Flow of HTML

A browser will render the elements of an HTML document that has no CSS from left to right, top to bottom, in the same order as they exist in the document. This is called the *flow* of elements in HTML.

In addition to the properties that it provides to style HTML elements, CSS includes properties that change how a browser *positions* elements. These properties specify where an element is located on a page, if the element can share lines with other elements, and other related attributes.

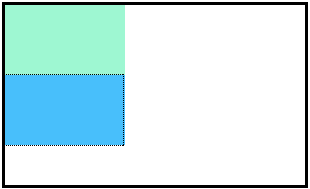
In this lesson, you will learn five properties for adjusting the position of HTML elements in the browser:

* position
* display
* z-index
* float
* clear

Each of these properties will allow us to position and view elements on a web page. They can be used in conjunction with any other styling properties you may know.

Position

Take a look at the *block-level* elements in the image below:



Block-level elements like these boxes create a *block* the full width of their parent elements, and they prevent other elements from appearing in the same horizontal space. The boxes in the image above were created with the following CSS:

.boxes {  
  width: 120px;  
  height: 70px;  
}

and the following HTML:

<div class="boxes"></div>  
<div class="boxes"></div>

Notice the block-level elements in the image above take up their own line of space and therefore don’t overlap each other. In the browser to the right you can see block-level elements also consistently appear on the left side of the browser. This is the default *position* for block-level elements.

The default position of an element can be changed by setting its position property. The position property can take one of four values:

1. static - the default value (it does not need to be specified)
2. relative
3. absolute
4. fixed

In the next few exercises, you’ll learn about the values in items 2, 3, and 4 above. For now, it’s important to understand that if you favor the default position of an HTML element, you don’t need to set its position property.

Position: Relative

One way to modify the default position of an element is by setting its position property to relative.

This value allows you to position an element *relative* to its default static position on the web page.

.box-bottom {  
  background-color: DeepSkyBlue;  
  position: relative;  
}

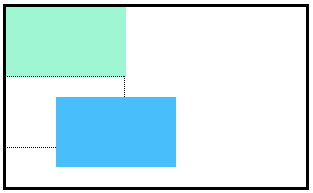
Although the code in the example above instructs the browser to expect a relative positioning of the div, it does not specify where the div should be positioned on the page.

.box-bottom {  
  background-color: DeepSkyBlue;  
  position: relative;  
  top: 20px;  
  left: 50px;  
}

In the example above, the <div> has been positioned using two of the four *offset properties*. The valid offset properties are:

1. top - moves the element down.
2. bottom - moves the element up.
3. left - moves the element right.
4. right - moves the element left.

In the example above, the <div> will be moved down 20 pixels and to the right 50 pixels from its default static position. The image below displays the new position of the box. The dotted line represents where the statically positioned (default) box was positioned.



Units for offset properties can be specified in pixels, ems, or percentages. Note that offset properties will not work if the value of the element’s position property is the default static.

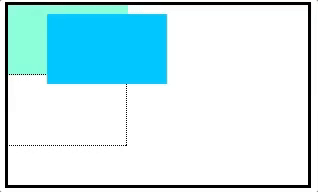
Position: Absolute

Another way of modifying the position of an element is by setting its position to absolute.

When an element’s position is set to absolute all other elements on the page will *ignore* the element and act like it is not present on the page. The element will be positioned relative to its closest positioned parent element. The parent has to be positioned relative otherwise the child element will default to the body.

.box-bottom {  
  background-color: DeepSkyBlue;  
  position: absolute;  
  top: 20px;  
  left: 50px;  
}

In the example above, the .box-bottom <div> will be moved down and right from the top left corner of the view. If offset properties weren’t specified, the top box would be entirely covered by the bottom box. Take a look at the gif below:



The bottom box in this image (colored blue) is displaced from the top left corner of its container. It is 20 pixels lower and 50 pixels to the right of the top box.

In the next exercise, we will compare the scrolling of absolute elements with fixed elements.

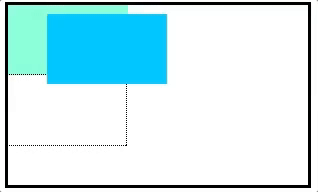
Position: Fixed

When an element’s position is set to absolute, as in the last exercise, the element will scroll with the rest of the document when a user scrolls.

We can *fix* an element to a specific position on the page (regardless of user scrolling) by setting its position to fixed.

.box-bottom {  
  background-color: DeepSkyBlue;  
  position: fixed;  
  top: 20px;  
  left: 50px;  
}

In the example above, the .box-bottom <div> will remain fixed to its position no matter where the user scrolls on the page, like in the image below:



This technique is often used for navigation bars on a web page.

Z-Index

When boxes on a web page have a combination of different positions, the boxes (and therefore, their content) can overlap with each other, making the content difficult to read or consume.

.box-top {  
  background-color: Aquamarine;  
}  
   
.box-bottom {  
  background-color: DeepSkyBlue;  
  position: absolute;  
  top: 20px;  
  left: 50px;  
}

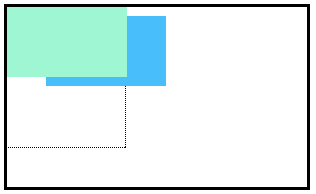
In the example above, the .box-bottom <div> ignores the .box-top <div> and overlaps it as a user scrolls.

The z-index property controls how far “back” or how far “forward” an element should appear on the web page when elements overlap. This can be thought of the *depth* of elements, with deeper elements appearing behind shallower elements.

The z-index property accepts integer values. Depending on their values, the integers instruct the browser on the order in which elements should be displayed on the web page.

.box-top {  
  background-color: Aquamarine;  
  position: relative;  
  z-index: 2;  
}  
   
.box-bottom {  
  background-color: DeepSkyBlue;  
  position: absolute;  
  top: 20px;  
  left: 50px;  
  z-index: 1;  
}

In the example above, we set the .box-top position to relative and the z-index to 2. We changed position to relative, because the z-index property does *not* work on static elements. The z-index of 2 moves the .box-top element forward, because it is greater than the .box-bottom z-index, 1. See the example image below:



In the image above, you can see the top box is moved in front of the bottom box.

Inline Display

Every HTML element has a default display value that dictates if it can share horizontal space with other elements. Some elements fill the entire browser from left to right regardless of the size of their content. Other elements only take up as much horizontal space as their content requires and can be directly next to other elements.

In this lesson, we’ll cover three values for the display property: inline, block, and inline-block.

The default display for some tags, such as <em>, <strong>, and <a>, is called *inline*. Inline elements have a box that wraps tightly around their content, only taking up the amount of space necessary to display their content and not requiring a new line after each element. The height and width of these elements cannot be specified in the CSS document. For example, the text of an anchor tag (<a>) will, by default, be displayed on the same line as the surrounding text, and it will only be as wide as necessary to contain its content. inline elements cannot be altered in size with the height or width CSS properties.

To learn more about <em>inline</em> elements, read <a href="#">MDN documentation</a>.

In the example above, the <em> element is inline, because it displays its content on the same line as the content surrounding it, including the anchor tag. This example will display:

To learn more about *inline* elements, read [MDN documentation](https://developer.mozilla.org/en-US/docs/Web/HTML/Inline_elements).

The CSS display property provides the ability to make any element an inline element. This includes elements that are not inline by default such as paragraphs, divs, and headings.

h1 {  
  display: inline;  
}

The CSS in the example above will change the display of all <h1> elements to inline. The browser will render <h1> elements on the same line as other inline elements immediately before or after them (if there are any).

Block Display

Some elements are not displayed in the same line as the content around them. These are called *block-level* elements. These elements fill the entire width of the page by default, but their width property can also be set. Unless otherwise specified, they are the height necessary to accommodate their content.

Elements that are block-level by default include all levels of heading elements (<h1> through <h6>), <p>, <div> and <footer>. For a complete list of block level elements, visit [the MDN documentation](https://developer.mozilla.org/en-US/docs/Web/HTML/Block-level_elements).

strong {  
  display: block;  
}

In the example above, all <strong> elements will be displayed on their own line, with no content directly on either side of them even though their contents may not fill the width of most computer screens.

The third value for the display property is inline-block. Inline-block display combines features of both inline and block elements. Inline-block elements can appear next to each other and we can specify their dimensions using the width and height properties. Images are the best example of default inline-block elements.

For example, <div>s in the CSS below will be displayed on the same line and with the specified dimensions:

<div class="rectangle">  
  <p>I’m a rectangle!</p>  
</div>  
<div class="rectangle">  
  <p>So am I!</p>  
</div>  
<div class="rectangle">  
  <p>Me three!</p>  
</div>

.rectangle {  
  display: inline-block;  
  width: 200px;  
  height: 300px;  
}

In the example above, there are three rectangular divs that each contain a paragraph of text. The .rectangle <div>s will all appear inline (provided there is enough space from left to right) with a width of 200 pixels and height of 300 pixels, even though the text inside of them may not require 200 pixels by 300 pixels of space.

Float

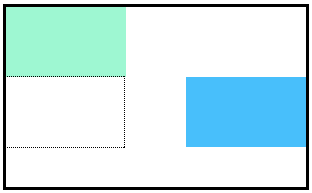
So far, you’ve learned how to specify the exact position of an element using offset properties. If you’re simply interested in moving an element as far left or as far right as possible on the page, you can use the float property.

The float property can be set to one of two values:

1. left - this value will move, or float, elements as far left as possible.
2. right - this value will move elements as far right as possible.

.boxes {  
  width: 120px;  
  height: 70px;  
}  
   
.box-bottom {  
  background-color: DeepSkyBlue;  
  float: right;  
}

In the example above, we float the .box-bottom element to the right. This works for static and relative positioned elements. See the result of the code below:



Floated elements must have a width specified, as in the example above. Otherwise, the element will assume the full width of its containing element, and changing the float value will not yield any visible results.

Clear

The float property can also be used to float multiple elements at once. However, when multiple floated elements have different heights, it can affect their layout on the page. Specifically, elements can “bump” into each other and not allow other elements to properly move to the left or right.

The clear property specifies how elements should behave when they bump into each other on the page. It can take on one of the following values:

1. left — the left side of the element will not touch any other element within the same containing element.
2. right — the right side of the element will not touch any other element within the same containing element.
3. both — neither side of the element will touch any other element within the same containing element.
4. none — the element can touch either side.

div {  
  width: 200px;  
  float: left;  
}  
   
div.special {  
  clear: left;  
}

In the example above, all <div>s on the page are floated to the left side. The element with class special did not move all the way to the left because a taller <div> blocked its positioning. By setting its clear property to left, the special <div> will be moved all the way to the left side of the page.

Review: Layout

Great job! In this lesson, you learned how to control the positioning of elements on a web page.

Let’s review what you’ve learned so far:

1. The position property allows you to specify the position of an element in three different ways.
2. When set to relative, an element’s position is relative to its default position on the page.
3. When set to absolute, an element’s position is relative to its closest positioned parent element. It can be pinned to any part of the web page, but the element will still move with the rest of the document when the page is scrolled.
4. When set to fixed, an element’s position can be pinned to any part of the web page. The element will remain in view no matter what.
5. The z-index of an element specifies how far back or how far forward an element appears on the page when it overlaps other elements.
6. The display property allows you control how an element flows vertically and horizontally a document.
7. inline elements take up as little space as possible, and they cannot have manually-adjusted width or height.
8. block elements take up the width of their container and can have manually-adjusted heights.
9. inline-block elements can have set width and height, but they can also appear next to each other and do not take up their entire container width.
10. The float property can move elements as far left or as far right as possible on a web page.
11. You can clear an element’s left or right side (or both) using the clear property.

When combined with an understanding of the box model, positioning can create visually appealing web pages. So far, we’ve focused on adding content in the form of text to a web page. In the next unit, you’ll learn how to add and manipulate images to a web page.