Layout

**Page Layout**

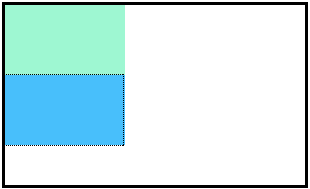
The box model is the first step in understanding how the browser lays out HTML elements. Visually appealing websites, however, are often the result of well positioned elements.

In this unit, you'll learn how to position HTML elements in order to gain more control of a web page's layout.

Let's begin!

**Position**

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



The boxes in the image above were created with the following CSS:

.boxes {

width: 120px;

height: 70px;

}

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;

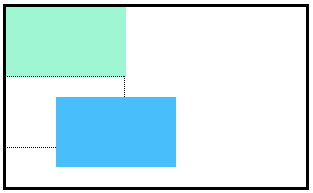
left: 50px;

}

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 position of the element is not set to relative.

# 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.

.box-bottom {

background-color: DeepSkyBlue;

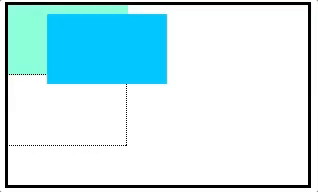
position: absolute;

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 out of view 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;

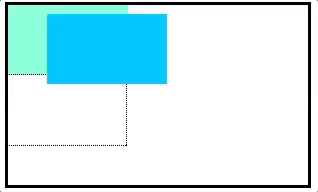
position: fixed;

top: 20px;

left: 50px;

}

In the example above, the 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.

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;

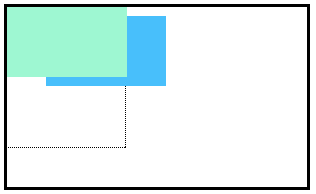
top: 20px;

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.

**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;

}

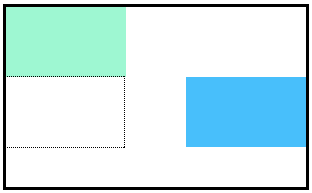
.box-bottom {

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 divs on the page are floated to the left side. The div 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.