CSS3

by Mike McGrath

Chapter 4: Controlling Layout

This chapter demonstrates how to position content boxes to control page layout.

Centering Content Boxes

One of the basic requirements in displaying document components is the ability to horizontally center blocks of content – in the same way provided by the old HTML ** ** tag.

In CSS the ability to horizontally center block-level elements is provided by the **margin** property and the **auto** keyword.

Beware Assigning an **auto** value to the **margin** property removes top and bottom margins – just as if they had been assigned a zero value.

When the **auto** keyword is specified to an element's **margin** property the browser first calculates the distance to the left and right of that element, up to the boundaries of its containing element, then divides the total in half to compute the value of each side margin. For example, applying a **margin : auto** to a "div" element of 80px width, that is contained within an outer element of 200px width, the browser divides the total difference of 120px in half then applies 60px wide margins to each side of the div element – so it gets positioned centrally within the containing element.

Notice that **margin : auto** does not center vertically but merely sets the top and bottom margins to zero, so there are no margin areas above or below the element block.

It is important to recognize that **margin : auto** centers block-level content boxes and should not be confused with **text-align : center** that can be used to center content within inline content boxes.

1. Create an HTML document with a "div" element containing two further elements

```
<div>Default Position
  <div class="inner">Default Position</div>
  <div class="inner center">Centered Block</div>
</div>
```

center.html

2. Now add another "div" element containing three further elements

```
<div class="center">Centered Block
<div class="inner">Default Position</div>
<div class="center inner">Centered Block</div>
<div class="center inner center-text">
                Centered Block + Centered Content</div>
</div>
```

3. Save the HTML document then create a linked style sheet containing a style rule to specify default width, margin, and colors for each element

```
div { width : 20em; margin : 0 0 2em 0;
        border : 0.1em solid black; background : orange }
```

center.css

4. Next add a style rule over-writing the previous rule for inner elements

```
div.inner { margin : 0; width : 10em; background : yellow }
```

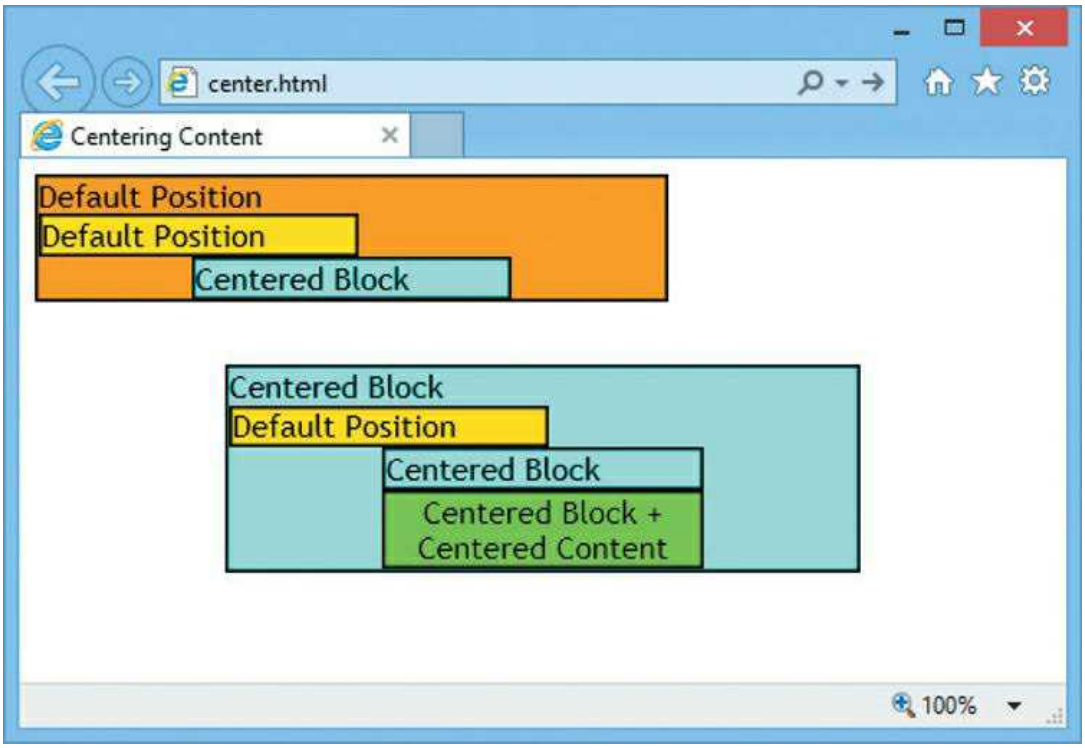
5. Now add a rule to center inner elements

```
div.center { margin : auto; background : aqua }
```

6. Finally add a style rule to center content within an inner element

```
div.center-text { text-align : center; background : lime }
```

7. Save the style sheet alongside the HTML document then open the web page in a browser to see the centered content boxes and centered content



Hot Tip When the browser window gets resized the second outer "div" element automatically gets repositioned to become horizontally centered again.

Positioning Boxes Absolutely

When laying out the element content boxes of a web page the CSS **position** property has a default value of **static** – representing the normal flow positioning scheme. Assigning a different value to an element's **position** property allows that element to be removed from the normal flow so it can be positioned independently.

Alternatives to the default **static** value can be specified using the **absolute, relative**, and **fixed** keywords to specify an alternative positioning scheme to that of the normal flow layout.

Don't Forget Each outer content box remains at its absolute position to the window when the browser window gets resized.

The **absolute, relative**, and **fixed** positioning schemes each use one or more of the CSS "offset" properties **top, right, bottom**, and **left**, to define their position.

When the position property is specified as **absolute** the positioning scheme places the element at the specified offset distance from the boundaries of its containing element. For example, a "div" element with top and left values of 100px will be positioned 100px below and to the right of its container boundaries.

1. Create an HTML document with four "div" elements that each contain an inner div element

```
<div class="top-left large">Top:0 Left:0
 <div class="btm-right small">Bottom:0 Right:0</div>
</div>

<div class="top-right large">Top:0 Right:0
```

```
 <div class="btm-left small">Bottom:0 Left:0</div>
</div>

<div class="btm-left large">
                          <br><br><br>Bottom:0 Left:0
<div class="top-right small">Top:0 Right:0</div>
</div>
<div class="btm-right large">
                           <br><br><br>Bottom:0 Right:0
<div class="top-left small">Top:0 Left:0</div>
</div>
```

2. Save the HTML document then create a linked style sheet with rules to set the absolute size of the elements

```
div.large { width : 200px; height : 90px;
                                  border : 1px solid black }
div.small { width : 80px; height : 50px;
                                  border : 1px solid black }
```

3. Now add style rules to set the absolute position of each element

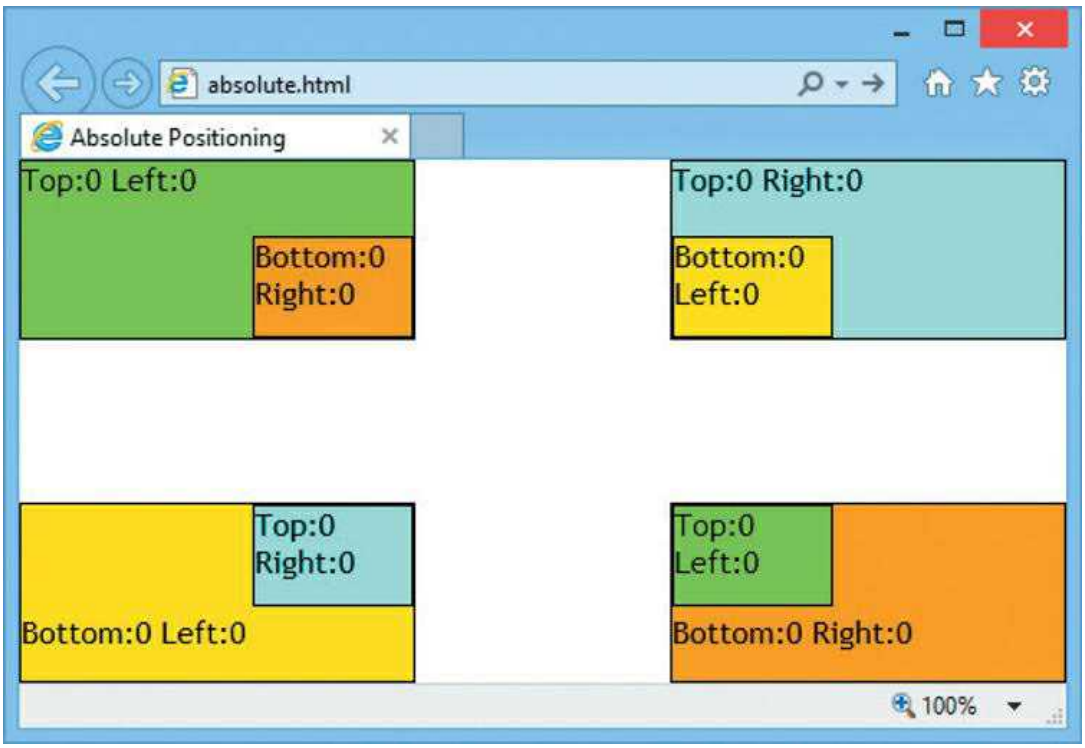
```
div.top-left { position : absolute; top : 0px; left : 0px;
                                          background : lime }

div.top-right { position : absolute; top : 0px; right : 0px;
                                           background : aqua }

div.btm-left { position : absolute;
           bottom : 0px; left : 0px; background : yellow }

div.btm-right { position : absolute;
         bottom : 0px; right : 0px; background : orange }
```

4. Save the style sheet alongside the HTML document then open the web page in a browser to see the content boxes positioned at absolute coordinates



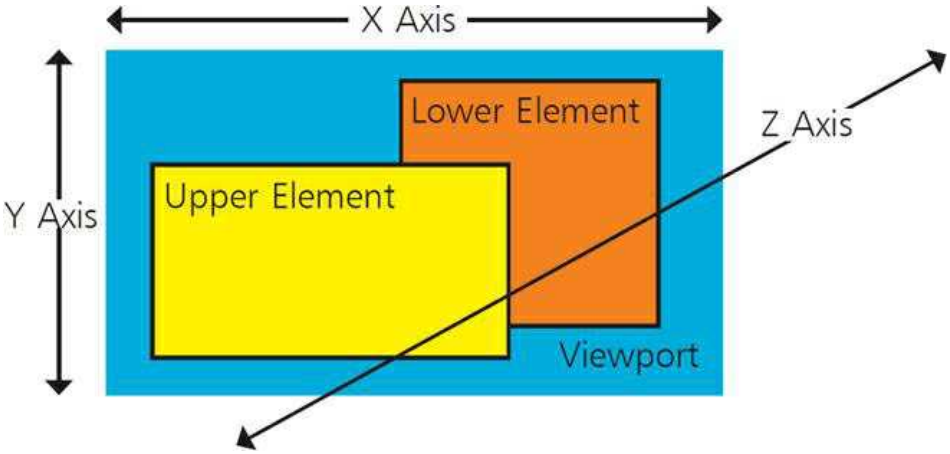
Stacking Content Boxes

Changing from the **static** default positioning scheme, by assigning the **absolute** value to the **position** property, allows elements to overlap – stacking one above the other in the same order they are listed in the HTML code.

The stacking order can be explicitly specified, however, in CSS by assigning an integer value to the **z-index** property of each element. The element with the highest value appears uppermost, then beneath that appears the element with the next highest value, and so on.

Don't Forget "Viewport" is simply another name for the browser window.

So the **absolute** positioning scheme allows the element position to be precisely controlled in three dimensions using XYZ coordinates – along the X axis with the **left** and **right** offset properties, along the Y axis using the **top** and **bottom** offset properties, and along the Z axis using the **z-index** stacking order property.



1. Create an HTML document containing four "div" elements to be positioned overlapping

```
<div id="pale-gray"> </div>

<div id="lite-gray"> </div>

<div id="dark-gray"> </div>

<div id="text-box">Text content here on the top of the element
```

```
stack - lower offset elements create a drop
shadow effect</div>
```

2. Save the HTML document then create a linked style sheet containing a rule to set the size of all elements

```
div { width : 400px; height : 50px }
```

3. Next add a style rule to position the uppermost element

```
div#text-box { position : absolute;
               top : 20px; left : 20px;
               background : yellow; z-index : 4 }
```

4. Now add a rule to position the next element below

```
div#dark-gray { position : absolute;
                top : 24px; left : 24px;
                background : #666; z-index : 3 }
```

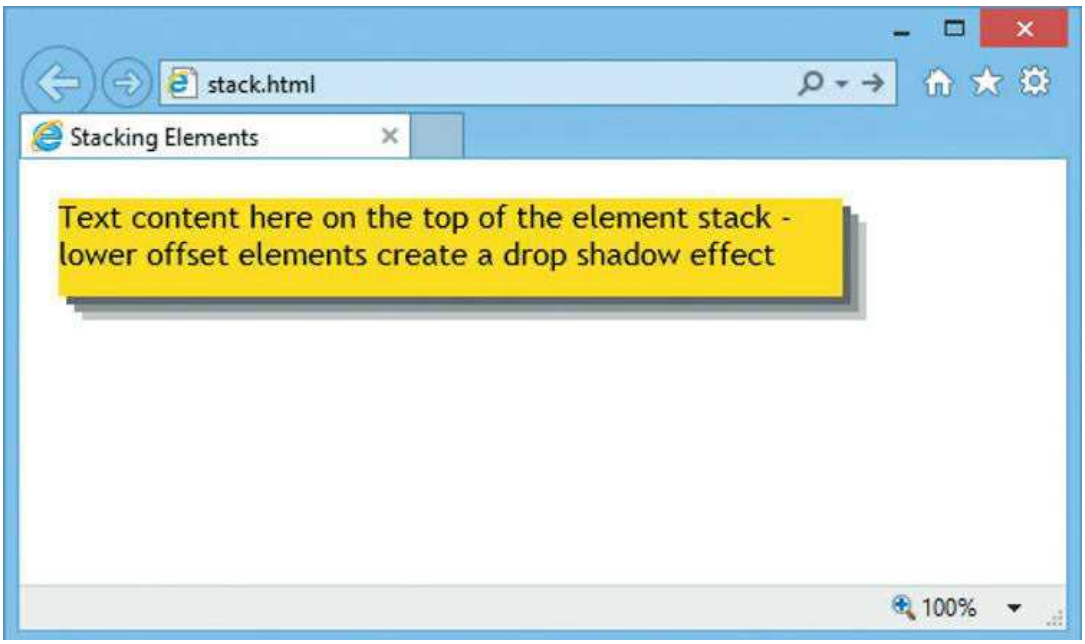
5. Then add a rule to position an element further below

```
div#lite-gray { position : absolute;
                top : 28px; left : 28px;
                background : #999; z-index : 2 }
```

6. Finally add a rule to position the lowest element

```
div#pale-gray { position : absolute;
                top : 32px; left : 32px;
                background : #CCC; z-index : 1 }
```

7. Save the style sheet alongside the HTML document then open the web page in a browser to see the elements positioned in all three dimensions



Hot Tip A smaller offset creates a more convincing drop shadow effect – a large offset is used here to illustrate the contribution of each element.

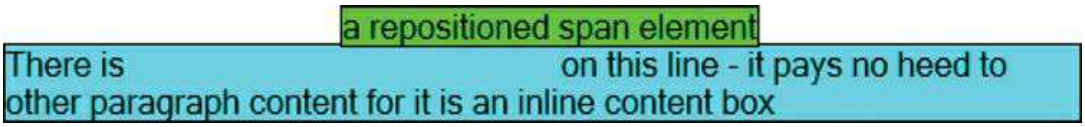
Positioning Boxes Relatively

In contrast to the **absolute** positioning scheme, which entirely removes an element from the normal flow layout, a **relative** positioning scheme adjusts the position of an element relative to the position it would originally occupy in the normal flow layout.

Changing from the default **static** positioning scheme, by assigning the **relative** value to the CSS **position** property, allows elements to be repositioned from their normal flow layout position using the offset properties **top, right, bottom** and **left.**

Beware Notice how a negative value is used here – these can be used with other properties too but may sometimes produce unexpected results.

For example, specifying a **top** value of −18px moves the selected element up and specifying a **left** value of 100px moves it to the right – but crucially the space occupied by its original layout position is preserved. Applying these relative position values to a span element within a paragraph has this effect:



Notice how the original content is shifted from its normal flow layout position into a newly-created content box positioned at the specified distance relative to its original position. This relative position will be maintained, even when the position of the outer containing element is changed.

So while **absolute** positioning may typically control the position of the outer element the **relative** positioning scheme is often useful to control the position of nested inner elements.

1. Create an HTML document containing four nested "div" elements to be positioned overlapping

```
<div id="pale-gray">
 <div id="lite-gray">
  <div id="dark-gray">
   <div id="text-box">Text content here on the innermost
        element - outer offset elements create a drop
        shadow effect</div>
  </div>
 </div>
</div>
```

2. Save the HTML document then create a linked style sheet containing a rule to set the size and color of the outermost containing element

```
div#pale-gray { width : 400px; height : 50px;
                    background : #CCC; margin : 20px }
```

3. Next add a style rule to position the first nested element relative to the outermost containing element

```
div#lite-gray { position : relative; left : -4px; top : -4px;
                         height : 100%; background : #999 }
```

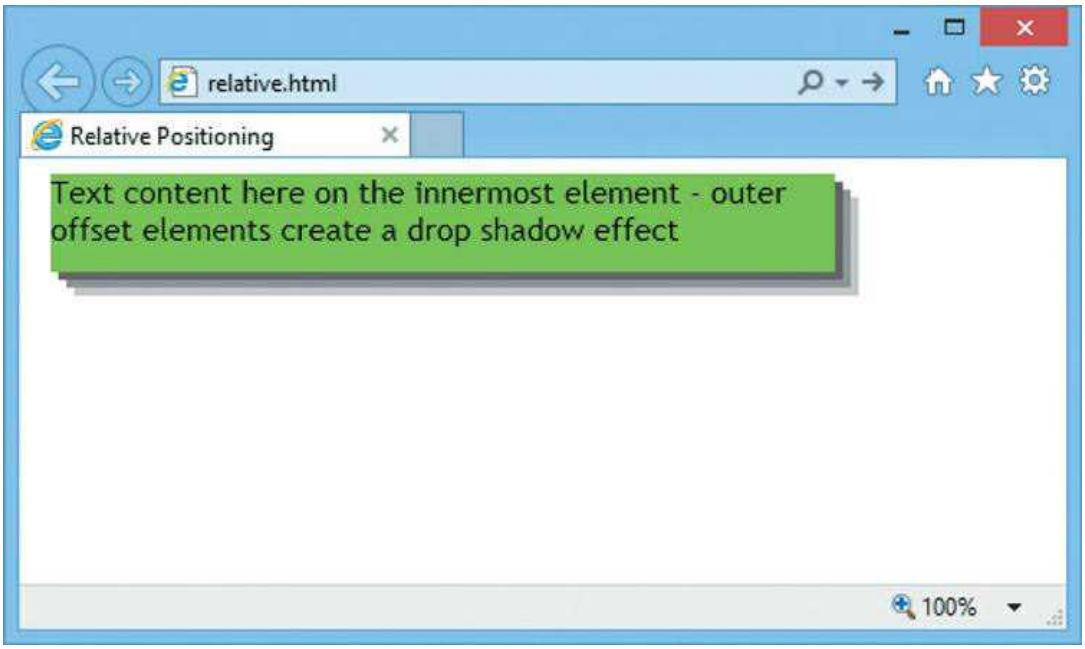
4. Now add a style rule to position the second nested element relative to the first nested element

```
div#dark-gray { position : relative; left : -4px; top : -4px;
                       height : 100%; background : #666 }
```

5. Finally add a style rule to position the innermost nested element relative to the second nested element

```
div#text-box { position : relative; left : -4px; top : -4px;
                        height : 100%; background : lime }
```

6. Save the style sheet alongside the HTML document then open the web page in a browser to see the nested elements positioned relative to their containing elements



Hot Tip The **margin** property is only set on the outer element simply to move the content away from the edge of the window.

Fixing Constant Positions

A **fixed** positioning scheme, like the **absolute** positioning scheme, completely removes the selected element's content box from the normal flow layout. But unlike **absolute** positioning, where offset values relate to the boundaries of the containing element, in **fixed** positioning the offset values relate to the viewport – the position is relative to the browser window, not to any part of the document.

Usefully, element positions can be fixed to emulate a frame-style interface where some frames remain at a constant position regardless of how the regular page is scrolled. For example, a banner header frame can be fixed at the top of the page so it remains constantly visible even when the page is scrolled.

1. Create an HTML document containing a heading element and a paragraph with a tall image to scroll

```
<h1>Header</h1>
<p>
<img src="ruler.png" width="71" height="521" alt="Ruler">
</p>
```

fixed.html

2. Save the HTML document then create a linked style sheet with a rule to fix the heading element to the top of the browser window and above other page content

```
h1 { position : fixed; top : 0; left : 0; margin : 0;
            background : lime; width : 100%; z-index : 2 }
```

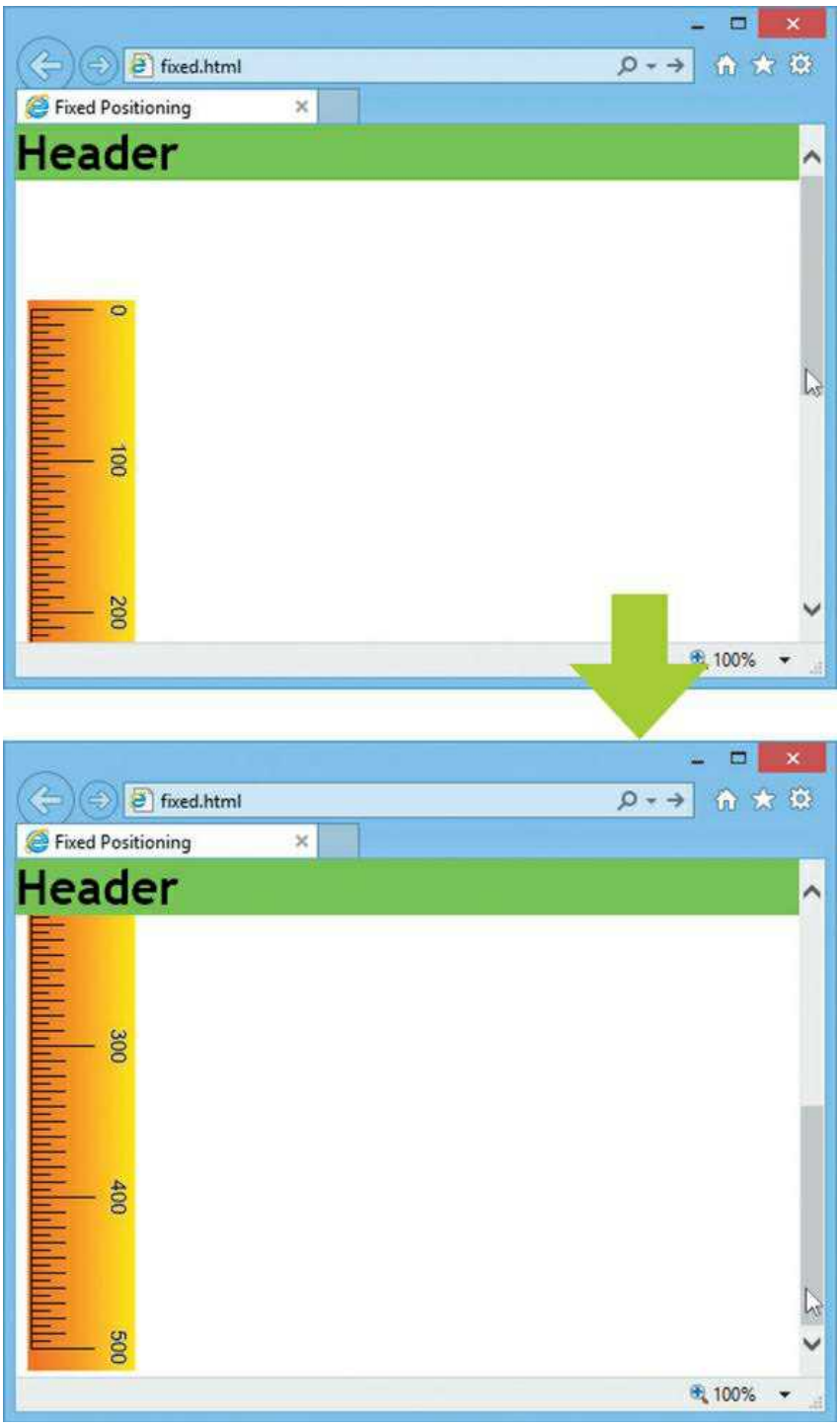
fixed.css

3. Now add a style rule to position the paragraph below the heading element

```
p { position : absolute; top : 100px; z-index : 1 }
```

4. Save the style sheet alongside the HTML document then open the web page in a browser to see the elements positioned by the style rules

5. Now scroll the browser window to see the heading remain in place while the paragraph content slides beneath it



ruler.png

Hot Tip Fixed positioning is sometimes used to display an identity logo that remains visible at a constant position regardless of where the page is scrolled.

Floating Content Boxes

The CSS **float** property allows a content box to be positioned at the side boundary of its containing element – using the **left** or **right** keywords to specify the preferred side. Typically this feature is used to flow text around images that have been floated to the side of a containing paragraph element.

You can also explicitly prevent text flowing alongside a floated content box using the CSS **clear** property – specifying **left, right**, or **both** keywords to determine which side must be clear, so that text will begin below the floated content box.

1. Create an HTML document containing three paragraphs and two images

```
<p>Massive acceleration - the forbidden fruit! It's easy to
avoid such unlawful <img src="viper-1.png" width="150"
height="128" alt="Viper Front"> activities in a normal
vehicle. But there is an evil serpent; a Viper that tempts
you to take a bite out of the asphalt. With a tasty 500-hp
V10 powering a mere 3,300-lb roadster, the Dodge Viper
SRT-10 tricks you into playing music with the loud pedal.</p>

<p>This car is too excessive, too epic for most people to
use on a daily basis. <img src="viper-2.png" width="155" height="115"
alt="Viper Rear">But for otherwise nice
couples who need only two seats this is the car that will
shame those who come up against them.</p>

<p class="clear">If you can afford to... Buy one. You'll
```

```
like it.</p>
```

2. Save the HTML document then create a linked style sheet with a rule to color all paragraph backgrounds

```
p { background : yellow }
```

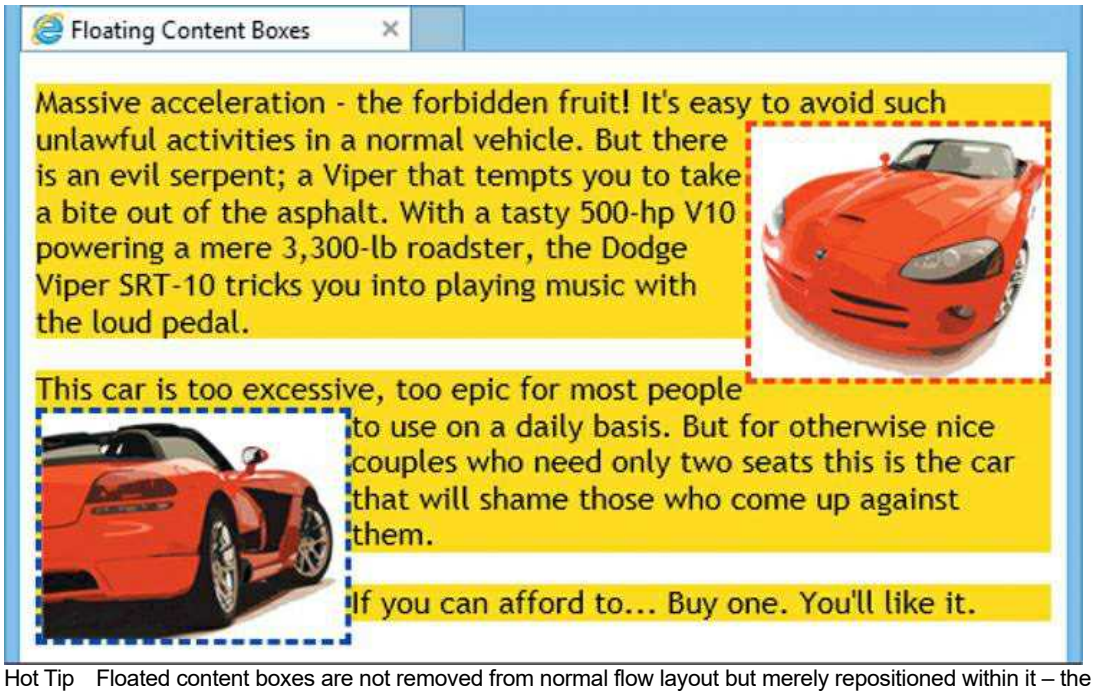
3. Now add a style rule to float the first image to the right side of its containing paragraph element and add a border

```
img[src="viper-1.png"] { float : right;
                         border : 3px dashed red }
```

4. Next add a style rule to float the second image to the left side of its containing paragraph element and add a border

```
img[src="viper-2.png"] { float : left;
                         border : 3px dashed blue }
```

5. Save the style sheet alongside the HTML document then open the web page in a browser to see the floated images

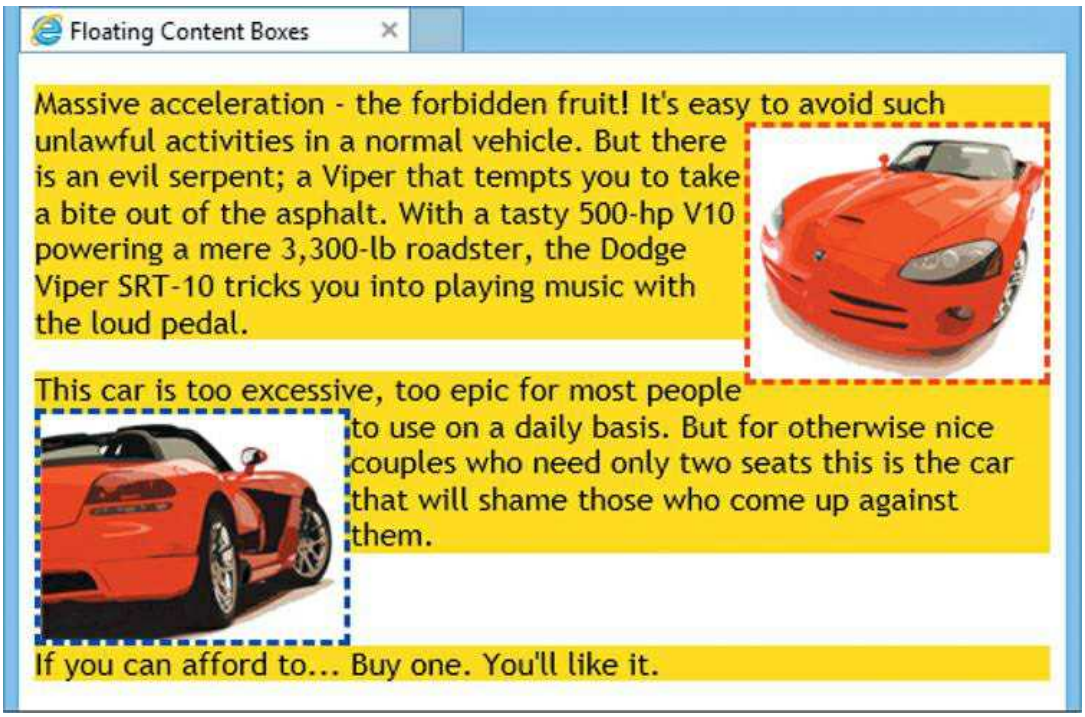


Hot Tip Floated content boxes are not removed from normal flow layout but merely repositioned within it – the space at their original position gets filled with surrounding content.

6. Edit the style sheet to add a rule that prevents the final paragraph flowing alongside the second floated image

```
p.clear { clear : left }
```

7. Save the style sheet once more then refresh the browser window to see the final paragraph now appear below the second image



Hot Tip With CSS any element can be floated to the side of its containing element by setting its **float** property.

Clipping & Handling Overflow

Although CSS provides many controls to specify the precise size and position of content boxes there is no guarantee that their content will fit neatly within their boundaries in all circumstances. For example, consider the effect of increasing the font size of text content that fits snugly within a block-level content box – the text will then "overflow" outside the box boundaries.

Overflowing content is generally **visible** by default, but the CSS **overflow** property can specify alternative handling behaviors using the **hidden** or **scroll** keywords.

Conversely, the content within absolutely-positioned block-level content boxes can be cropped using the **clip** property – the rectangular section to remain visible being identified by the coordinates of its corner points. In CSS these coordinates are assigned to the **clip** property as a comma-separated list within the parentheses of the special **rect()** keyword, always in the order of top-left, top-right, bottom-right, bottom-left.

1. Create an HTML document with six "div" elements that each contain the same image

```
<div class = "crop auto-clip">
            <img src = "patch.png" alt="Blocks"></div>

<div class = "crop blok-clip">
            <img src = "patch.png" alt="Blocks" ></div>

<div class = "crop line-clip">
            <img src = "patch.png" alt="Blocks" ></div>

<div class = "spill show-overflow">
            <img src = "patch.png" alt="Blocks" ></div>

<div class = "spill hide-overflow">
            <img src = "patch.png" alt="Blocks" ></div>

<div class = "spill keep-overflow">
            <img src = "patch.png" alt="Blocks" ></div>
```

overflow.html

2. Save the HTML document then create a linked style sheet with rules that specify the absolute position and size of each div element

```
div.crop { position : absolute; top : 20px;
                          width : 100px; height : 100px }
div.spill { position : absolute; top : 150px;
                                width : 75px; height : 75px }
```

3. Add a style rule that allows the first image to be automatically cropped to the full image size

```
div.auto-clip { left : 20px; clip : auto }
```

4. Next add style rules that crop the visible part of the second and third image to reveal only those parts of the image within specified coordinates

```
div.blok-clip
{ left : 140px; clip : rect(25px, 100px, 100px, 25px) }

div.line-clip
{ left : 260px; clip : rect(25px, 100px, 50px, 0px) }
```

Hot Tip The ability to toggle content visibility, between visible and hidden, presents exciting dynamic possibilities – see Chapter 7 for more details and examples.

5. Now add a style rule that allows those parts of the fourth image that overflow the content box to be visible

```
div.show-overflow { left : 20px; overflow : visible }
```

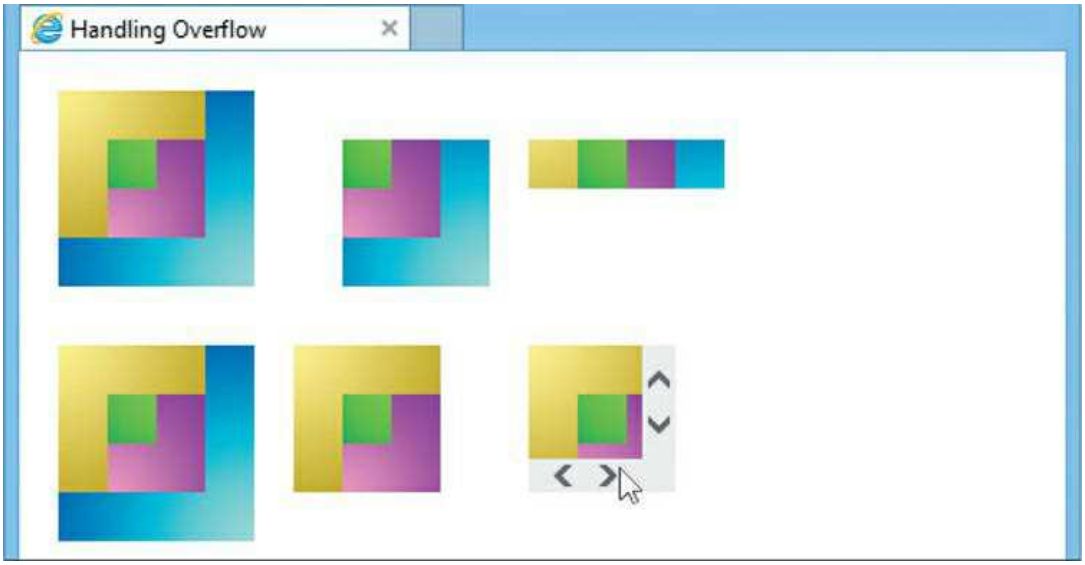
6. Add a style rule to hide those parts of the fifth image that overflow the content box

```
div.hide-overflow { left : 140px; overflow : hidden }
```

7. Finally add a style rule to hide those parts of the sixth image that overflow the content box and provide scroll bars so hidden parts can be viewed

```
div.keep-overflow { left : 260px; overflow : scroll }
```

8. Save the style sheet alongside the HTML document then open the web page in a browser to see the cropped images and compare how the overflow is handled



Constructing Columns

Traditionally, web page authors have used HTML tables to control how page content is laid out by creating a borderless grid of cells into which components of the page are placed. This method works well enough for general layout but lacks the precise control offered by CSS. For example, the **absolute** positioning scheme allows components to be positioned easily with pinpoint accuracy but this could not be achieved easily using tables. Thus CSS provides a superior layout method so web page authors are now discouraged from using HTML tables for this purpose.

Moving to page layout with CSS invariably raises questions of how best to control the page content. The following example suggests one way to create a common page layout featuring a header and footer plus three content columns.

1. Create an HTML document containing "div" elements for header and content panels, all wrapped in an outer container element that is followed by a footer element

```
<div id="wrapper">

<div id="hdr">Header Panel<br>
<img src="tile.png" width="250" height="25" alt="Box">
</div>

<div id="nav">Navigation Panel
<ul><li>Link<li>Link</ul></div>

<div id="ads">Supplement Panel<br>
<img src="tile.png" width="75" height="100" alt="Box">
</div>

<div id="txt">Content Panel<br>
<img src="tile.png" width="150" height="200" alt="Box">
</div>

</div> <div id="ftr">Footer Panel</div>
```

2. Save the HTML document then create a linked style sheet with a rule to remove all margins and padding, and to center all content

```
* { margin : 0; padding : 0; text-align : center }
```

3. Add a rule to ensure the page is the entire window height

```
html, body { height : 100% }
```

4. Now add a rule to have the container element use the entire window area – except for a bottom margin area where the footer will appear

```
#wrapper { min-height : 100%; height : auto !important;
      height : 100%; margin : 0 auto -30px }
```

5. Next add style rules to set the header and footer heights and background color

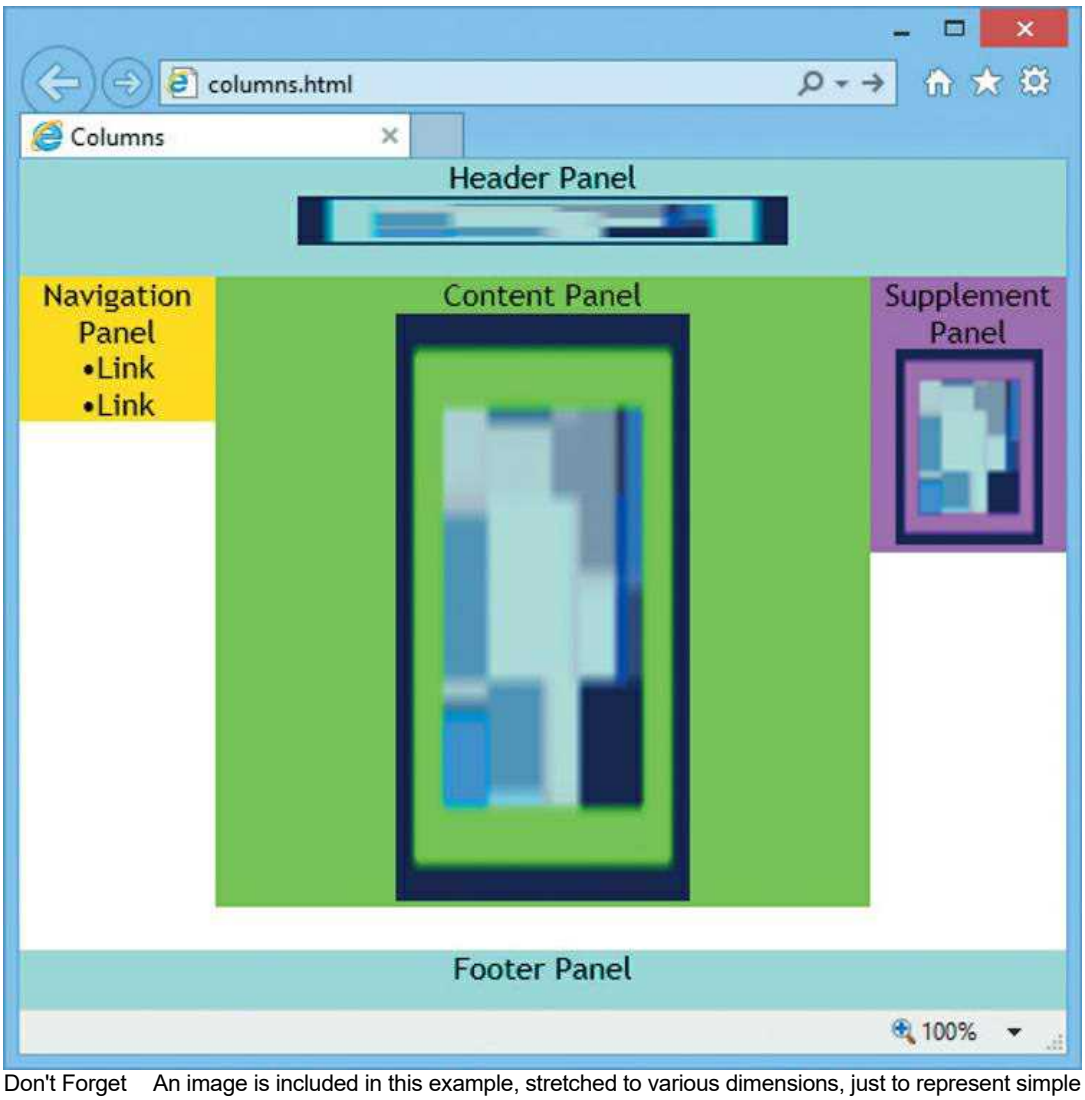
```
#hdr { height : 60px; background : aqua }
#ftr { height : 30px; background : aqua }
```

Hot Tip This example solves the issue of how to create a "sticky footer" that always remains at the bottom of the window.

6. Finally add rules to position and color the three columns

```
#nav { float : left; width : 100px; background : yellow }
#ads { float : right; width : 100px; background : fuchsia }
#txt { margin : 0 100px; background : lime }
```

7. Save the style sheet alongside the HTML document then open the web page in a browser to see the page layout



Don't Forget An image is included in this example, stretched to various dimensions, just to represent simple page content.

Summary

- A **margin : auto** rule horizontally centers block-level content boxes, acting much like the old HTML ** ** tag
- A **text-align : center** rule horizontally centers text within a content box
- The **absolute, relative**, and **fixed** positioning schemes use the **top, right, bottom**, and **left** properties to specify position
- A **position : absolute** rule removes an element from the normal flow layout and positions it at a specified distance from the boundaries of its containing element
- The stacking order of overlapping elements can be specified by assigning integer values to their **z-index** property
- A **position : relative** rule moves an element from the position it would occupy in the normal flow layout and preserves the space it would have occupied
- A **position : fixed** rule removes an element from the normal flow layout and positions it at a specified distance from the boundaries of the viewport
- The **float** property accepts **left, right**, or **both** keywords to reposition an element within the normal flow layout and fill the space it would have occupied with content
- The **clear** property accepts **left, right**, or **both** keywords to explicitly prevent text flowing alongside a floated content box
- Overflowing content is **visible** by default but the **overflow** property can accept **hidden** or **scroll** keywords to specify alternative behavior

- The **clip** property accepts the **rect()** keyword that specifies the coordinates of a clipping area within its parentheses
- Web page authors are now discouraged from using HTML tables for layout as CSS provides a superior layout method