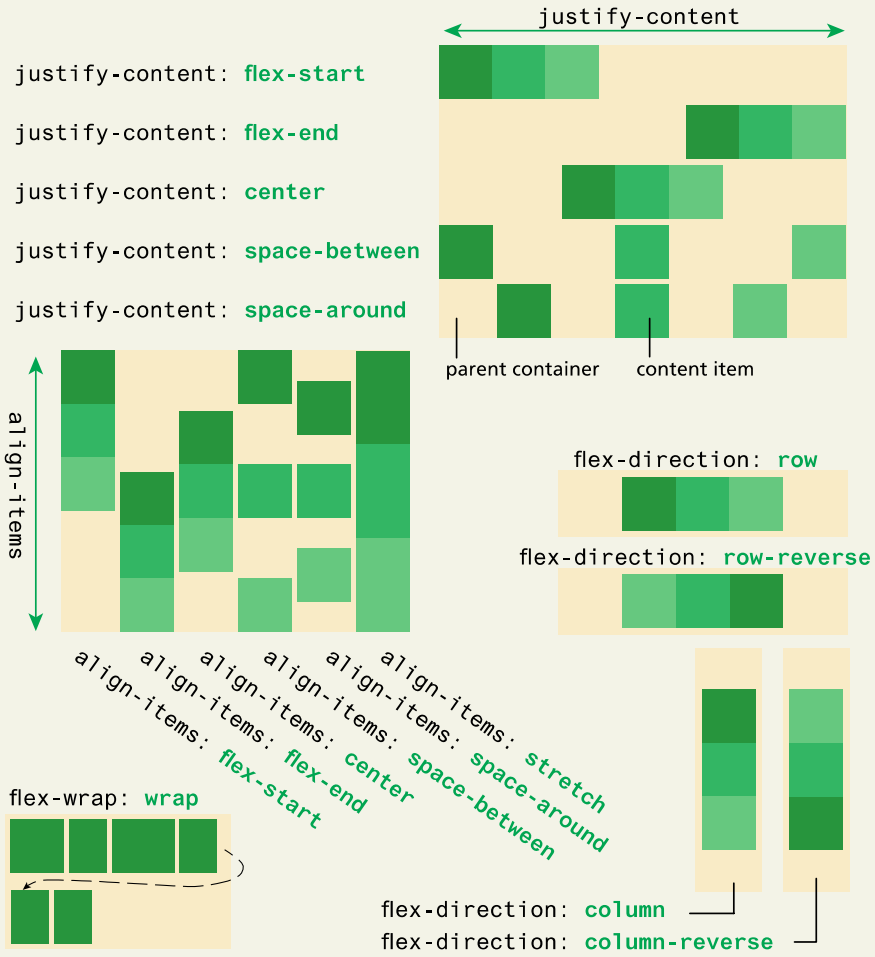
Using Flexbox to Create Columns


```
            auris porta arcu id...</p>
        <p>Phasellus vel felis purus...</p>
    </div>
</div>
```

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

Constructing Multicolumn Layout

The flexbox parent (container) properties



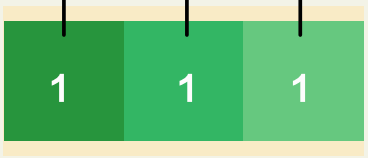
Constructing Multicolumn Layout

The flexbox child (item) properties

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

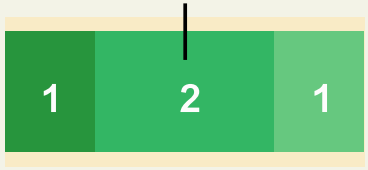
```
flex: 1 1 auto
```

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
```

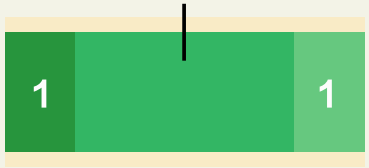


Defines the growth factor of an element relative to the other items.

width=*n*

width=*n* × 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.

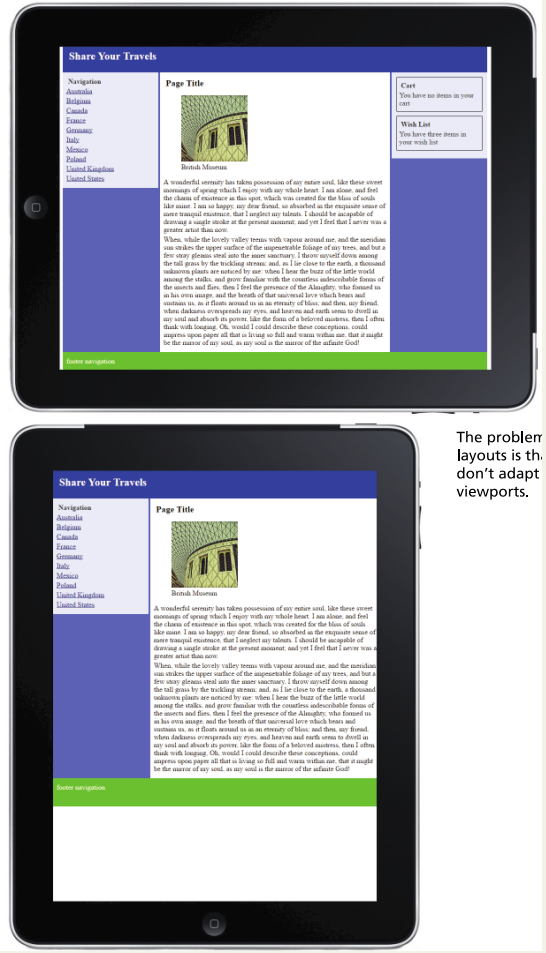
Approaches to CSS Layout

Fixed Layout

ht

Approaches to CSS Layout

Problem with Fixed Layout



The problem
layouts is th
don't adapt
viewports.

maller

Approaches to CSS Layout

Liquid Layout

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

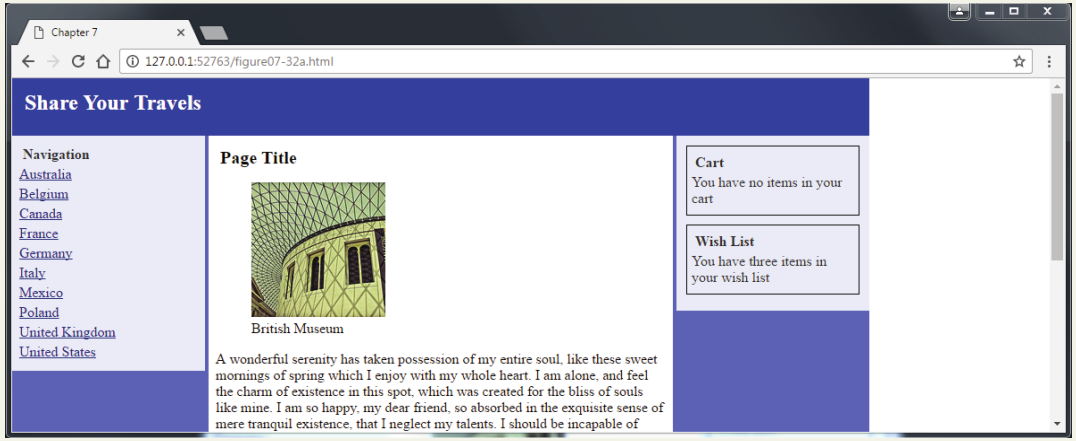
advantage of a liquid layout is that it adapts to different browser sizes

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

Approaches to CSS Layout

Liquid Layout

nts
can get too spread out
as the browser expands.

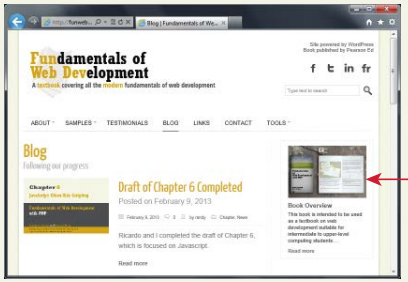


Chapter 7

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

Responsive Design

Responsive Layouts



two-column design
changes to one column.

Responsive Design

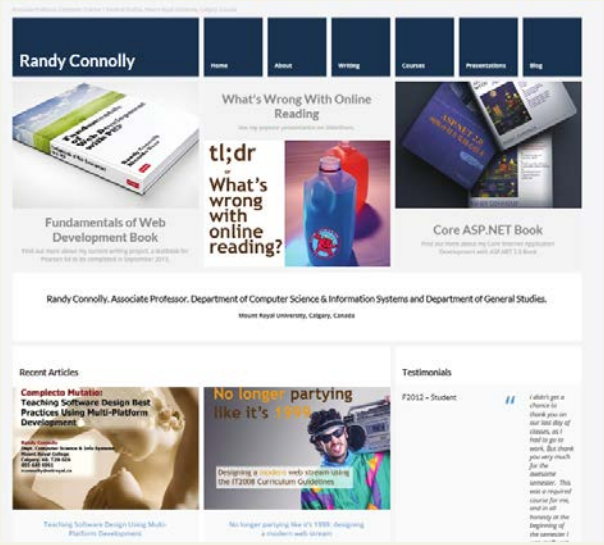
4 elements

1. Liquid layouts
2. Setting viewports via the <meta> tag
3. Customizing the CSS for different viewports using media queries
4. Scaling images to the viewport size

Responsive Design

Setting Viewports

Mobile browser renders web page on its viewport



960px

Mobile browser viewport

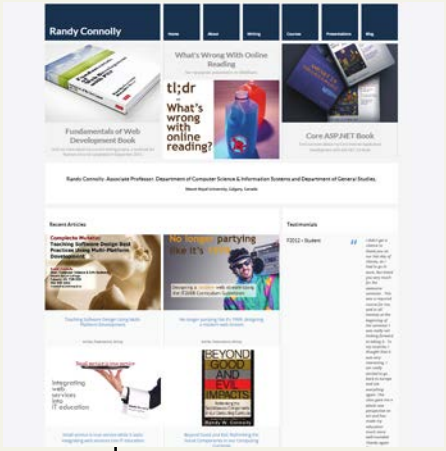
It then scales the viewport to fit within its actual physical screen

320px

Mobile browser screen

Responsive Design

Setting Viewports




```
<meta name="viewport" content="width=device-width" />
```

1. Mobile browser renders web page on its viewport and because of the <meta> setting, makes the viewport the same size as the pixel size of screen.

2. It then displays it on its physical screen with no scaling.

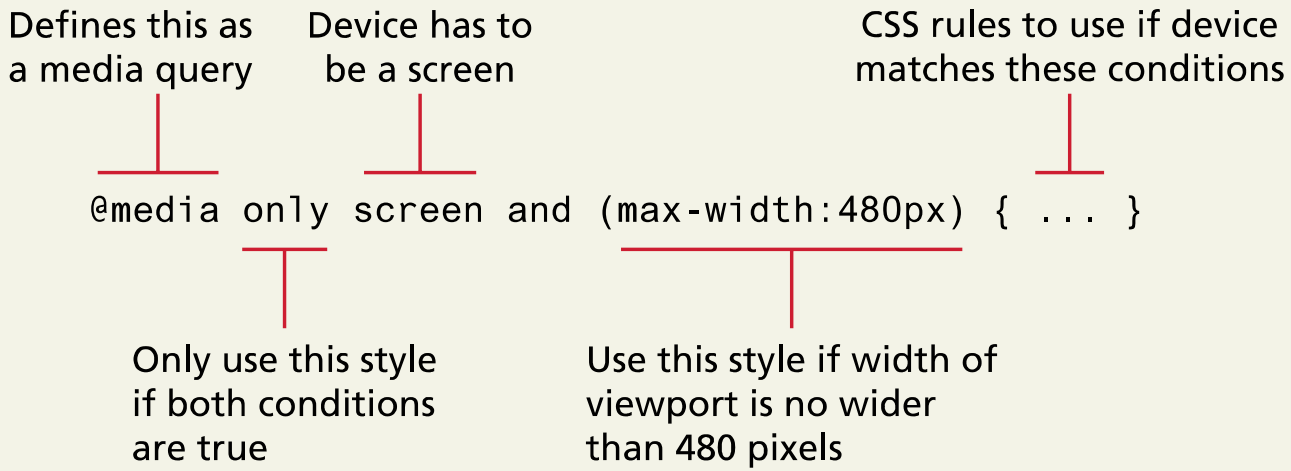
320px

320px

Responsive Design

Media Queries

A media query is a way to apply style rules based on the medium that is displaying the file



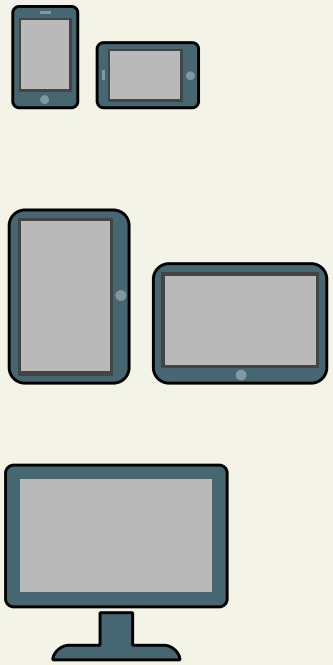
Responsive Design

Media Queries

- **width** Width of the viewport
- **height** Height of the viewport
- **device-width** Width of the device
- **device-height** Height of the device
- **orientation** Whether the device is portrait or landscape
- **color** The number of bits per color

Responsive Design

Media Queries



styles.css

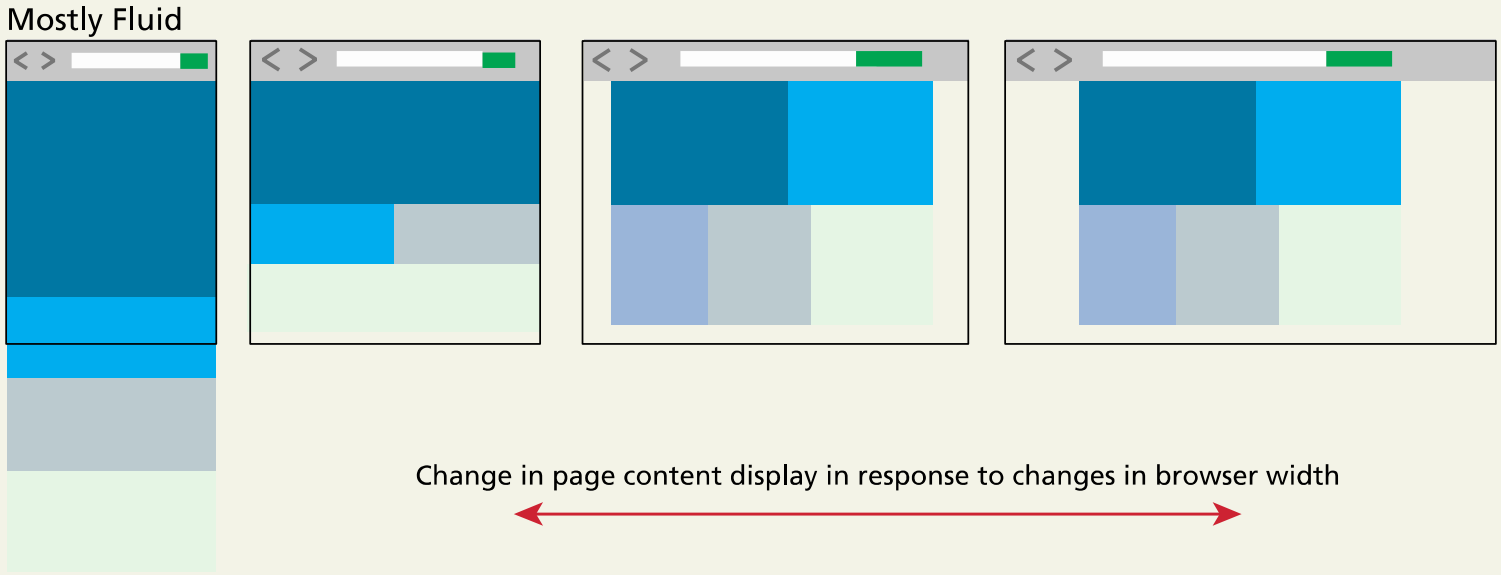
```
/* rules for phones */
@media only screen and (max-width:480px)
{
  #slider-image { max-width: 100%; }
  #flash-ad { display: none; }
    ...
}


/* CSS rules for tablets */
@media only screen and (min-width: 481px)
    and (max-width: 768px)
{
    ...
}



/* CSS rules for desktops */
@media only screen and (min-width: 769px)
{
    ...
}
```

Instead of having all the rules in a single file, we can put them in separate files and add media queries to <link> elements.

```
<link rel="stylesheet" href="mobile.css"  media="screen and (max-width:480px)" />
<link rel="stylesheet" href="tablet.css"  media="screen and (min-width:481px)
    and (max-width:768px)" />
<link rel="stylesheet" href="desktop.css" media="screen and (min-width:769px)" />
```

Responsive Design

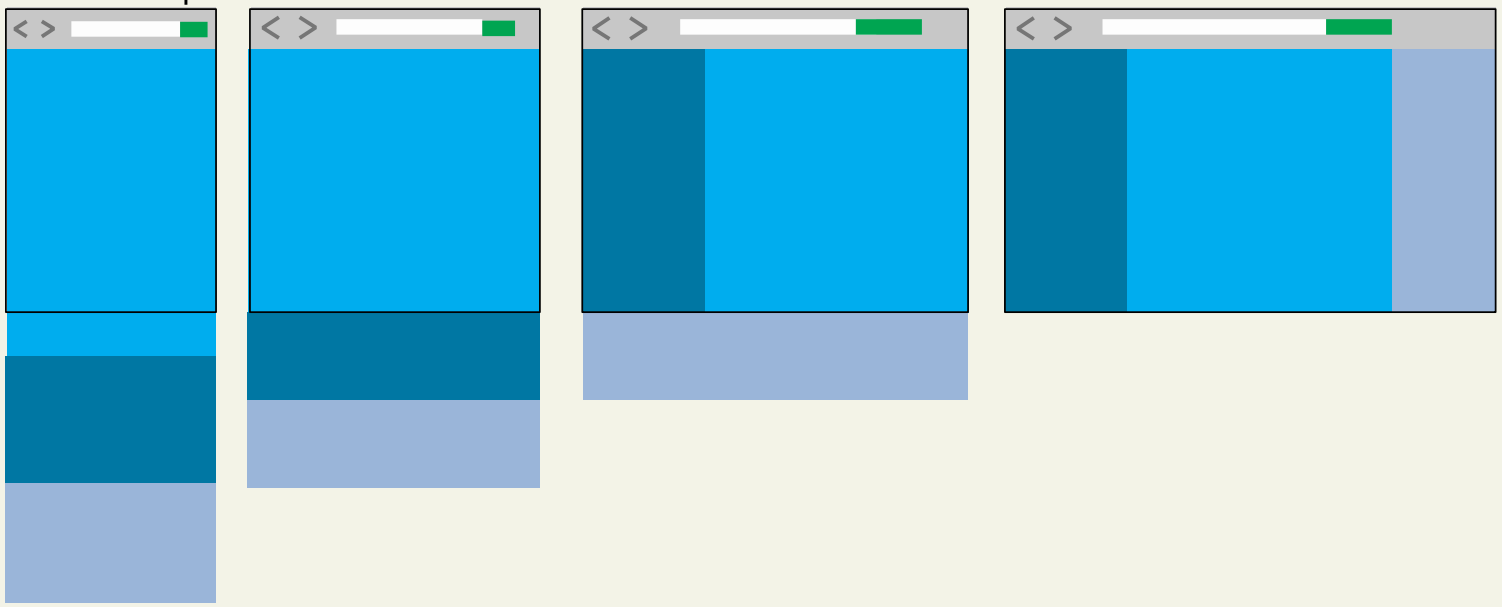
Responsive Design Patterns



Responsive Design

Responsive Design Patterns

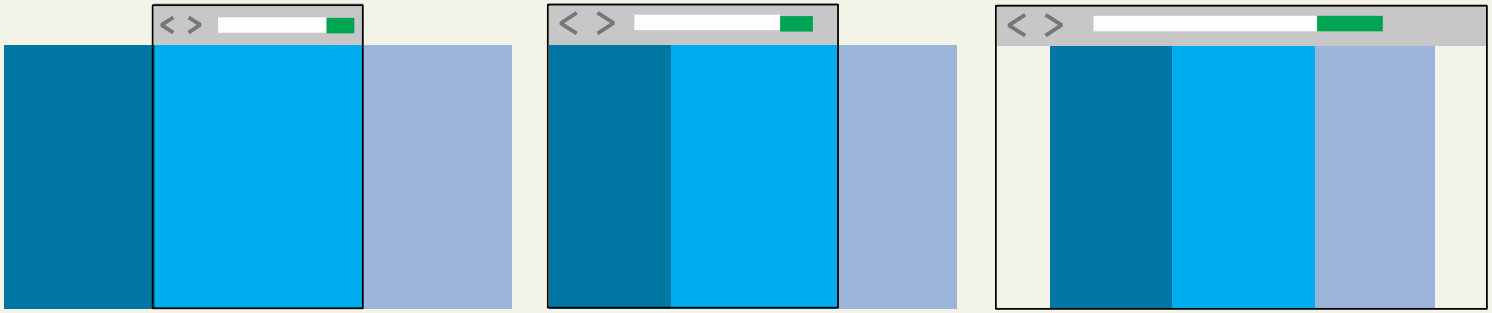
Column Drop



Responsive Design

Responsive Design Patterns

Off Canvas



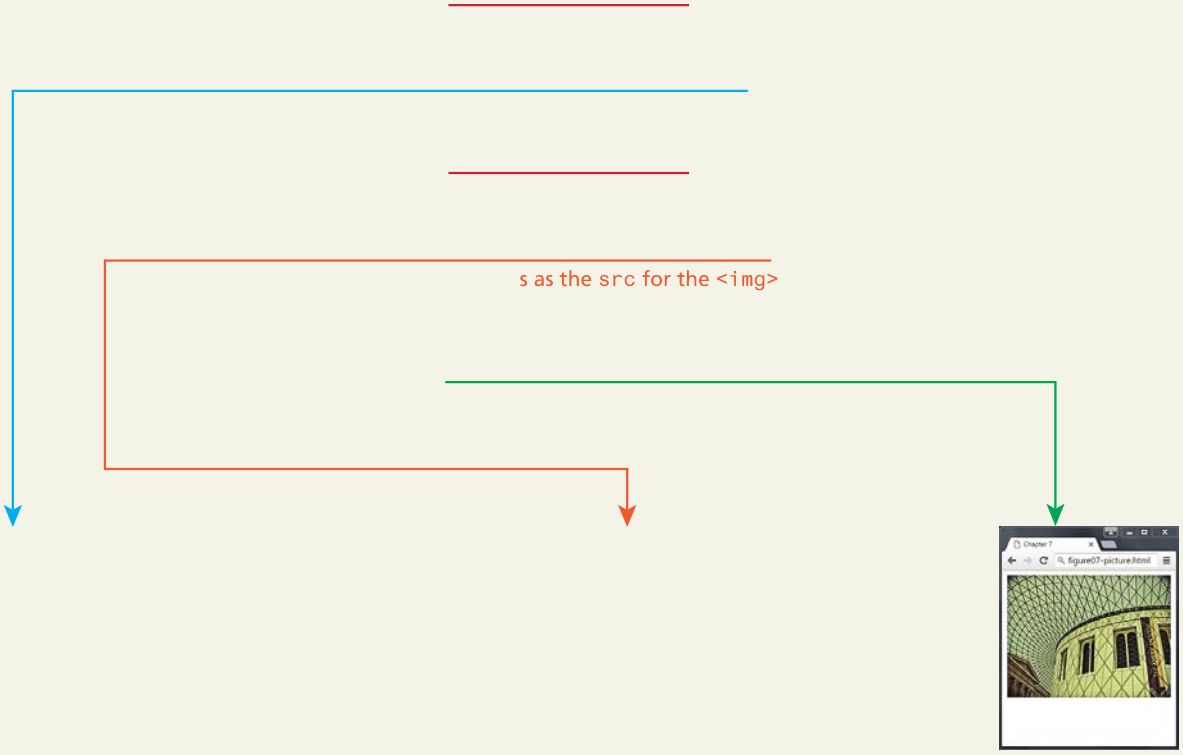
Responsive Design

Scaling Images

- img {
  max-width: 100%;
  }
- <picture>

Responsive Design

Scaling Images



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

Filters, Transitions, and Animations

Filters

```
                    deg)
saturate(200%)
```

Filters, Transitions, and Animations

Filters

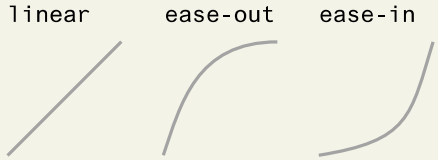
```
#someImage {
        filter: grayscale(100%);
        /* At time of writing, Chrome and Opera
needs prefix */
        -webkit-filter: grayscale(100%);
}

#anotherImage {
        /* multiple filters are space separated */
        filter: blur(5px) hue-rotate(60deg) saturate(2);
        -webkit-filter: blur(5px) hue-rotate(60deg)
saturate(2);
}
```

Filters, Transitions, and Animations

Transitions

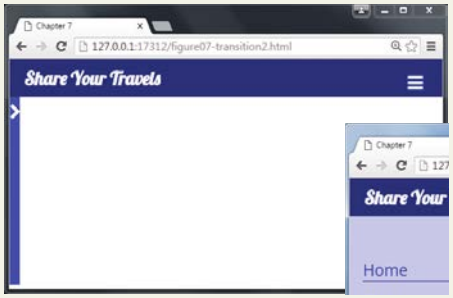
Button as it appears during transition between two states

ears when hovered over



Filters, Transitions, and Animations

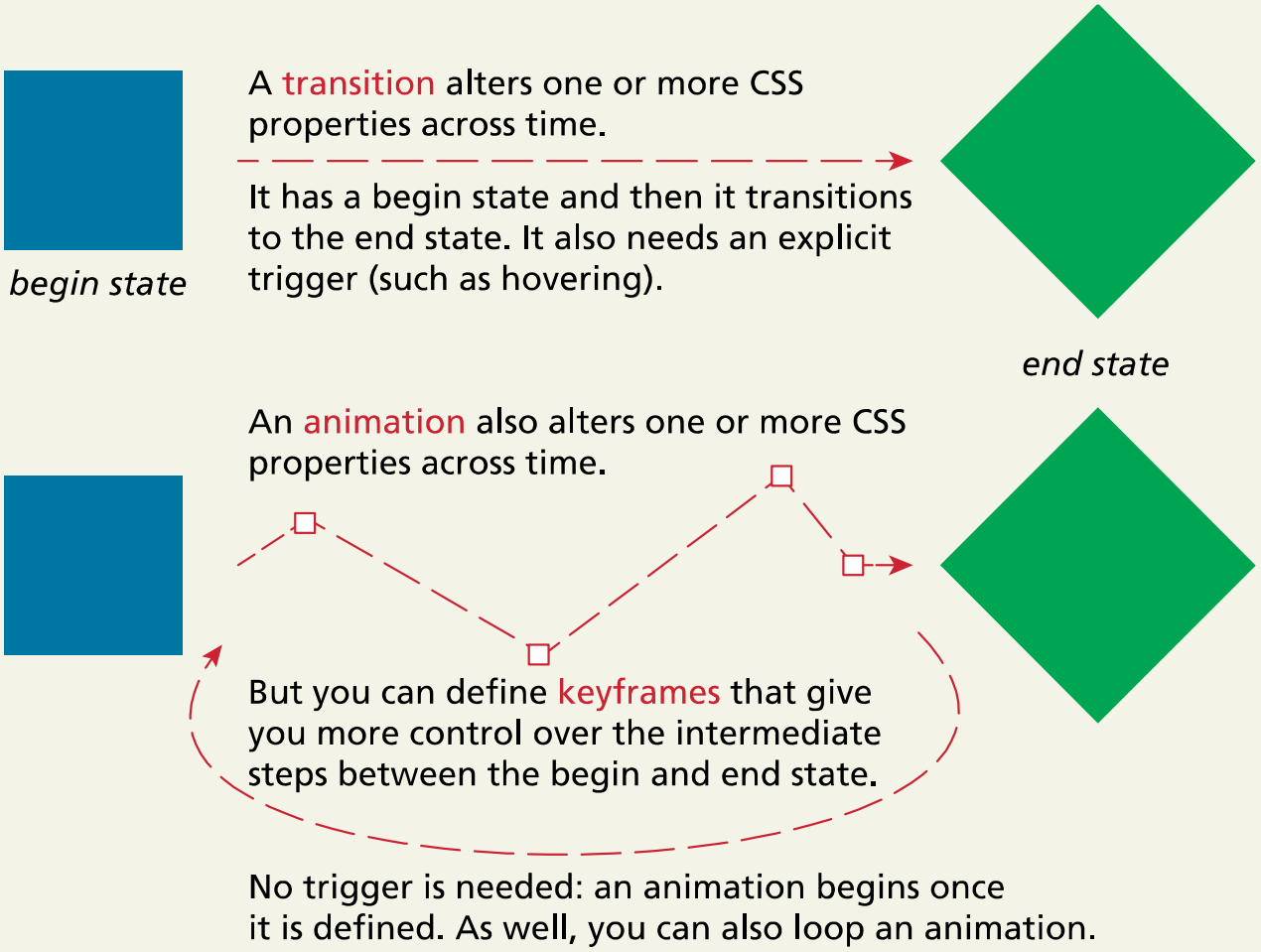
Transitions



hen the mouse is no longer hovering over the menu. This creates illusion of menu sliding back out of sight.

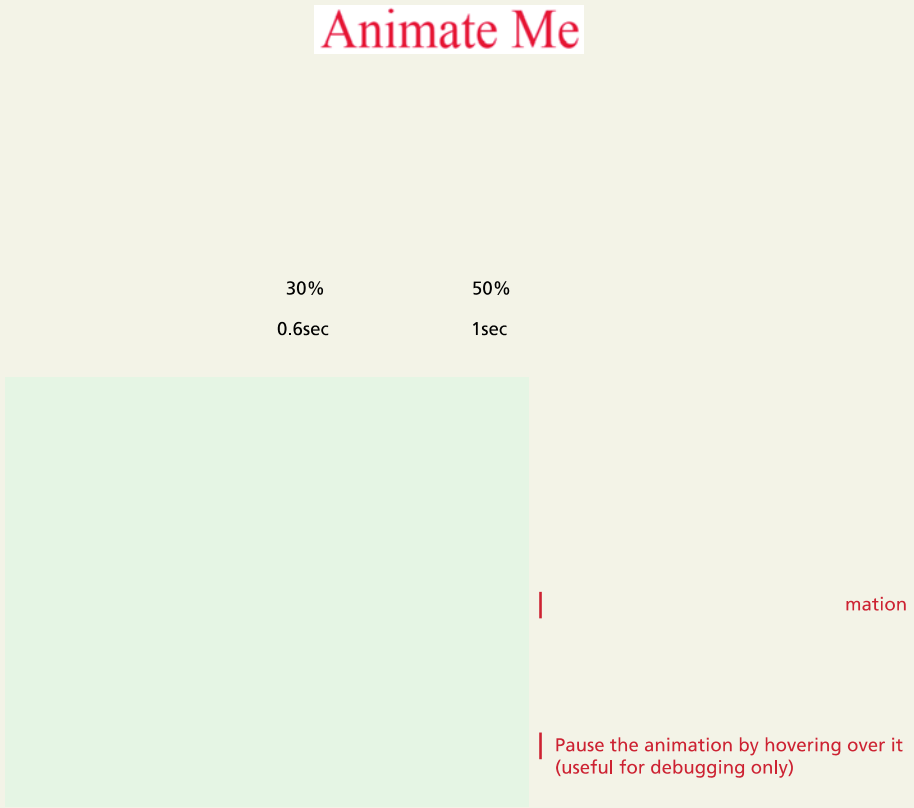
Filters, Transitions, and Animations

Transitions vs animations



Filters, Transitions, and Animations

Animations

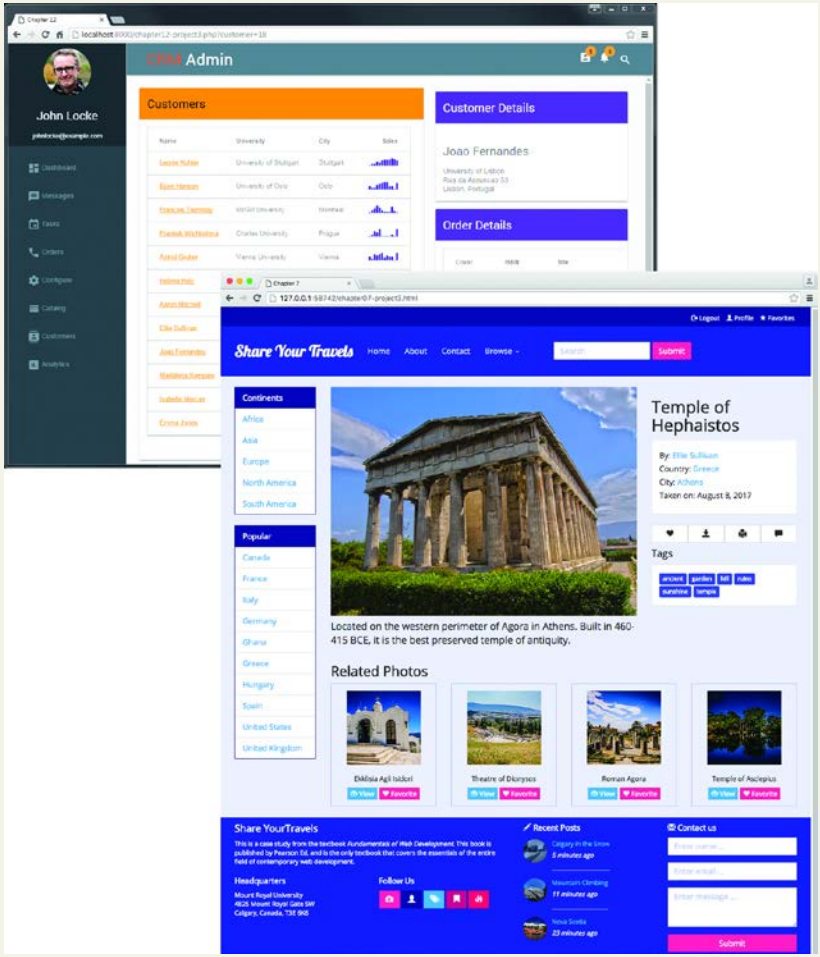


Chapter 7

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

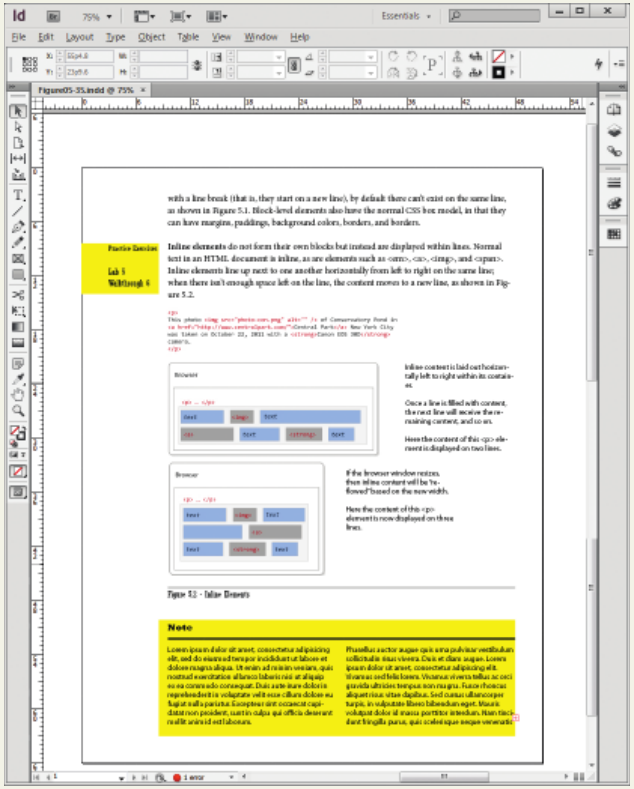
CSS Frameworks and Preprocessors

CSS Frameworks



CSS Frameworks and Preprocessors

Grid in print design



s on the page do not look random, but planned and harmonious.

CSS Frameworks and Preprocessors

Using Bootstrap

```
<head>
  <link href="bootstrap.css" rel="stylesheet">
</head>
<body>
  <div class="container">
    <div class="row">
      <div class="col-md-2">
        left column
      </div>
      <div class="col-md-7">
        main content
      </div>
      <div class="col-md-3">
        right column
      </div>
    </div>
  </div>
</body>
```

CSS Frameworks and Preprocessors

CSS **Preprocessors**

```
$colorSchemeA: #796d6d;
$colorSchemeB: #9c9c9c;
$paddingCommon: 0.25em;


footer {
  background-color: $colorSchemeA;
  padding: $paddingCommon * 2;
}


@mixin rectangle($colorBack, $colorBorder) {
  border: solid 1pt $colorBorder;
  margin: 3px;
  background-color: $colorBack;
}

fieldset {
  @include rectangle($colorSchemeB, $colorSchemeA);
}

.box {
  @include rectangle($colorSchemeA, $colorSchemeB);
  padding: $paddingCommon;
}
```

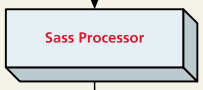
Sass source file, e.g., `source.scss`

This example uses Sass (Syntactically Awesome Stylesheets). Here three variables are defined.

You can reference variables elsewhere. Sass also supports math operators on its variables.

A mixin is like a function and can take parameters. You can use mixins to encapsulate common styling.

A mixin can be referenced/called and passed parameters.



The processor is some type of tool that the developer would run.

```
footer {
  padding: 0.50em;
  background-color: #796d6d;
}

fieldset {
  border: solid 1pt #796d6d;
  margin: 3px;
  background-color: #9c9c9c;
}

.box {
  border: solid 1pt #9c9c9c;
  margin: 3px;
  background-color: #796d6d;
  padding: 0.25em;
}
```

Generated CSS file, e.g., `styles.css`

The output from the processor is a normal CSS file that would then be referenced in the HTML source file.

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 cont.

9 Summary

Summary

Key Terms

- absolute positioning
- animations
- BEM
- block
- block-element-modifier
- block-level elements
- clear property
- containing block
- CSS framework
- CSS media queries
- CSS preprocessors
- elements
- filters
- fixed layout
- fixed positioning
- flexbox layout
- float property
- fluid layout
- image placeholder services
- inline elements
- keyframes
- liquid layout
- modifiers
- nonreplaced inline elements
- normal flow
- positioning context
- progressive enhancement
- relative positioning
- replaced inline elements
- responsive design
- style guides
- transforms
- transitions
- viewport
- z-index

Questions