Fundamentals of Web Development

Third Edition by Randy Connolly and Ricardo Hoar

Chapter 7

CSS 2: Layout

In this chapter you will learn . . .

- Approaches to CSS layout using CSS flexbox and grid models
- What responsive web design is and how to construct responsive designs
- How to use CSS3 filters, transitions, and animations
- What are CSS preprocessors

Older Approaches to CSS Layout

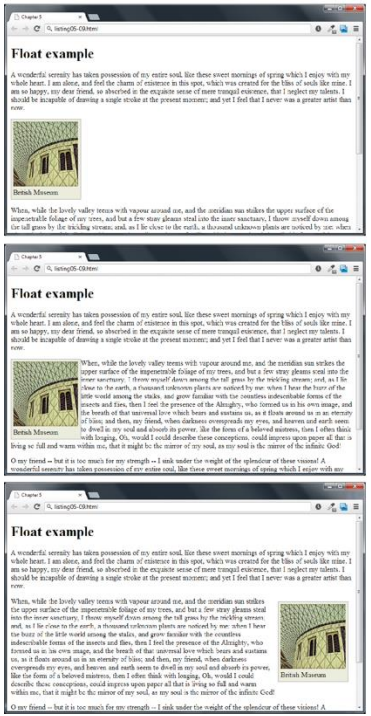
- The display property in CSS provides a mechanism for the developer to change an element to block, inline, or inline-block.
- For the first 20 years of CSS, designers had to “hack” together multi column layouts using **floats** and/or **positioning**. There may be times when you may have to support legacy CSS so it makes sense to learn the basics of floats and positioning.
- Newer approaches (flexbox and grid display modes) make columnar layouts much easier to implement.

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** and content is “reflowed” around the floated element

A floated block-level element should have a width specified




```
<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;
}
```

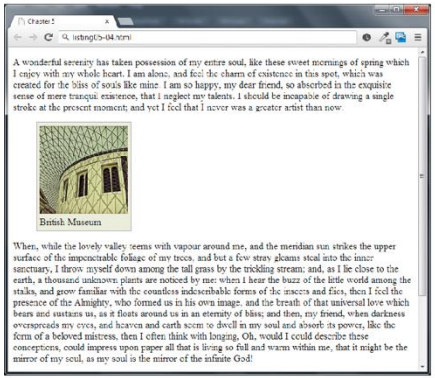
```
figure {
   …
   width: 150px;
   float: right;
   margin: 10px;
}
```

Positioning Elements

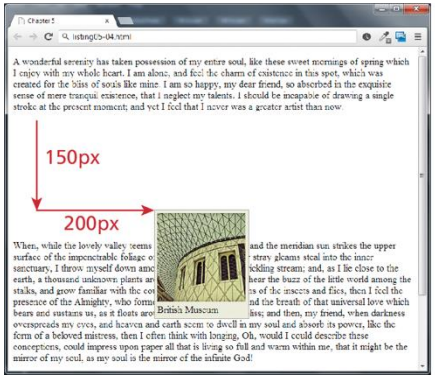
The **position** property is used to specify the type of positioning. The **left**, **right**, **top**, and **bottom** properties are used to indicate the distance the element will move.

- **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**.
- **sticky** The element is positioned in according to the normal flow, and then offset relative to its nearest scrolling ancestor. This is used to allow an item to scroll, and then stay fixed in position once its scroll position is reached.

Relative Positioning

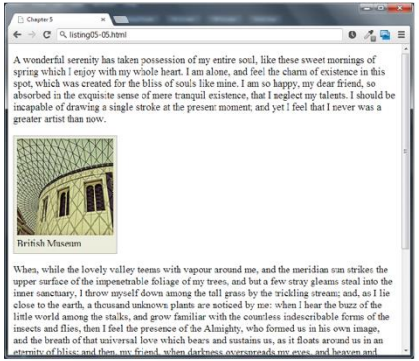



```
<p>A wonderful serenity has taken ... </p>

<figure>
   <img src="images/828.jpg" alt="British Museum" />
   <figcaption>British Museum</figcaption>
</figure>

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

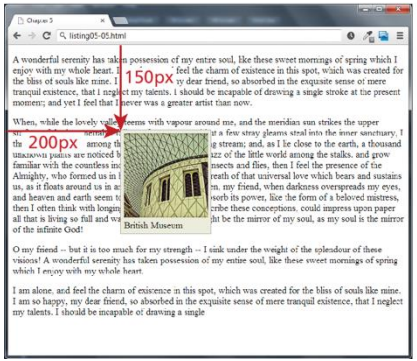



```
figure {
   border: 1pt solid #A8A8A8;
   background-color: #EDEDDD;
   padding: 5px;
   width: 150px;
   position: relative;
   top: 150px;
   left: 200px;
}
```

Absolute positioning




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

<figure>
   <img src="images/828.jpg" alt="British Museum" />
   <figcaption>British Museum</figcaption>
</figure>

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

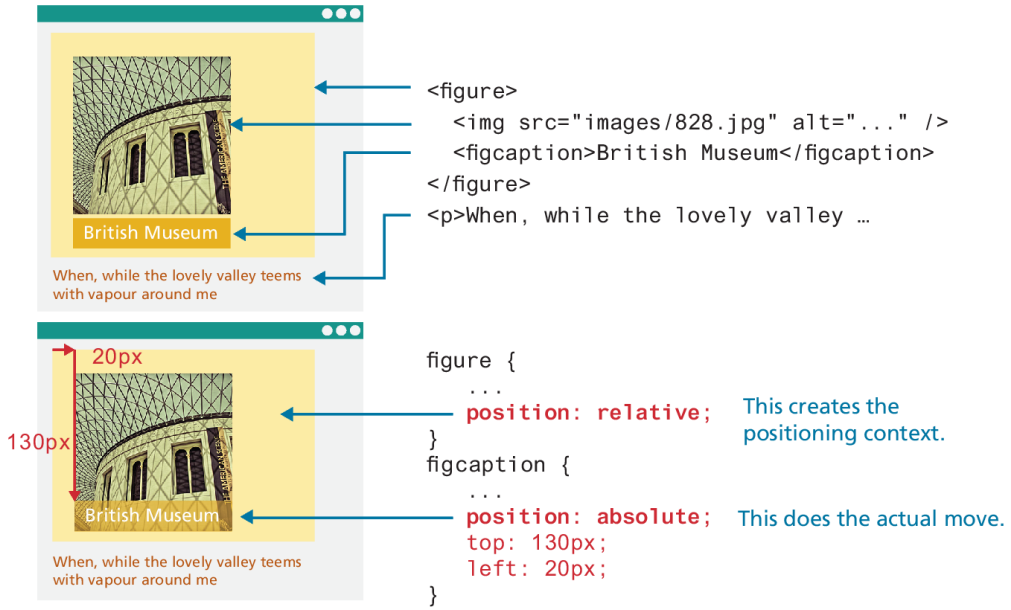



```
figure {
   margin: 0;
   border: 1pt solid #A8A8A8;
   background-color: #EDEDDD;
   padding: 5px;
   width: 150px;
   position: absolute;
   top: 150px;
   left: 200px;
}
```

Overlapping and Hiding Elements

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.

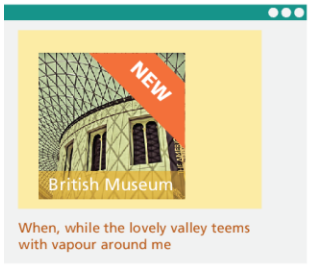


Overlapping and Hiding Elements (ii)

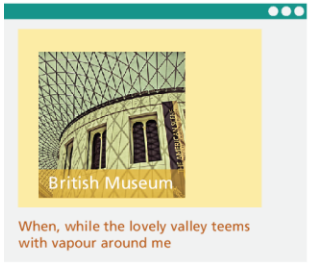
Consider an image that is the same size as the underlying one is placed on top of the other image using absolute positioning.

You can hide this image using the **display** property

```
<figure>
  <img src="828.jpg" alt="..." />
  <figcaption>British Museum</figcaption>
  <img src="new-banner.png" class="overlayed" />
</figure>
```




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




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

This is the preferred way to hide: by adding this additional class to the element.

```
<img src="new-banner.png" class="overlayed hide" />
```

Flexbox Layout


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

```
<div class="media">
   <img class="media-image" src="c.jpg" >
   <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>
```

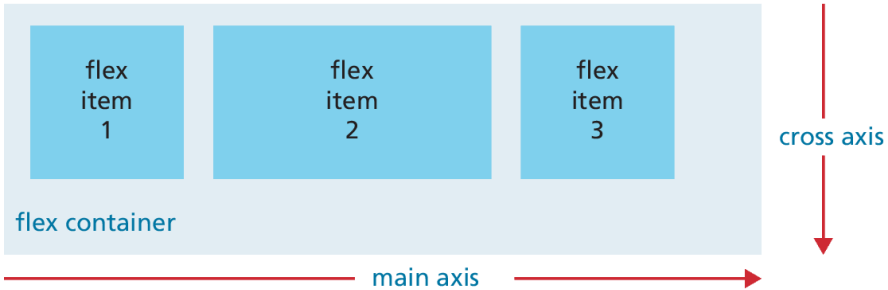
Prior to flexbox, one would create such a layout within a container using floats plus margins. The problem with this approach is that margins needed to be in pixels and had to exactly match image size. If image size changed (or you wanted same kind of style elsewhere), you had to modify the style.


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

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

Flex containers and items

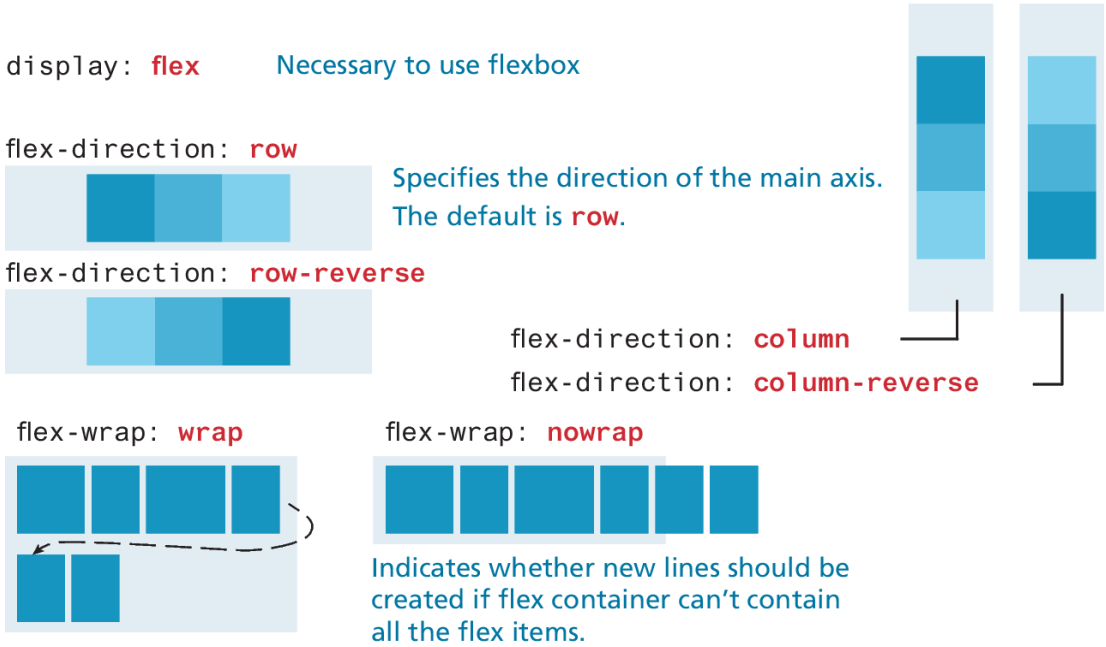
There are two places in which you will be assigning flexbox properties: the **flex container** and the **flex items** within the container.




```
<div class="container">
  <div>flex item 1</div>
  <span>flex item 2</span>
  <div>flex item 3</div>
</div>
```

Both block-level and inline elements will become flex items.

The flexbox container properties



The flexbox container properties (ii)

justify-content: flex-start

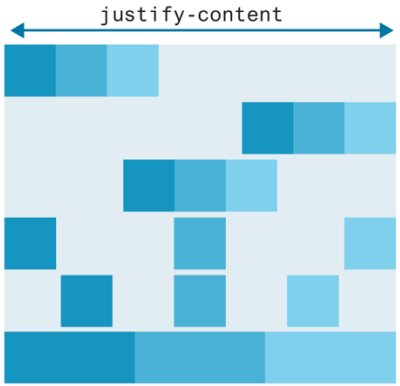
justify-content: flex-end

justify-content: center

justify-content: space-between

justify-content: space-around

justify-content: stretch



Controls flex item alignment along main axis.

Controls flex item alignment along cross axis (i.e., wrapped items).

align-items: flex-start

align-items: flex-end

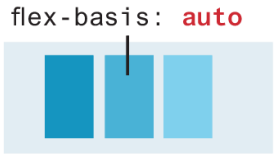
align-items: center

align-items: space-between

align-items: space-around

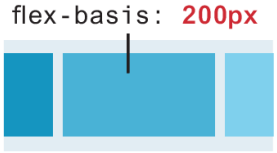
align-items: stretch

The flexbox item (child) properties

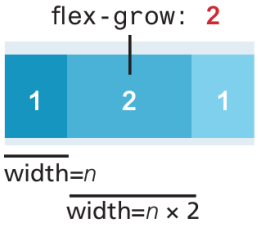


The `flex-basis` property determines the initial size of the flex item before the remaining space is distributed.

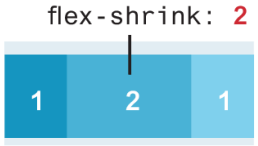
The default `auto` value means that the size is determined by the width and height.



You can specify a width using px, %, or other measurement units.



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



Defines how much an item will shrink when not enough space in container.

Flexbox Cards

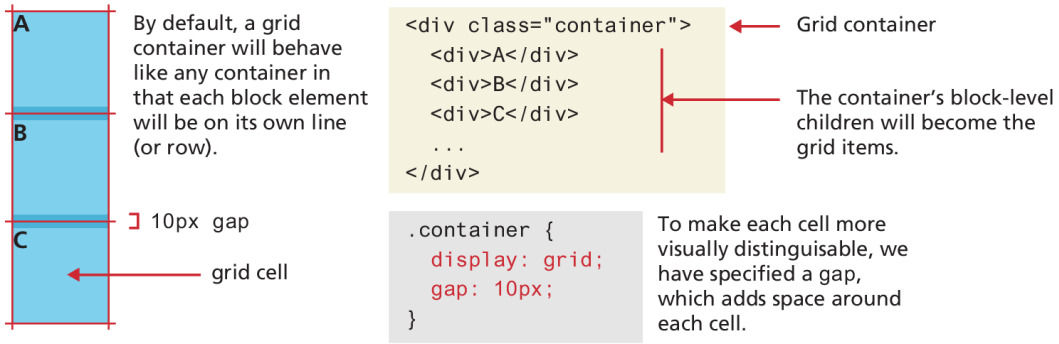

```
<div class="card">
   <div class="content">
      <img src="w.jpg" />
      <h3>Albert Hall</h3>
   </div>
   <footer>
      <a href="#">View</a>
   </footer>
</div>
```


```
.card {
  ...
  display: flex;
  flex-direction: column;
}

.card .content {
   flex: 1 1 auto;
}
```

Grid Layout

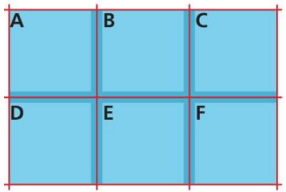
Grid layout is adjustable, powerful, and, compared to floats, positioning, and even flexbox, is relatively easy to learn and use!

- Each blocklevel child in a parent container whose **display** property is set to **grid** will be automatically placed into a grid cell



Specifying Grid Structure

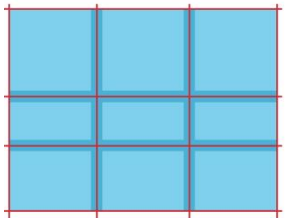
grid-template-columns is used for adding columns by specifying each column's width using the **fr** unit.




```
.container {
  display: grid;
  gap: 10px;
  grid-template-columns: 1fr 1fr 1fr;
}
```

This makes each column equal to an equal fraction of the available space.

You can specify the number of columns per row/line via the `grid-template-column` property.



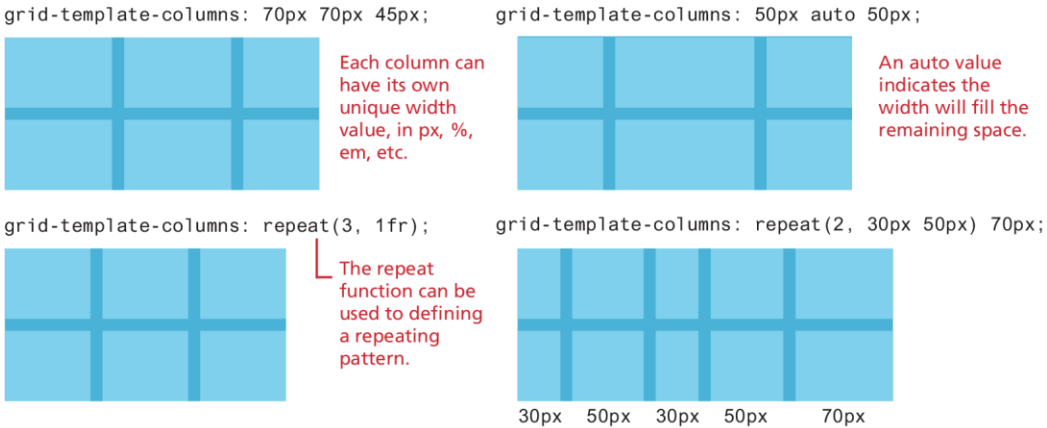
```
.container {
  ...
  height: 300px;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 80px 30px 50px;
}
```

While you often do not need to set a row height, you can make rows taller than their actual content.

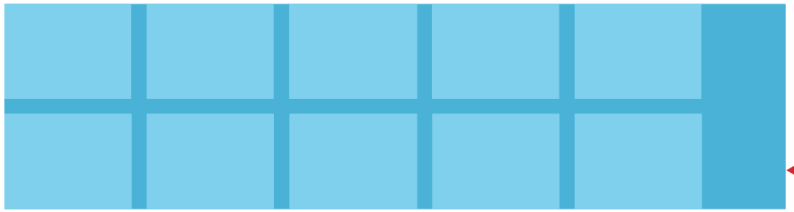
Specifying column widths

Column widths can be specified

The CSS **repeat()** function provides a way to specify repeating patterns of columns.



Specifying column widths (ii)

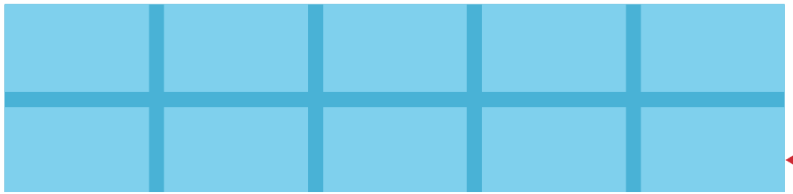


width: 560px;

```
grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
```

Fills the row with as many columns as can fit into the container space.

The min column size is 100px and the max is whatever size is necessary to allow each column to be equal sized.



width: 560px;

```
grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
```

Fits the columns into space by expanding the column width into the available container space.

Contrasting grid approaches

```
<!-- Bootstrap 4 Approach -->
<div class="container">
  <div class="row">
    <div class="col"><img src=1.gif /></div>
    <div class="col"><img src=2.gif /></div>
    <div class="col"><img src=3.gif /></div>
  </div>
  <div class="row">
    <div class="col"><img src=4.gif /></div>
    <div class="col"><img src=5.gif /></div>
    <div class="col"><img src=6.gif /></div>
  </div>
</div>
```

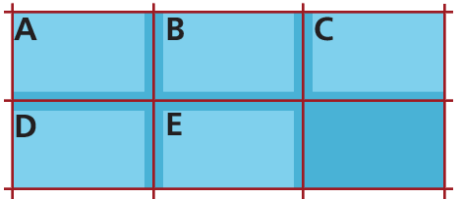
```
<!-- CSS Grid Approach -->
<div class="container">
  <img src=1.gif />
  <img src=2.gif />
  <img src=3.gif />
  <img src=4.gif />
  <img src=5.gif />
  <img src=6.gif />
</div>

<!-- CSS for grid approach -->
.container {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(100px, 1fr);
}
.container img { display: block; }
```

LISTING 7.1 Comparing Bootstrap grid with CSS Grid

Explicit Grid Placement Example 1

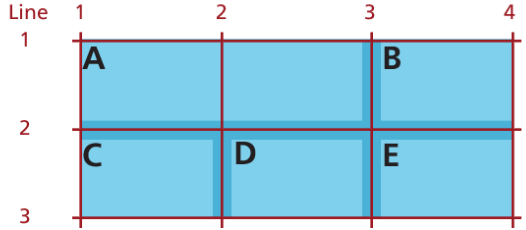
```
<div class="container">
  <div class="a">A</div>
  <div class="b">B</div>
  <div class="c">C</div>
  <div class="d">D</div>
  <div class="e">E</div>
</div>
```

```
.container {
  display: grid;
  gap: 10px;
  grid-template-columns: repeat(3,1fr);
  grid-template-rows: repeat(2,200px);
}
```



With implicit layout, grid items are placed automatically.

Example 1




```
.a {
  grid-column-start: 1;
  grid-column-end: 3;
}
```

The start and end numbers refer to the line number not the column number.

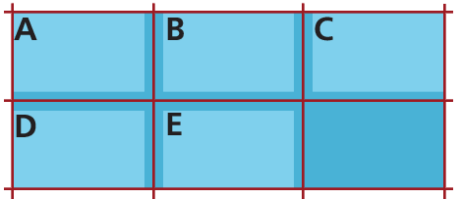
The same effect also possible using either of the following:

```
grid-column: 1 / 3;
grid-column: 1 / span 2;
```

Explicit Grid Placement Example 2

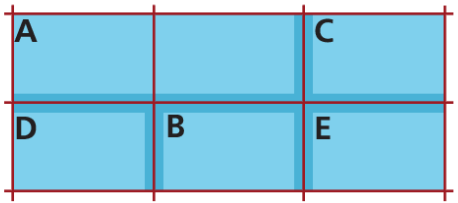
```
<div class="container">
  <div class="a">A</div>
  <div class="b">B</div>
  <div class="c">C</div>
  <div class="d">D</div>
  <div class="e">E</div>
</div>
```

```
.container {
  display: grid;
  gap: 10px;
  grid-template-columns: repeat(3,1fr);
  grid-template-rows: repeat(2,200px);
}
```



With implicit layout, grid items are placed automatically.

2

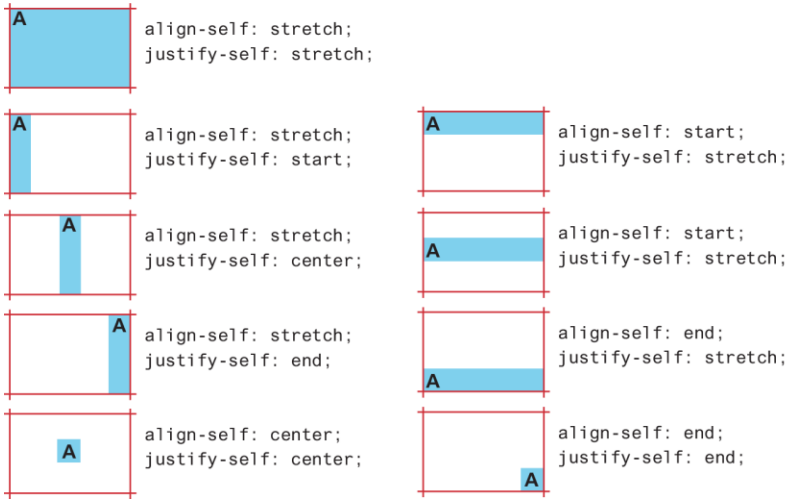



```
.b {
  grid-row: 2;
  grid-column: 2;
}
```

Grid cells can be placed into any row and column.

Cell properties

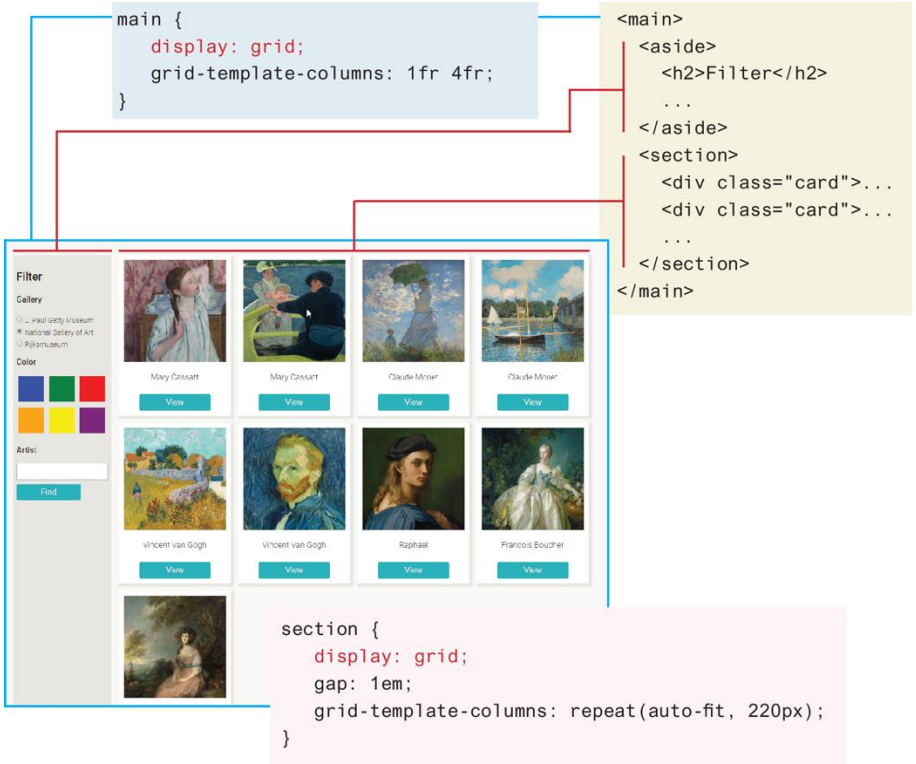
- **align-self and justify-self** control the cell content's horizontal and vertical alignment within its grid container.



- You can similarily control cell alignment within a grid container using **align-items** and **justify-items**

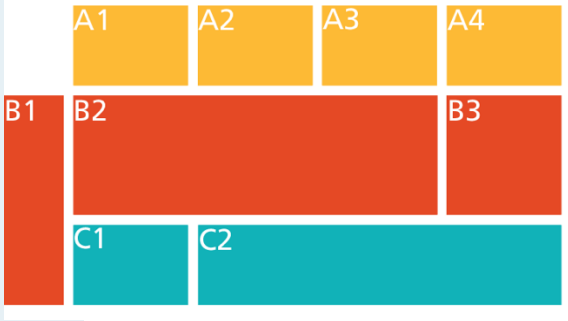
Nested Grid

- **align-self and justify-self** control the cell content's horizontal and vertical alignment within its grid container.



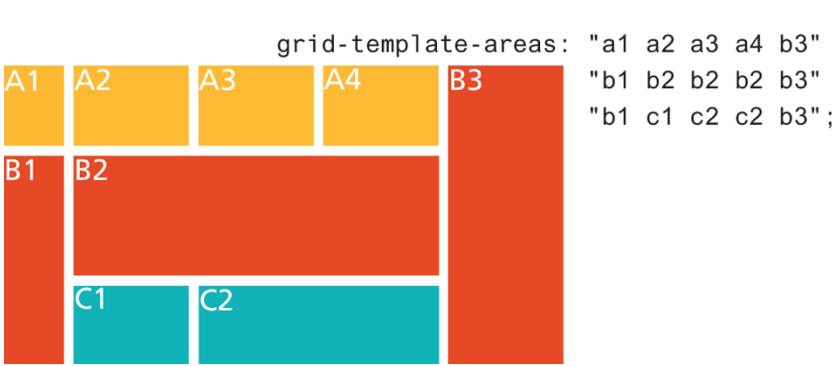
Grid Areas

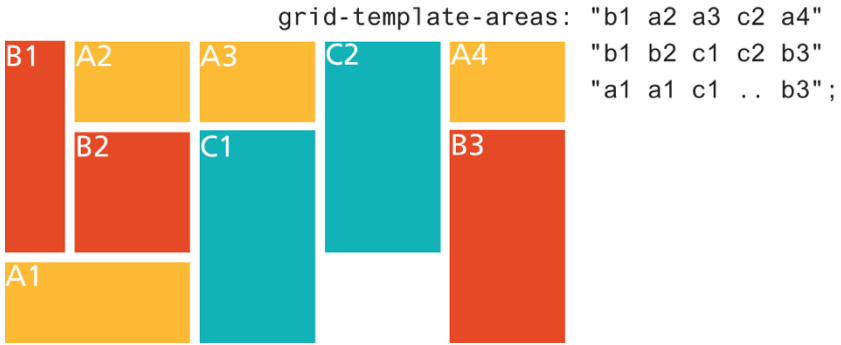
```
<style>
.container {
  grid-gap: 10px;
  display: grid;
  grid-template-rows: 100px 150px 100px;
  grid-template-columns: 75px 1fr 1fr 1fr 1fr;
  grid-template-areas: ". a1 a2 a3 a4"
                       "b1 b2 b2 b2 b3"
                       "b1 c1 c2 c2 c2";
}
.a1 { grid-area: a1; }
.a2 { grid-area: a2; }
.a3 { grid-area: a3; }
.a4 { grid-area: a4; }
.b1 { grid-area: b1; }
.b2 { grid-area: b2; }
.b3 { grid-area: b3; }
.c1 { grid-area: c1; }
.c2 { grid-area: c2; }
</style>

<section class="container">
  <div class="yellow a1">A1</div>
  <div class="yellow a2">A2</div>
  <div class="yellow a3">A3</div>
  <div class="yellow a4">A4</div>
  <div class="orange b1">B1</div>
  <div class="orange b2">B2</div>
  <div class="orange b3">B3</div>
  <div class="cyan c1">C1</div>
  <div class="cyan c2">C2</div>
</section>
```



LISTING 7.2 Using grid areas

Grid Areas (ii)

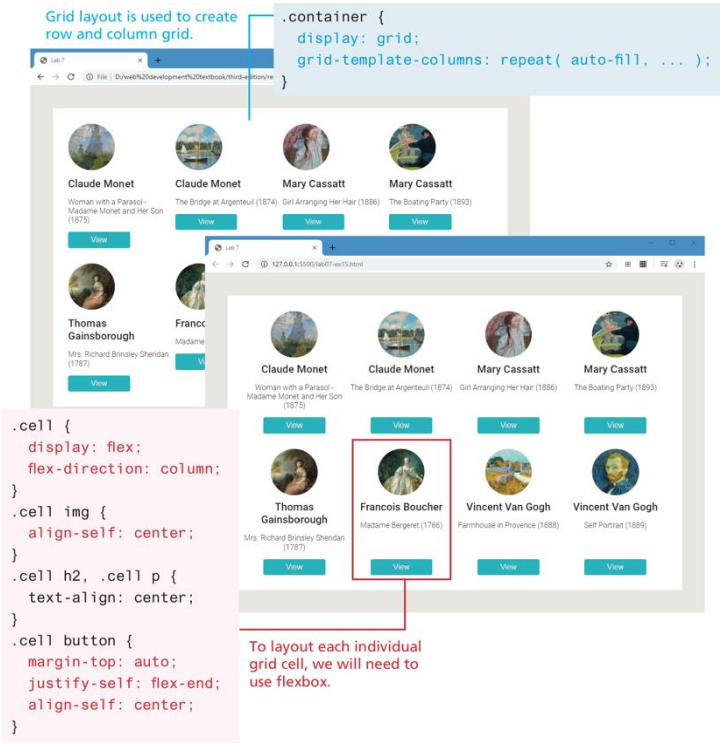




LISTING 7.2 Using grid areas

Grid and Flexbox Together

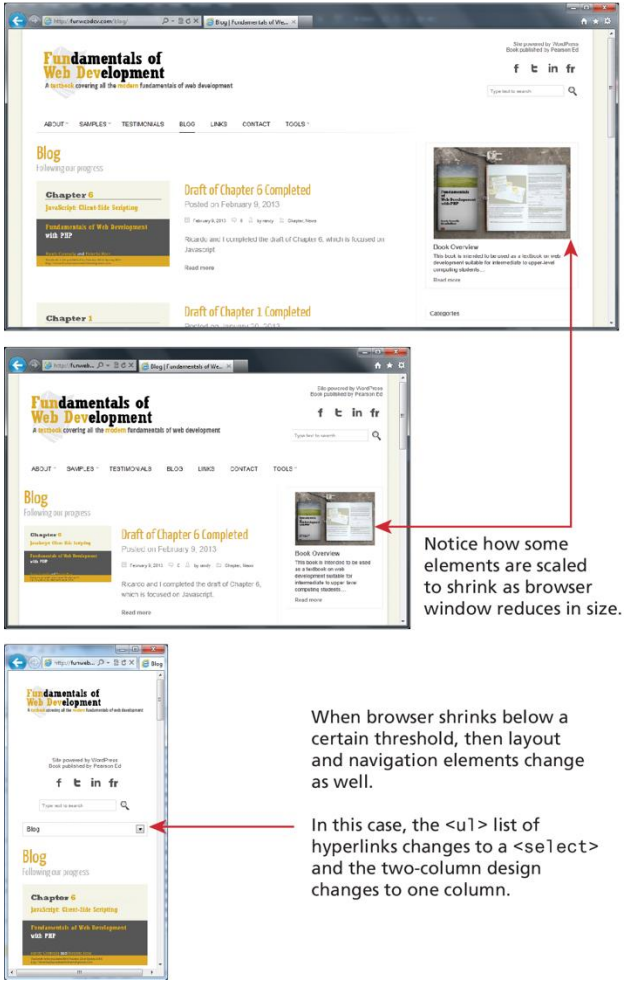
- **grid** and **flexbox** each have their strengths and these strengths can be combined
- **grid** layout is ideal for constructing the layout structure of your page
- **flexbox** is ideal for laying out the contents of a grid cell.



Responsive Design

In a **responsive design**, the page "responds" to changes in the browser size that go beyond simple percentage scaling of widths.

- smaller images will be served and
- navigation elements will be replaced as the browser window shrinks



Mobile First Design

NOTE

Mobile-first design suggests that the first step in the design and implementation of a new website should be the design and development of its mobile version (rather than as an afterthought as is often the case).

The rationale for the mobile-first approach lies not only in the increasingly larger audience whose principal technology for accessing websites is a smaller device such as a phone or a tablet.

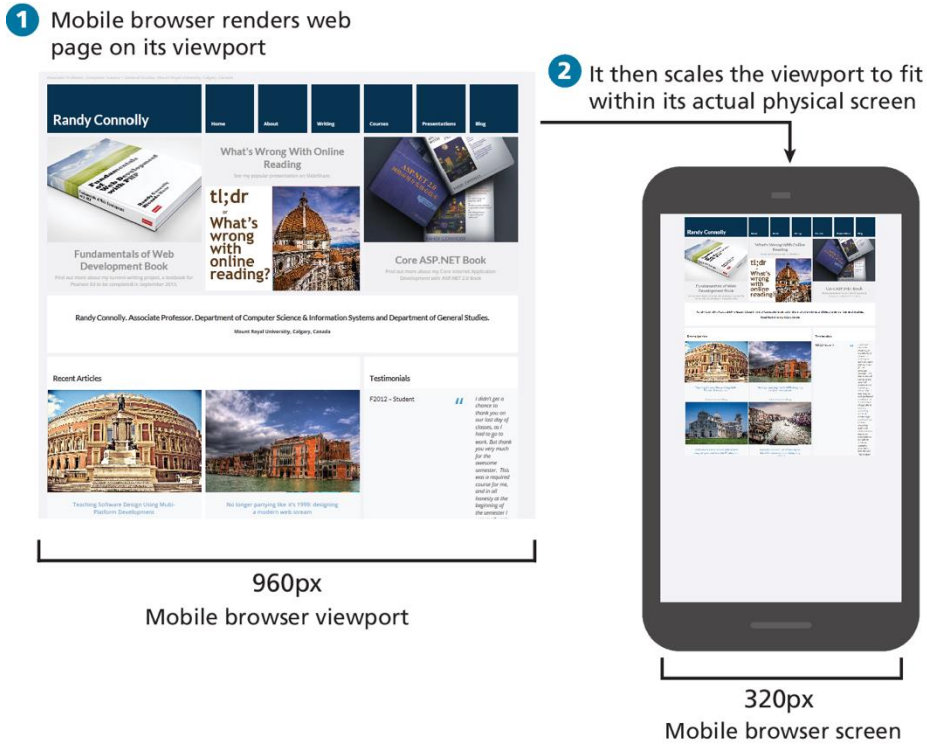
Focusing first on the mobile platform also forces the designers and site architects to focus on the most important component of any site: the content.

Viewports

The browser's **viewport** is the part of the browser window that displays web content.

Mobile browsers will by default scale a web page down to fit the width of the screen.

Generally, results in a viewing experience that works but is very difficult to read and use.

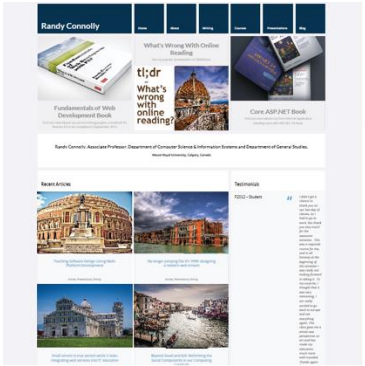


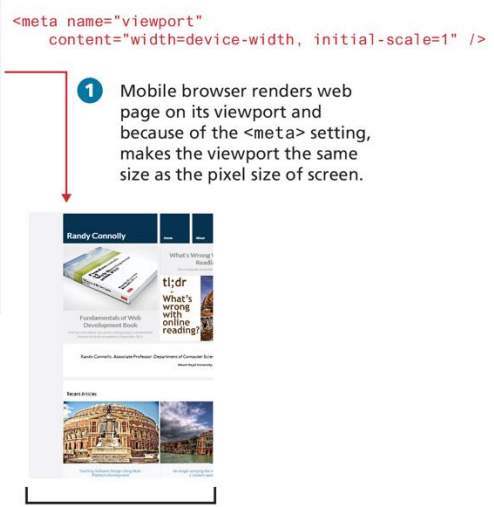
Setting Viewports

```
<html>
<head>
<meta name="viewport"
content="width=device-width, initial-scale=1" />
```

LISTING 7.3 Setting the Viewport

In Listing 7.3, the **width** attribute controls the size of the viewport, while **initialscale** sets the zoom level.

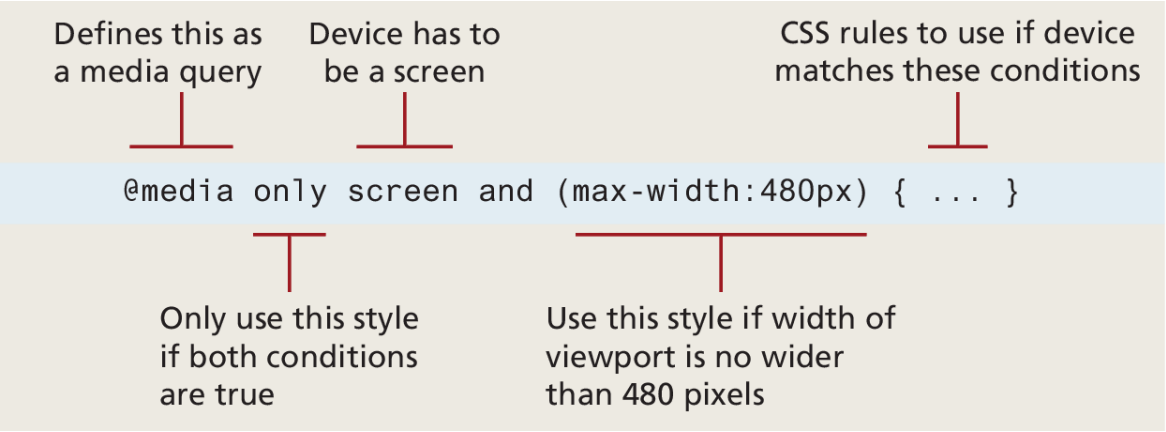




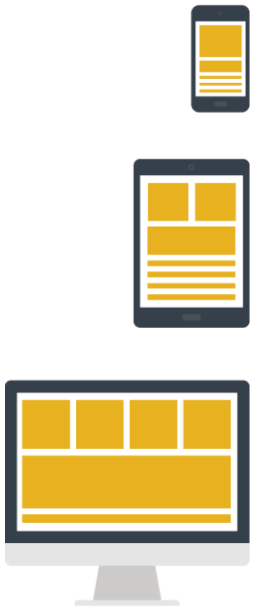
Media Queries

CSS media queries are a way to apply style rules based on the medium that is displaying the file

Contemporary responsive sites will typically provide CSS rules for phone displays first, then tablets, then desktop monitors, an approach called **progressive enhancement**



Media queries in action



styles.css

```
/* 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)" />
```

Scaling images

Making images scale in size is straightforward, but does not change the downloaded size of the image

```
img {
    max-width: 100%;
}
```

HTML5.1 defines the new <picture> element that lets the designer specify multiple <img> elements. The browser determines which <img> to use based on the viewport size.

The <picture> element

```
<picture>
    <source media="(min-width:960px)"
            srcset="images/828-large.jpg">
    <source media="(min-width:480px)"
            srcset="images/828-medium.jpg">
    <img alt="..." src="images/828-small.jpg">
</picture>
```

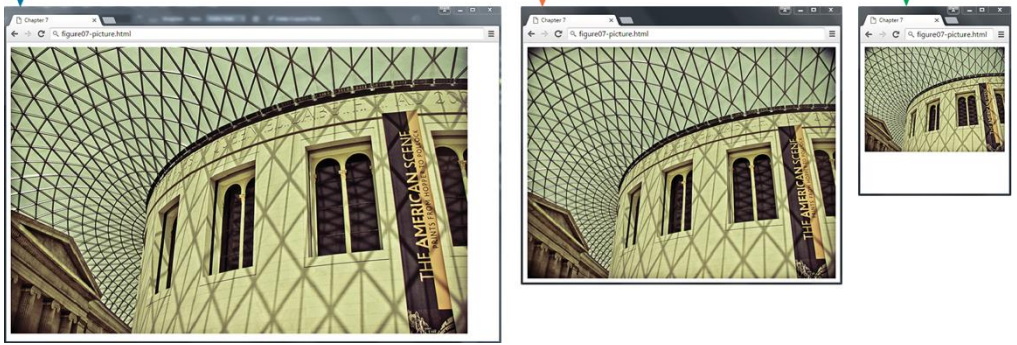
Is this true?

if yes, then use this as the src for the <img>

Is this true?

if yes, then use this as the src for the <img>

Otherwise use the src specified in the <img>



CSS Effects

Four CSS3 modules have become broadly popular amongst designers:

- transformations,
- filters,
- transitions, and
- animations

Transforms

CSS **transforms** provide additional ways to change the size, position, and even the shape of HTML elements.

CSS transforms allow you to **rotate**, **skew**, **transform** (move), and **scale** an element.

It is also possible to transform an element in 3D space using the perspective(), rotate3d(), scale3d(), and translate3d() functions

```
<figure>
   <img src="700.jpg" alt="...">
   <figcaption>Emirates Stadium</figcaption>
</figure>
```


```
figure {
   padding: 1em;
   background: #FFCC80;
   width: 200px;
}
```


```
figure {
   transform: rotate(45deg);
}
```

Notice that the transform affects all the content within the transformed container.


```
figure {
   transform: skew(-20deg);
}
```

Notice that the y-axis extends downwards.


```
figure img {
   transform: translatex(100px) translatey(-30px);
}
```

You can combine transforms.

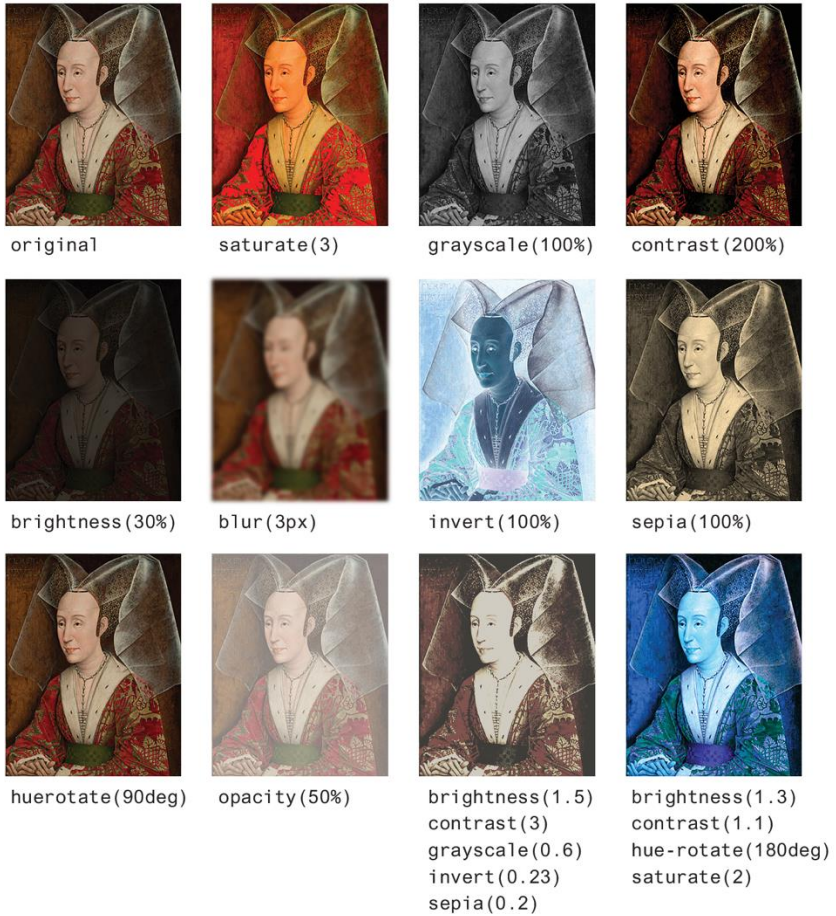

```
figure {
   transform: rotate(15deg);
}
figure img {
   transform: rotate(45deg) scale(0.5);
}
```

Filters

Filters provide a way to modify how an image appears in the browser. Filters are specified by using the filter property and then one or more filter functions

```
#someImage {
   filter: grayscale(100%);
}
#anotherImage {
   /* multiple filters are space separated */
  filter: blur(5px) hue-rotate(60deg) saturate(2);
}
```

LISTING 7.4 Using a Filter



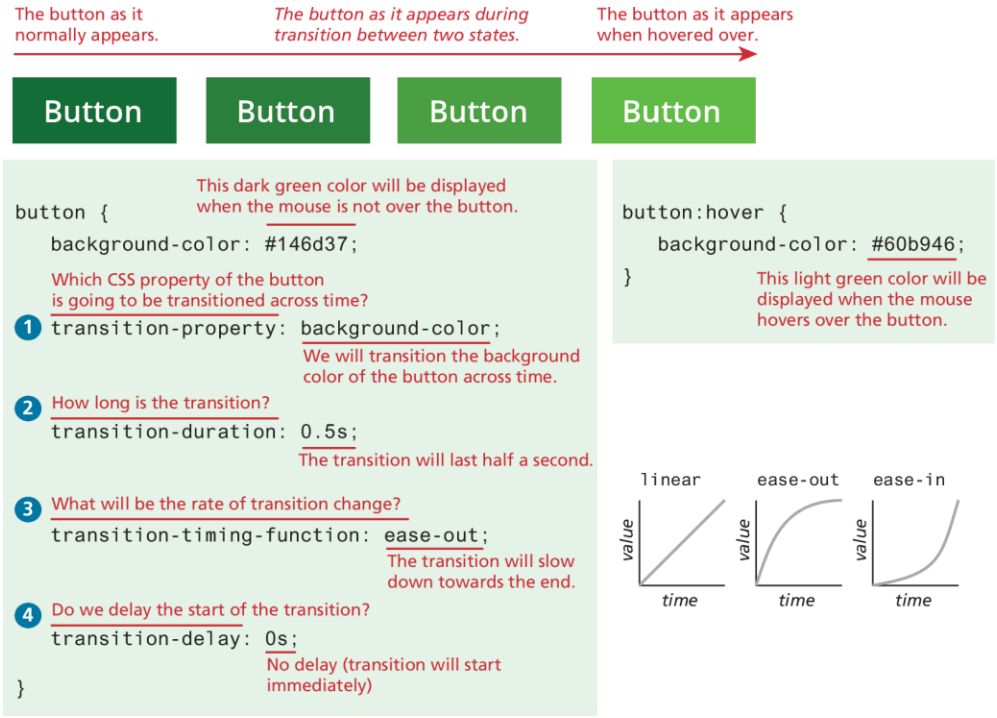
Transitions

Transitions are a powerful new feature of CSS3. **Transitions** will take effect across a length of time. provide a way to indicate that a property change

- **transition-delay** The delay time in seconds before the animation begins.
- **transition-duration** How long in seconds for the transition to complete.
- **transition-property** The name of the CSS property to which the transition is applied.
- **Transition-timing-function** The function that defines how the intermediate steps in the transition are calculated. CSS defines a variety of different easing functions which define the rate of the transition.

Transition with example

1. The CSS property which will be transitioned.
2. The duration of the transition.
3. The easing function to use, which changes the speed and style of the transition(optional).
4. How long to delay before starting the transition (optional).



A sliding menu transition example

When the user hovers the mouse over the visible part of the menu , it appears to "slide" out from the left and become visible.

```
<nav class="menu">
   <p><i class="fa fa-chevron-right"></i></p>
   <ul>
      <li><a href="#">Home</a></li>
      <li><a href="#">Blogs</a></li>
      <li><a href="#">Photos</a></li>
      <li><a href="#">Contact</a></li>
   </ul>
</nav>
```

Menu is initially hidden by being positioned outside the visible area.

```
.menu {
   position: absolute;
   left: -210px;
}
```

```
.menu:hover {
   left: 0;
   transition: left .6s ease-out;
}
```

When the user hovers over the menu, move the left edge of the element to left edge of the browser (i.e., it will now be visible).

Using the transition shorthand property.

Transition the `left` property across 0.6 seconds and use the ease-out function (i.e., slow down transition at end).

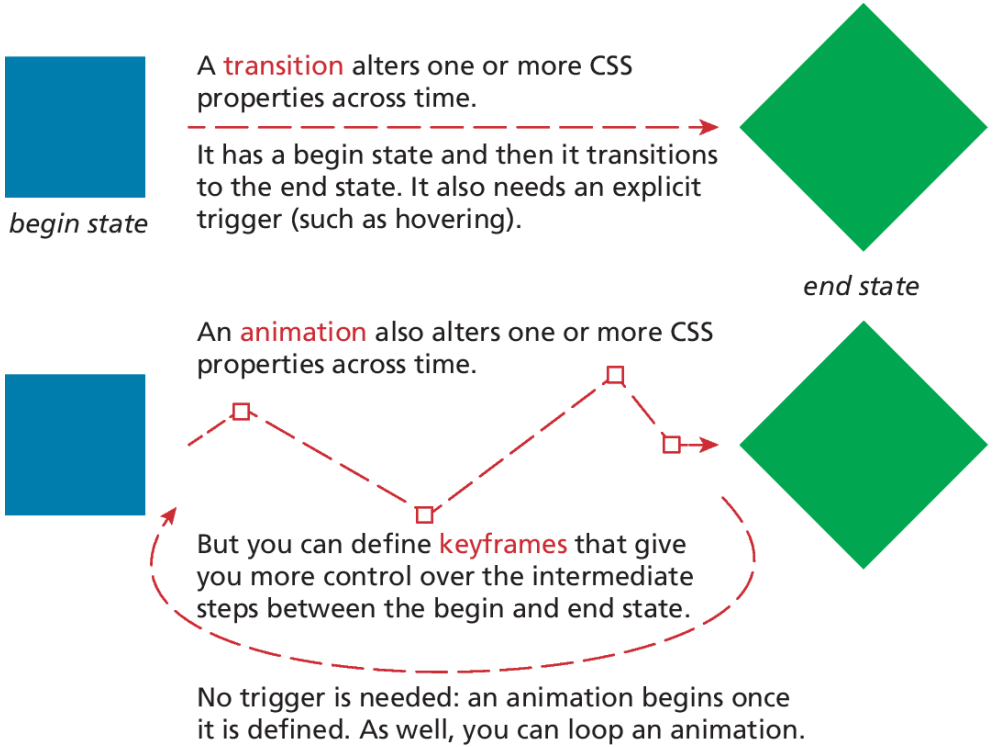
```
.menu {
   transition: left .6s ease-out;
}
```

We want the same transition when the mouse is no longer hovering over the menu. This creates illusion of menu sliding back out of sight.

Animations

An **animation** alters one or more properties between a start state and an end state (like a transition).

Unlike transitions, animations allows designers to control the intermediate steps through **keyframe** states.



Animation example

```
<p class="animated">Animate Me</p>
```

```
.animated {
   animation-iteration-count: infinite;
   animation-name: bounceIn;
   animation-play-state: running;
   animation-duration: 2s;
   animation-timing-function: ease-out;
   animation-delay: 1s;
}

.animated:hover {
   animation-play-state: paused;
}
```

- Run animation indefinitely.
- Play animation named bounceIn.
- Play animation once it is defined.
- Animation lasts 2 seconds.
- Slow animation towards the end.
- Wait a second before starting animation.
- Pause the animation by hovering over it. (useful for debugging).

CSS preprocessors

CSS preprocessors allow the developer to write CSS that takes advantage of programming ideas such as variables, inheritance, calculations, and functions.

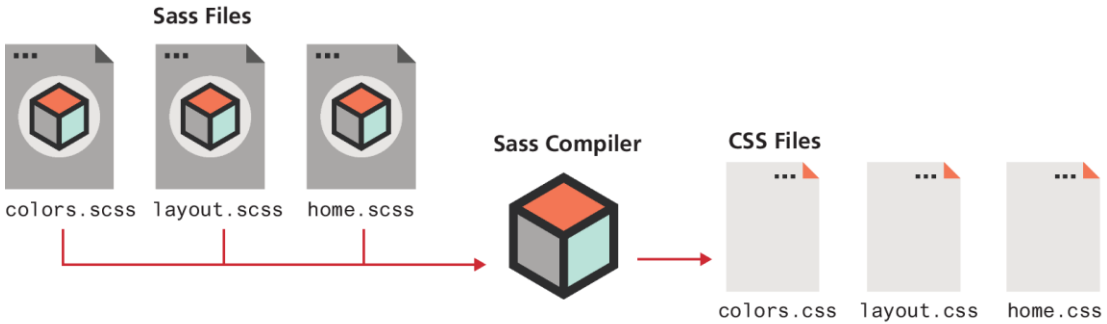
CSS itself has many limitations:

- No variables (prior to 2018).
- No encapsulation.
- No scoping rules
- No modularity. Duplication. Duplication. Duplication.

CSS preprocessor addresses these limitations. A developer can use variables, nesting, functions, or inheritance to handle duplication and avoid copy and pasting, and search and replacing.

Sass

Via command line or GUI tool, a Sass compiler compiles one or more Sass files into a regular CSS file that can be referenced in the usual way via the <link> element.



Sass Variables and Types

In Sass, a variable declaration looks like a property declaration in CSS.

The following code defines two Sass variables and then references them.

```
$primary-color: #647ACB;
$spacing: 20px;
.box {
    background-color: $primary-color;
    margin-top: $spacing;
}
```

Sass nesting

- Sass provides a way to nest your styling. Listing 7.6 demonstrates how you can potentially simplify your styling using Sass nesting and variables whose scope is limited to a single block (and to its children).

```
.container {
    .sidebar {
        $side-color : red;
        color: $side-color;
    h2 {
        color: $side-color; /* .sidebar h2 color is red */
        ...
   }
}
main {
    ...
    $main-color: black;
    header {
        h2 {
                color: $main-color;
                /* .sidebar main h2 color is black */
        }
        ...
     }
     article {
        h2 {
              color: $main-color;
...}}}}
```

LISTING 7.6 Using Sass nesting

Mixins and Functions

One of the key limitations of CSS is that even though there is often a lot of repetitive styling, outside the recent CSS variables there is no language feature for eliminating it.

Mixins in Sass are like a function that returns a style.

```
$mid-neutral : #9AA5B1;
$shadow : 0 0 3px 3px $mid-neutral;

@mixin card() {
  border: solid 1px $mid-neutral;
  margin: 3px;
  box-shadow: $shadow;
}

.box {
  @include card();
  background-color: white;
}

.feature {
  @include card();
}
```

This SCSS will compile into the following CSS.

```
.box {
  border: solid 1px #9AA5B1;
  margin: 3px;
  box-shadow: 0 0 3px 3px #9AA5B1;
  background-color: white;
}

.feature {
  border: solid 1px #9AA5B1;
  margin: 3px;
  box-shadow: 0 0 3px 3px #9AA5B1;
}
```

Key Terms

- absolute positioning
- animation
- BEM
- block-level elements
- box-level elements
- card
- containing block
- CSS frameworks
- CSS media queries
- CSS preprocessors
- grid layout
- filters
- fixed positioning
- flexbox layout
- flex container
- flex items
- float property
- iframe
- implicit grid
- inline elements
- keyframe
- minification
- mobile-first design
- normal flow
- positioning context
- progressive enhancement
- relative positioning
- responsive design
- Sass
- style guide
- transforms
- transitions
- UI design tool
- viewport

Copyright

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.