Figure 57: `background-size` can optionally take *x-axis* and *y-axis* parameters separated by space character.

Figure 58: `cover` and `contain`.

Figure 59: By combining `background-repeat: no-repeat;` with `background-size: 100%` it is possible to stretch the image only horizontally, across the entire width of the element.

What if you want to repeat background vertically but keep it stretched across the width? No problem, simply remove `no-repeat` from previous example. This is what you will end up with:

Figure 60: This is really useful if you need to blow up a large image across the screen without sacrificing vertical repeat.

Sometimes it is needed to stretch the image across to fit the bounding box of an element. This often comes at a price of some distortion, however:

Figure 61: `background-size: 100% 100%`

Note here, 100% 100% is repeated twice. The first value tells CSS to "stretch the image vertically", the second 100% does the same horizontally. In HTML, whenever you need to specify multiple values, they are often separated by a space. Vertical coordinates always come first.

6.1.3 object-fit: fill — cover — contain — none

Cover and *contain* values from previous section can also be used on HTML elements such as images, videos and similar media. In this case, when `cover` and `contain` are used with `object-fit` CSS property, behavior of these elements follows rules demonstrated in the diagrams below:

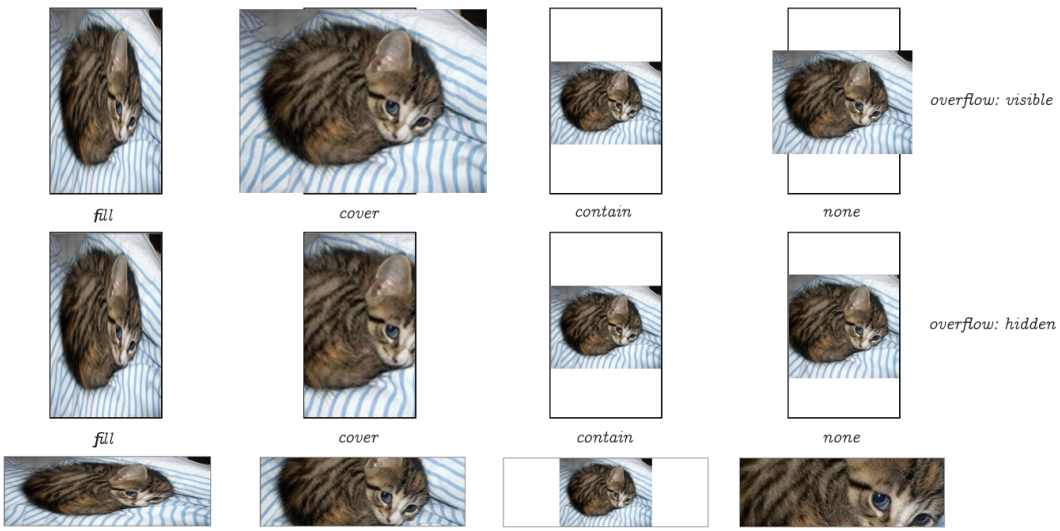


Figure 62: `fill`, `cover`, `contain` and `none` produce similar results to background-size property, except they work on HTML media elements, rather than background images.

6.1.4 background-position

Figure 63: Center on the screen without repeat can be achieved by combining `background-repeat: no-repeat` and `background-position: center center;`

In another scenario you can force the image to be always in the center and keep the repeat:

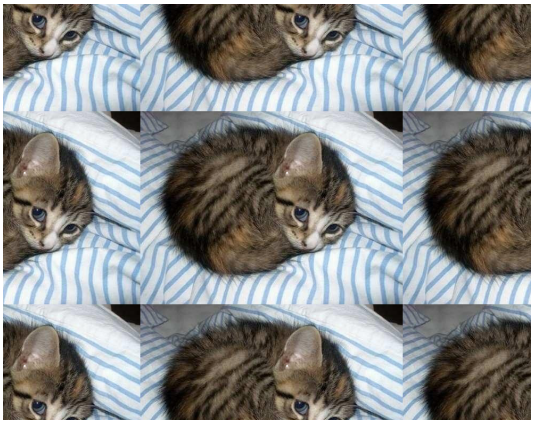


Figure 64: `background-position: center center;` with `background-repeat: repeat;`

6.1.5 repeat-x

You can repeat the image across the x-axis only (horizontally) using `repeat-x`:

Figure 65: CSS style: `background-position: center center; background-repeat: repeat-x;`

6.1.6 repeat-y

To the same effect but on the y-axis `repeat-y` property can be used:

Figure 66: The `background-repeat` property set to `repeat-y`

Like any other CSS property, you have to juggle around the values to achieve the results you want. I think we covered pretty much everything there is about backgrounds. Except one last thing...

6.1.7 Multiple Backgrounds

It is possible to add more than one background to the same HTML element. The process is rather simple.

Consider these images stored in two separate files:

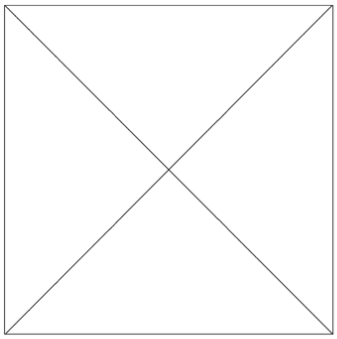


image1.png

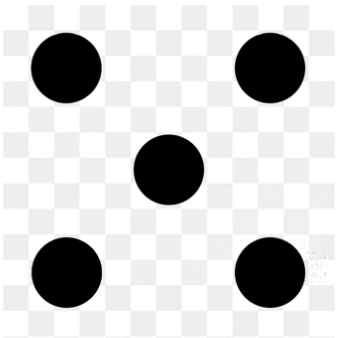


image2.png

Figure 67: Two images that will be used for creating multiple backgrounds.

The chessboard pattern in the image on the right is only used to indicate *transparency* here. The white and grayish squares are not an actual part of the image itself. This is the "see-through" area, which you would usually see in digital manipulation software.

When the image on the right is placed on top of other HTML elements or images, the checkered area will not block that content underneath. And this is the whole idea behind multiple backgrounds in HTML.

6.1.8 Image Transparency

To fully take advantage of multiple backgrounds, one of the background images should have a transparent area. But how do we create one?

In this example, the second image (*image2.png*) contains 5 black dots on a transparent background (*indicated by a checkered pattern.*)

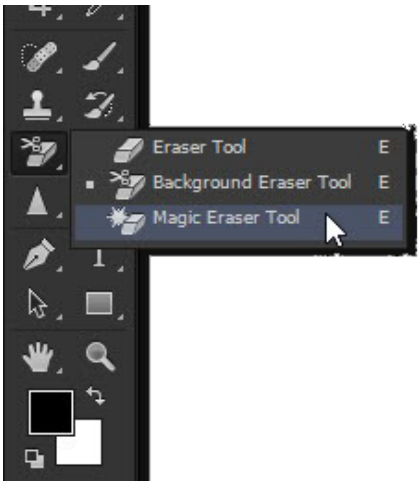


Figure 68: To create images with transparent backgrounds, tools such as Photoshop can be used. Just select the Magic Eraser tool from the toolbox.

Like many other CSS properties that accept multiple values – all you have to do – to set up multiple backgrounds is to provide a *set of values* to the `background` property separated by comma.

6.1.9 Specifying multiple background images

To assign multiple (layered) background images to the same HTML element, the following CSS can be used:

```
body { background: url('image2.png'), url('image1.png');
    }
```

The order in which you supply images to the background's url property is important. Note that the top-most image is always listed first. This is why we start with *image2.png*

This code produces the following result:

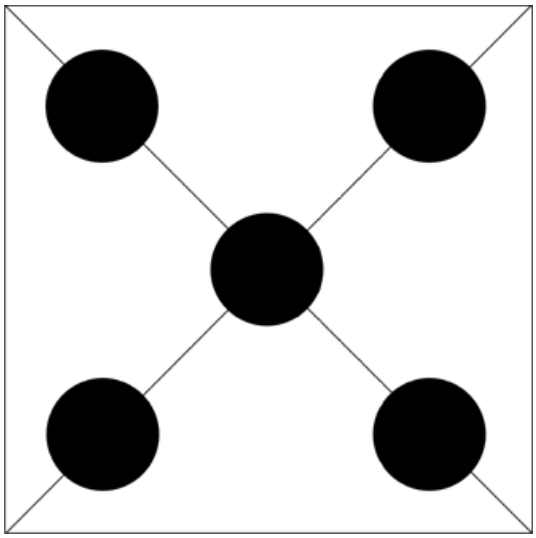
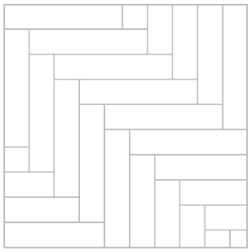


Figure 69: Multiple background in use. Here `image2.png` is superimposed on top of `image1.png`.

In this example we demonstrated multiple backgrounds in theory on a subjective `<div>` (or similar) element with square dimensions. Let's take a look at another example:

puppy.png



pattern.png

Figure 70: A puppy and a linoleum-like pattern.

Note here again, that the puppy.png image will be the first item on the comma-separated list. This is the image we want to *superimpose* on top of all of the other images on the list.

Combining the two:

```
body { background: url('puppy.png'), url('pattern.png');
    }
```

We get the following result:

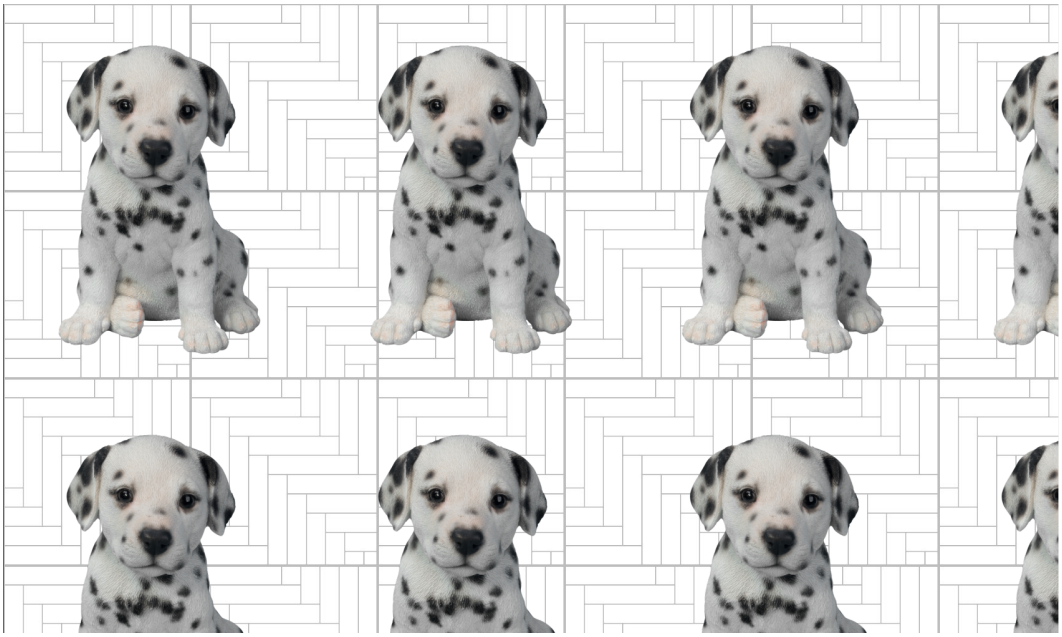


Figure 71: Multiple backgrounds.

6.1.10 Other background properties that take comma-separated lists

In the same way, you can supply other parameters to each individual background, using the other background properties demonstrated below:

```
background
background-attachment
background-clip
background-image
background-origin
background-position
background-repeat
background-size
```

The following property cannot be used with a list, for obvious reasons:

```
background-color
```

6.2 Images As HTML Elements

When I started working with HTML images, for some reason I always thought that they were completely separate entities from your standard HTML elements such as `<div>` for example. Perhaps this is because images by nature are so much more impressive than text content.

The *Morris Marina* is a car that was manufactured by Austin-Morris division of British Leyland from 1971 until 1980.

Figure 72: A simple image `<img = "auto.png" alt = "Morris Marina">` specimen that we will use to demonstrate working with images in HTML in this chapter.

I was wrong. The image element in HTML that can be used as `<image>` or `<img>` is just like any other HTML element. The only difference is the content area, which now contains an image, instead of text.

To prove that, here is an image with border, padding and margin set to some arbitrary values:

Figure 73: An image is just like any other blocking element containing *border*, *padding*, and *margin*. To be precise, an image is actually an inline *and* blocking element at the same time. In other words `display: inline-block`.

If you set any HTML element's `display` property to `inline-block` that element will behave exactly like an HTML image, even if it contains some other type of content inside.

One of the most common placement for images is in the middle of the page.

Figure 74: An image with `img {margin: auto}` margin set to *auto*.

Automatic margins will ensure that the image is aligned to the center of the page, and if there is any text surrounding the image it will appear above or

below it, based on where it is located in your HTML document.

Figure 75: Margin property can be used to position an image within its parent.

The example above assumes that the image's `display` property is set to `block` and its position is set to `relative`.

Image margins are often used to bump text away from the image to create more white space, making the surrounding text easier to read.

But images can be also positioned using `display: block` and `position: absolute` combo. You just need to provide additional values for `top` and `left` properties to define its precise location on the screen *relative to its parent element*.

The above example can be interpreted as though the image's top/left properties were set to `top: 100px` and `left: 200px` to position it within parent element at exactly the same spot without having to do it via the margin property. Which is often the preferred way of doing it.

Note: In order for any HTML element to be accurately positioned within its parent container element using absolute pixel location the *parent element* is required to set its `display` property to a value of either `absolute` or `relative`. If you fail to do this, the behavior of the child element is unpredictable. But it will be most likely positioned relative to the root parent element such as `<body>`.

11 Chapter XII: HTML Elements – Common Properties

Every HTML element, *blocking* or *inline* has structural composition that may not be obvious at first – because all properties that deal with size are set to 0 by default.

11.0.1 Anatomy of HTML Elements

In reality, an HTML element consists of *content area*, *padding*, a *border* and a *margin*.

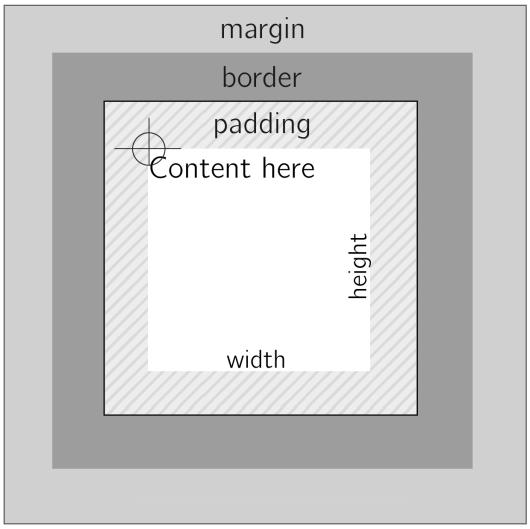


Figure 118: HTML Element Anatomy – structural composition of an HTML element.

Note that the X and Y location of an element (determined by `top` and `left` CSS properties) refers to the upper left corner of the *content area* when padding, border and margin are set to 0.

Changing *padding*, *border* or *margin* to something greater than its default value of 0 will not automatically change neither its location nor its dimensions

(***top*** *and* ***left***, ***width*** *and* ***height*** *values of the element.*) even though the element will noticeably be located at a different location on the screen.

How can we then determine the actual width and height of the element in these cases? You can use JavaScript library such as jQuery that provides methods that calculate these values for you. But this is outside of the scope of this book.

Increasing padding value to the element will increase its blocking width, but not the width of its content area. So at times it's difficult to determine as to what should be the element's actual width.

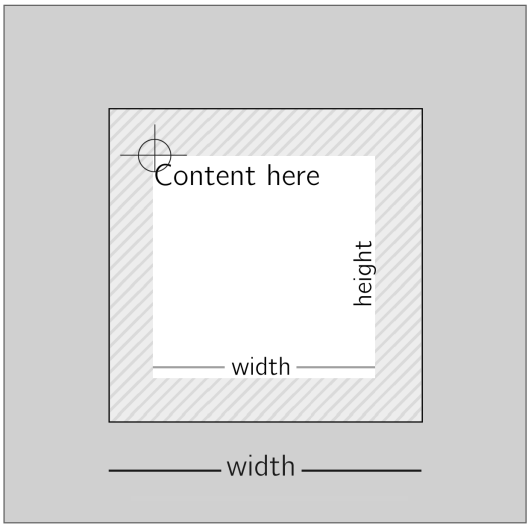


Figure 119:

However, you can retain the padding and suppress this from happening by setting your element's `box-sizing` property to a value of `border-box`:

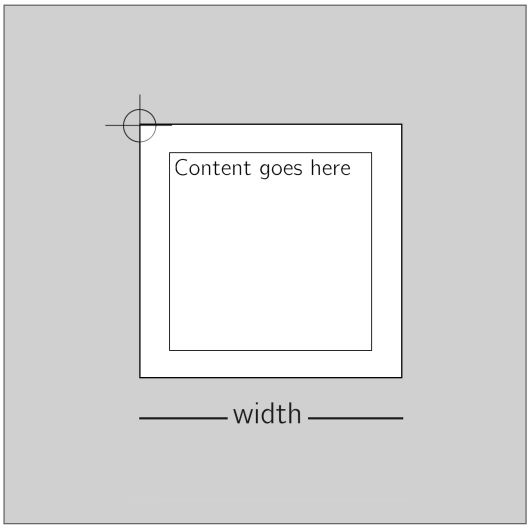


Figure 120: `box-sizing: border-box;`
Notice that the padding now occupies the *inner* area of the element. Now width and height of the element are their original values, and the content is still padded within the element.

Another common thing to do to HTML elements is to max out their rounded corners properties to create a circle using `border-radius` property:

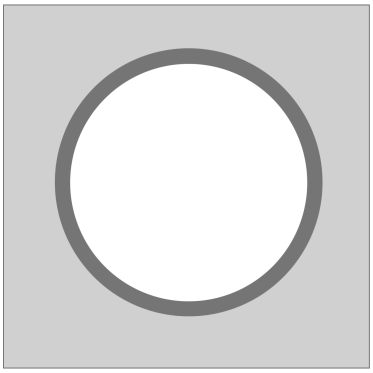


Figure 121: `border-radius: 500px;` (*or set it to actual width of the square element*)

To expand on the previous example, a shadow can be added to create an even more dramatic effect:

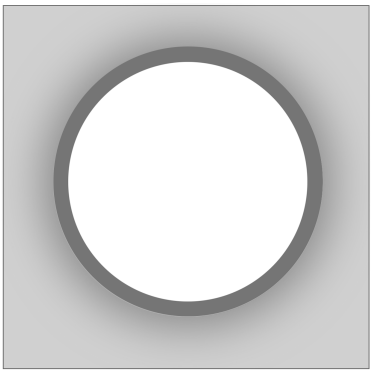


Figure 122: `box-shadow: 0 0 10px #0000`

The values of the box-shadow property are explained below:

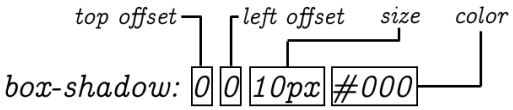


Figure 123: Four values supplied to `box-shadow` property separated by space. Here color black (*hexadecimal #000*) is used to provide the base shadow color.

These techniques are often used for decorating HTML buttons:

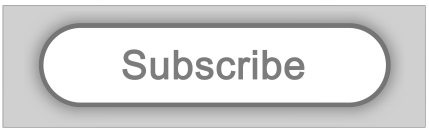


Figure 124: Rounded corners and a shadow used to create a Subscribe button with centered text.

Box shadow can be used in creative ways to add *double* (or even *triple*) border effect.

11.0.2 Visibility

Visibility of an HTML element is controlled by `visibility` property.

Possible values are `visible` (*default*) and `hidden`.

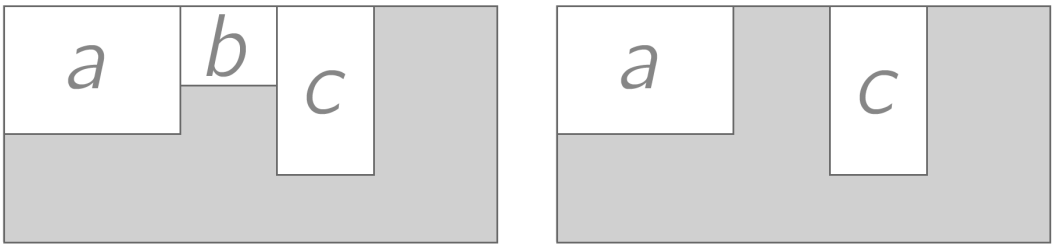


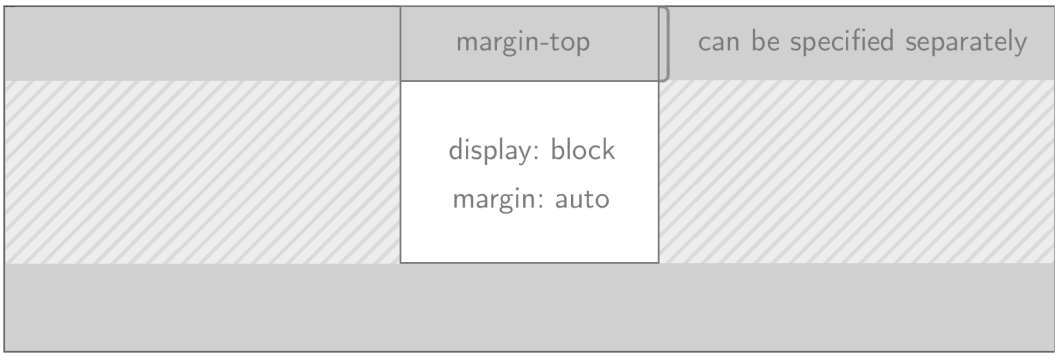
Figure 126: **Left:** Three elements with `visibility:visible` (*default*). **Right:** The same three elements, with element *b*'s `visibility` set to `hidden`.

Setting visibility property to `hidden` doesn't actually remove the element from the document. The browser simply skips drawing it. But structurally, it is still part of the document:

11.0.3 Positioning

It is very common to want to position an element exactly at the middle of the screen. This technique is often used to create the main container for the entire website. It's convenient because even if the browser is resized, the element remains automatically aligned to the center.

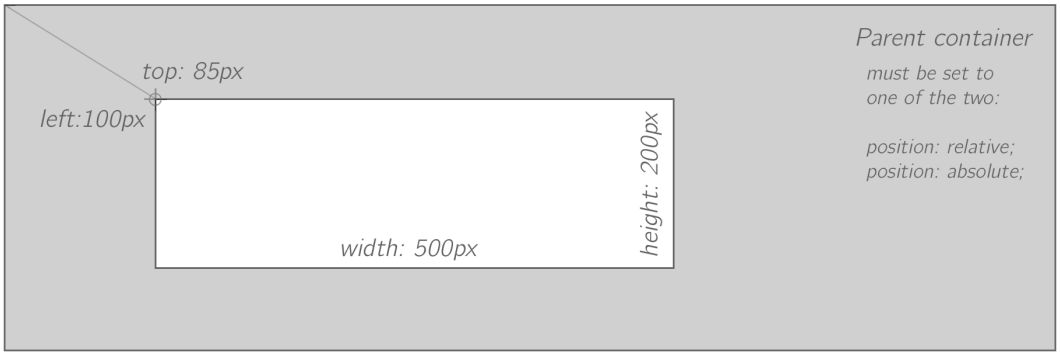
One way of accomplishing that would be to simply add *automatic margins* to an HTML element whose `display` property is set to `block`:



Align an element to the center of the screen using automatic margins

Figure 127: `display: block; margin: auto; width: 500px;`

To place an element at an exact location, relative to its parent element `position: absolute` property can be used. It also requires supplying `display: block` and the actual placement in pixels using `top` and `left` properties:



Using absolute placement with position:absolute property

Figure 128: `display:block; position:absolute; top:85px; left:100px; width:500px; height:200px;`

One of my favorite features of CSS when it comes to *absolute* positioning of elements is the ability to define the origin of location, based on any of the four corners of an element.

This is useful when you want to create a placeholder for pop-up notification messages that can potentially appear in any corner of the screen:

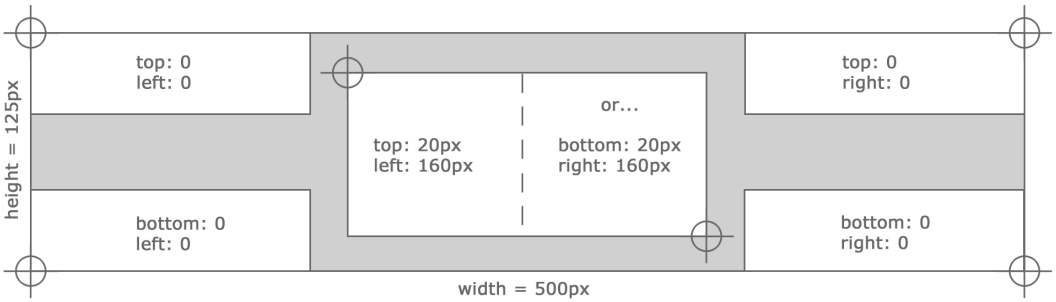


Figure 129: You can use `left`, `right`, `top` and `bottom` properties to change origin of location. But you cannot use `left` together with `right`, or `top` together with bottom on the same element... for obvious reasons.

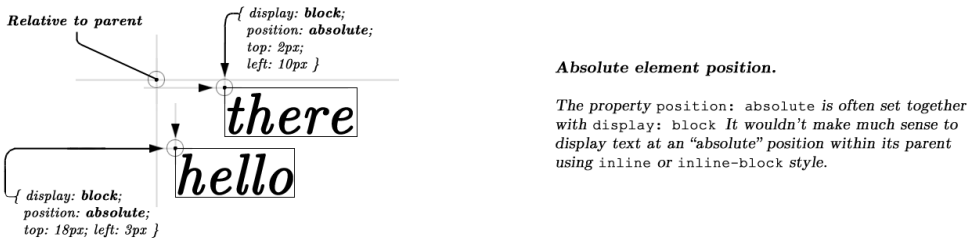


Figure 130: Absolute position applied to HTML elements containing text.

To fix your element on the screen relative to the browser view (*regardless if the horizontal scroll bar changes location*) is often used for creating "overlay" effect. The idea behind it is simple:

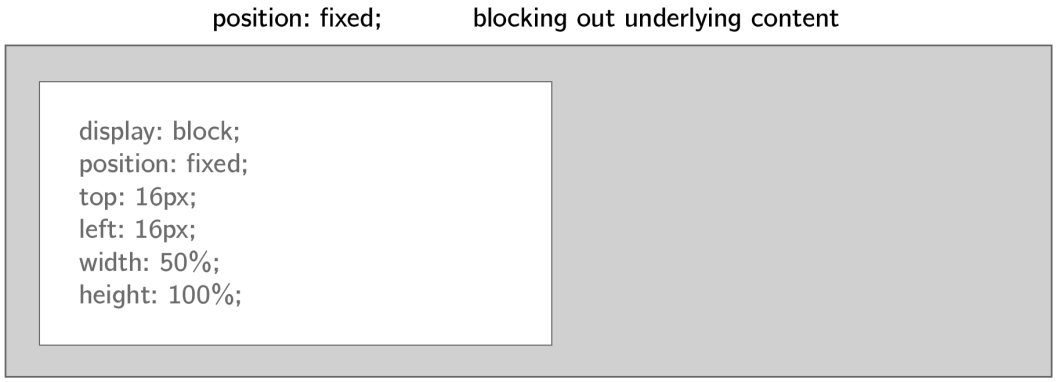


Figure 131: `display:block; position:fixed; top:16px; left:16px; width:50%; height:100%;`

Elements "fixed" to the screen can be used to "gray out" the background when you need to display a custom modal dialog box as depicted on the following diagram:

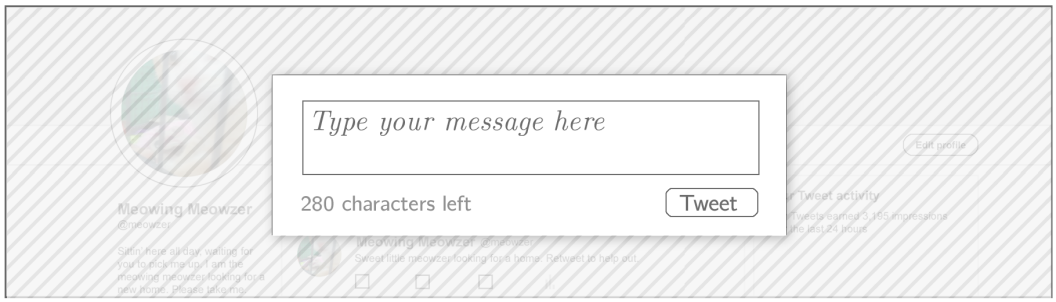


Figure 132: When you post a tweet, the background is shaded out by a full screen element with 50% opacity. The primary user interface controls are then displayed on top of that layer.

11.0.4 Floating Elements

Elements that use relative and absolute position are great for making our life easier when creating various layouts. But there comes a time when you need to "float" an element to make room for other content. Floating elements usually shift to either left or right side, opening up more room for elements specified right after them in your HTML code.

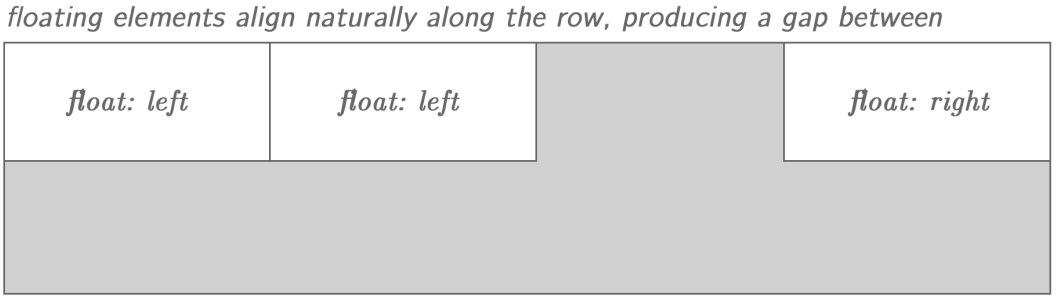


Figure 133: Two elements with `float:left` behave similar to standard inline text. Another element with `float:right` was added here to demonstrate that content doesn't have to leave the same horizontal area, at the expense of creating a gap if the combined width of all floating elements is shorter than the parent.

11.1 Element Modifications

11.1.1 Rounded Corners

Creating rounded corners by changing border-radius property

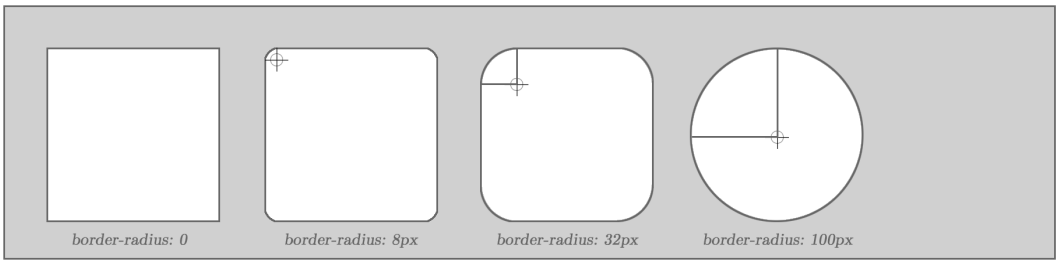


Figure 136: Round corners in a nutshell.

Rounded corners are used to create buttons:

Creating a custom button

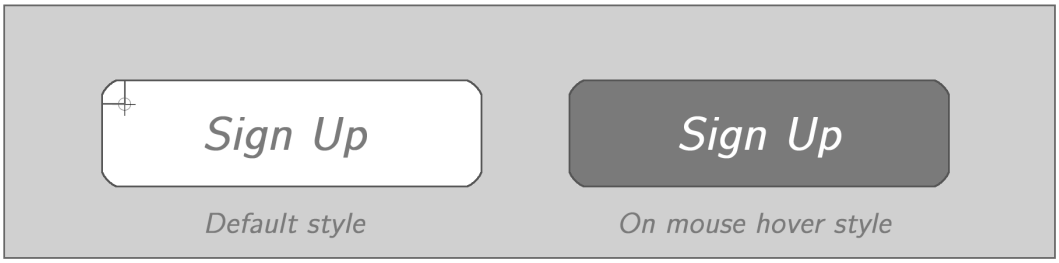


Figure 137: Buttons with rounded corners.

11.1.2 Z-Index

Let's say we mastered HTML element placement on the screen in two dimensions (X and Y.) Still you will encounter cases where you need to pick out some elements to appear "on top" of the others. Even if this isn't how they were listed in your HTML code structure.

The `z-index` property to the rescue. You can bring out any element whose `display` property was set to value of `absolute` and ensure that it is always

displayed "above" the rest. In other words, on the Z-axis.

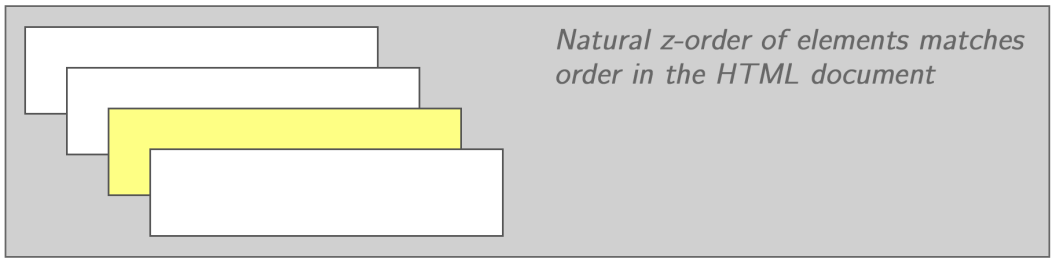


Figure 138: Natural order of elements in HTML document.

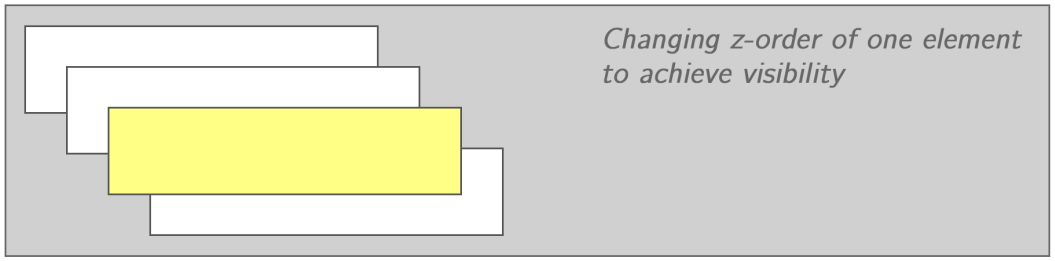


Figure 139: The `z-index` property can bring out elements to the front.

11.1.3 box-shadow

We already covered box shadow earlier. But here are a few more examples.

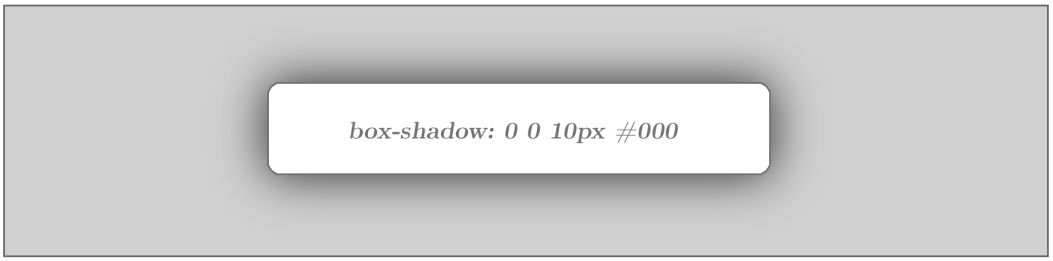


Figure 140: Shadow is centered at the element.

Figure 141: Shadow is displaced by a few pixels from the element to create a *drop-shadow* effect.

And finally... by specifying light colors you can turn a shadow into this glowing effect:

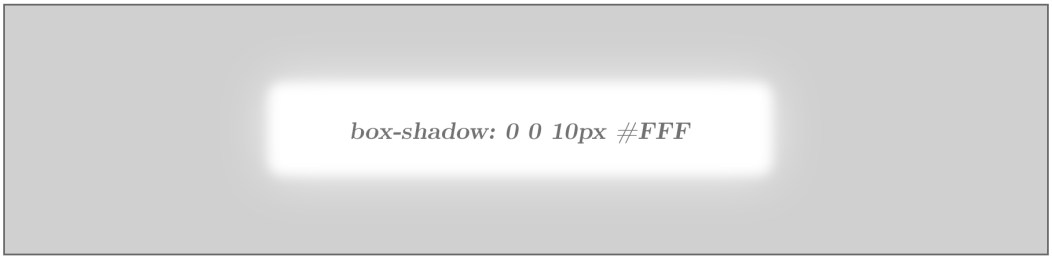


Figure 142: Glowing effect can be simulated by using bright colors with `box-shadow` property.

14.1 Clock

By the end of this section we will create this simple clock:

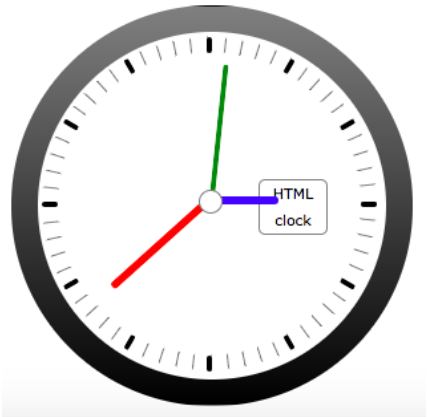


Figure 166: The notches and clock hands are actually HTML elements positioned using combination of techniques covered in this book.

No images were used in creation of this clock. Everything is a `DIV` element. Here is the same clock with `border: 1px solid black;` and `background: white;` properties that expose its wireframe structure:

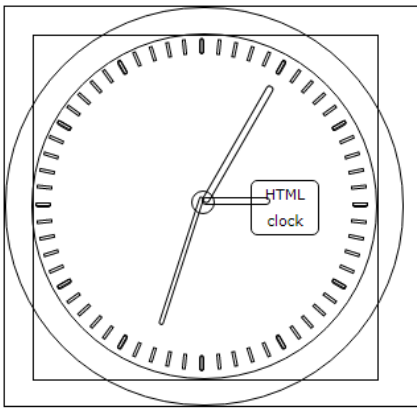


Figure 167: Wireframe view of the HTML clock showing transparent elements.

14.2 Calculator

In our previous example we created an animated clock. But it's not really an *interactive* application. Meaning, it does not offer the visitor a chance to interact with it or take some sort of input.

In this chapter, let's create another simple application that takes basic input from the user. This calculator application should be interesting-enough to demonstrate user input without having to deal with too much complexity.

First, let's take a look at what we're actually trying to build here:

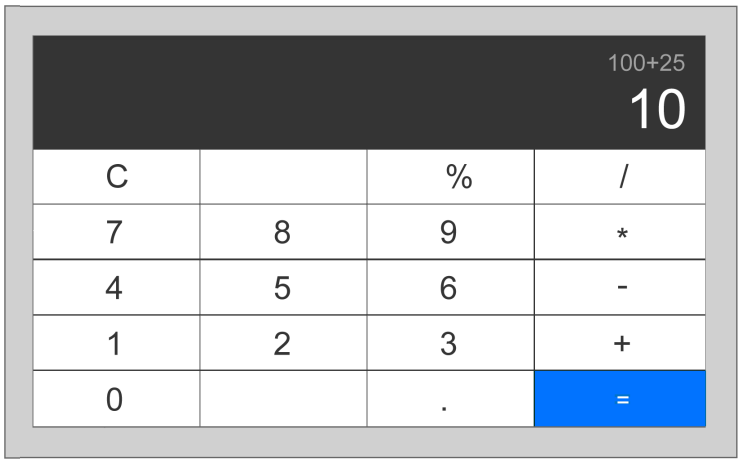


Figure 168: Calculator application we will build in this section of the book.

Let's begin by building out the HTML scaffold!

14.2.1 HTML – Application Scaffold

The HTML here is the simplest part of the entire application. We just need to add the view and some buttons. Note that no id's are necessary for most of the buttons. This is because our JavaScript code will read the values directly from the element's content from the `event.target` object via the `onClick` event.