7 Harnessing the full power of float

This chapter covers

- Creating a drop cap using float
- Using float to wrap text around the image
- Using CSS shapes to make the text follow the floated image's shape

Grid and Flexbox have given us the ability to create layouts that once were incredibly difficult to realize, if not impossible. One of the most common examples is a three-column layout with all three columns the same height regardless of the contents. Another layout technique, which unlike its grid and flexbox counterparts has been around for quite some time, is float. Part of the CSS Logical Properties and Values Module, *float* is purpose-built to allow other content to wrap around the element being floated; as a result, it shines at manipulating images inside text and creating drop caps.

Drop caps are a way to style and add emphasis to text. They consist of creating a larger (sometimes more ornate) capital letter, usually at the beginning of a page or paragraph. Drop caps were often used in the illuminated manuscripts of the Middle Ages. The *F* at the beginning of the paragraph in figure 7.1 is an example of a drop cap in the Carmina Burana manuscript. Later, with the advent of the printing press, the concept carried over into print; printers created specialized glyphs and plates or simply used a larger font size. Drop caps are much rarer on the web, but they're by no means impossible to create, and they're a great way to make our online typography more interesting.

Figure 7.1 Drop cap at the beginning of the paragraph in the Carmina Burana manuscript

Another way to make content more visually striking is to style our images to fit nicely in the text. When we add images to content, we often add our image element and maybe some margin, and don't think about the process much more. Using CSS shapes in conjunction with float, however, we can make our text wrap in the actual shape of the image to create a much more striking effect. We can flow text around virtually any shapes we create, even curves.

In this chapter, we'll take a close look at our typography and images to make our content more visually interesting while making sure to keep it accessible. We'll start with an unstyled excerpt from *The Call of the Wild*, by Jack London (**http://mng.bz/61WR**). We'll use float to add a drop cap to our first paragraph. Then we'll wrap our text around our images (both raster and vector), following the content of those images. Figure 7.2 shows the starting point and the finished product.

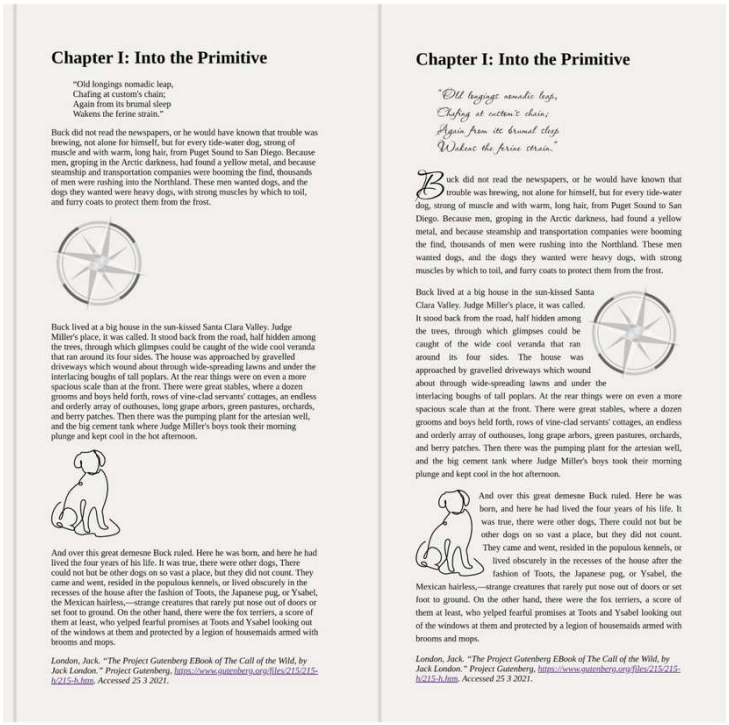


Figure 7.2 The starting point (left) and finished product (right)

NOTE A *raster image* is created by using a grid of pixels, whereas a *vector image* is drawn with the help of mathematical formulas. For in-depth information about the difference between rasters and vectors, check out chapter 3.

Listing 7.1 and listing 7.2 contain the starting HTML and CSS, respectively, for the page we'll build on in this chapter. To follow along as we style the page, you can download the starting code from the GitHub repository at **http://mng.bz/oJXD** or from CodePen at **https://codepen.io/michaelgearon/pen/MWodXxM**. Our HTML consists of a `<main>` element inside which we have a header (`<h1>`), block quote (`<blockquote>`), three paragraphs (`<p>`), two images (`<img>`), and the source citation (`<cite>`).

Listing 7.1 Starting HTML

```
<main>
  <h1>Chapter I: Into the Primitive</h1>
  <blockquote>"Old longings nomadic...</blockquote>
  <p>Buck did not read the newspapers, or he...</p>
  <img class="compass" src="./img/compass.png"                    ①
       width="175" height="175" alt="a black and gray compass">   ①
  <p>Buck lived at a big house in the...</p>
  <img class="dog" src="./img/dog.svg"                            ②
       width="126" alt="line drawing of a dog">                   ②
  <p>And over this great demesne Buck ruled...</p>
  <cite>London, Jack...</cite>
 </main>
```

① Compass image

② Dog image

Our CSS includes some base styles to set up our page, including `margin` , `padding` , and `background-color` . The `body` 's width is restricted

to `78ch`, and margins center the content when the screen width exceeds our maximum value. We also set up the default font for the page, which is Times New Roman. Last, to ensure that the images don't overflow on small screens, we give them a maximum width, which is set to `100%`. In other words, the images can't be wider than their container.

NOTE Notice that we use `ch` for our `max-width`. `ch` is a relative unit based on the font family being used. `1ch` is equal to the width of—or, more precisely, the horizontal amount of space occupied by—the glyph `0` (zero).

Listing 7.2 Starting CSS

```
html {
  padding: 0;
  margin: 0;
}

body {
  background-color: rgba(206, 194, 174, 0.24);
  padding: 4rem;
  font-size: 16px;
  max-width: 78ch;                                    ①
  margin: 0 auto;                                     ②
  font-family: 'Times New Roman', Times, serif;
  border-left: double 5px rgba(0,0,0,.16);
  min-height: 100vh;                                  ③
  box-sizing: border-box;
}
img {
  max-width: 100%;
}
```

① Prevents our content from becoming excessively wide

② Centers the content

③ Regardless of window size, the background covers the whole window.

7.1 Adding a drop cap

We have some base CSS to style the page, so now we're going to turn our attention to the text. By virtue of the fact that the width of our body is capped at a width that works well for our text, we don't need to worry about line length. But we do need to address the leading.

7.1.1 Leading

Leading (pronounced 'le-diŋ) is the amount of space between lines. The term comes from the days of the printing press when compositors used lead bars of various widths to adjust the spacing between lines of text. The CSS property we're going to use to accomplish the same outcome is `line-height`. This property can take a number value (`line-height: 2`) or a number with a unit (`line-height: 5px`). The unit can be relative, such as ems, or fixed, such as pixels. Unless the unit is relative to the font size when we provide a unit (such as `em`), if the font is scaled or a child element has a different font size, the line height may not look correct and can negatively affect legibility. When

we use a unitless number, the line height is automatically calculated relative to the font size of the element, eliminating this concern. Therefore, we'll use a unitless `line-height`. We'll set a `line-height` of `1.5` on all paragraphs by creating a rule specifically for the paragraph element and then applying the height as follows: `p { line-height: 1.5; }`.

TIP Research shows that text with a `line-height` between `1.5` and `2` makes line tracking easier for people with cognitive disabilities (**https://www.w3.org/TR/WCAG20-TECHS/C21.html**).

7.1.2 Justification

For optimum effect when we have the text follow the image, we're going to justify our text. *Justifying* the text means we're going to make all our lines the same width—a technique often used in newspapers to make the right edge of a column on text straight rather than ragged.

WARNING The Web Content Accessibility Guideline (WCAG) includes three levels of conformance that build on one another: A, AA, and AAA. A is the least restrictive, and AAA is the most stringent. Most often, websites aim for an AA level of conformance. But if we're required to conform to AAA, it's worth mentioning that justifying text goes against accessibility guideline 1.4.5, which is a requirement for AAA (**http://mng.bz/v1ja**).

To justify our text, we're going to use the `text-align` property, which can take a value of `left`, `right`, `center`, or `justify`. We'll add `text-align: justify;` to our paragraph rule. Now that rule has two properties, `text-align` and `line-height`, that take care of styling the paragraph. The following listing shows the completed paragraph rule, and figure 7.3 shows the result.

Listing 7.3 Completed paragraph rule

```
p {
  line-height: 1.5;
  text-align: justify;
}
```

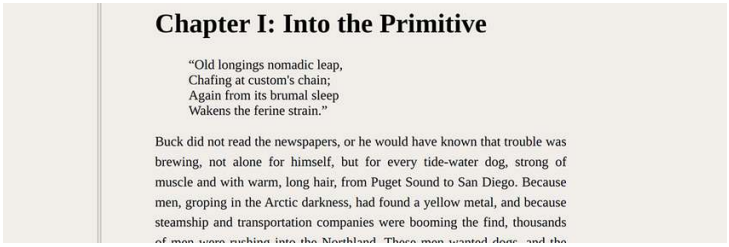


Figure 7.3 Styled paragraphs

With the paragraph taken care of, we can hone in on the first letter of the first paragraph to create our drop cap.

7.1.3 First letter

We don't need to add any elements to the HTML to select the first letter of our first paragraph. We can use the pseudo-class `:first-of-type` to

select the first paragraph and then the pseudo-element `::first-letter` to get to the letter, in this case a *B*, both of which can be chained. In code, these selections translate to `p:first-of-type::first-letter {}`.

NOTE A pseudo-class is added to a selector to target a specific state; a pseudo-element allows us to select part of the element.

With the letter selected, we can start styling it to make it look like a drop cap. To make it stand out from the rest of the text, we're going to pick a more ornate typeface. In this case, we'll import Passions Conflict (**http://mng.bz/X5vE**; figure 7.4) from Google Fonts.

Figure 7.4 Passions Conflict glyphs

Because this typeface has particularly ornate capital letters, it's well suited for use as our drop cap. We'll also use it later in this chapter to style the quote at the beginning of the text. Using a beautiful typeface such as this one is a wonderful way to embellish a page—but only for short bits of content. Handwriting and display fonts can be quite difficult to read, so they're not well suited for large blocks of text. For a drop cap, large header, or short quote, however, these fonts differentiate the element from the rest of the content and give the page some personality.

This particular font has glyphs that are quite a bit smaller than those of Times New Roman (the font we're using for the rest of our content). Because we're creating a drop cap, which by definition is larger than the rest of the text, we're going to have to adjust the font size. We're also going to adjust the line height of the letter to make it fit nicely with the text. Finally, we're going to float our first letter to the left so that the text flows around the letter, accomplishing our desired effect.

The `float` property places an element to the right or left of its container based on the value passed to it. According to the Mozilla Developer Network, "The element is removed from the normal flow of the page, though still remaining a part of the flow" (**http://mng.bz/ydle**). Inline elements around it (our text) use the leftover space to wrap around the floated element.

The `float` property can take one of three values: `left`, `right`, and `none` (element isn't floated). Because our text is in English, which flows from left to right, we want to keep the letter *B* to the left, so we're going to float the first letter of the first paragraph (*B*) to the left by adding `float: left;` to our rule. The following listing shows the completed CSS rule we create to style our drop cap, as well as the import of the Passions Conflict typeface.

Listing 7.4 Styling and positioning the first letter of the first paragraph

```
@import url(
  'https:/ /fonts.googleapis.com/css2?
➥ family=Passions+Conflict&display=swap'                    ①
);

p:first-of-type::first-letter {                              ②
  font-size: 6em;                                            ②
  float: left;                                               ②
  line-height: .5;                                           ②
  font-family: 'Passions Conflict', cursive;                 ②
}                                                            ②
```

① Import of the Passions Conflict typeface

② Rule that styles the letter B at the beginning of our first paragraph

Notice that we altered the line height of the first letter to adjust the space below the *B*. By default, line height is proportional to font size. Because our letter is large, the line height it requires is tall, so we decrease it to make the text flow more naturally below the drop cap. Figure 7.5 shows the output generated.

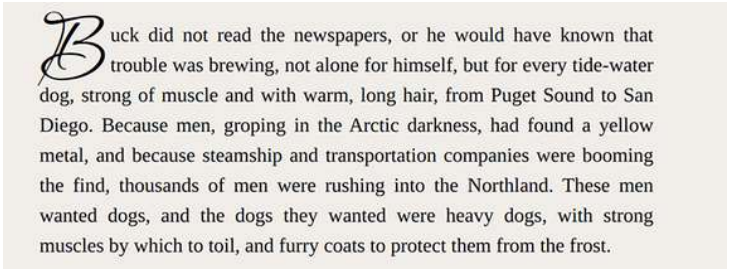


Figure 7.5 Drop cap

We use `em`s and a unitless `line-height` so that if we ever change the font size of the paragraph, the drop cap will scale accordingly. The value of `6em` is set based on the `font-size` of the parent element, which in this case is our paragraph tag.

To reposition our *B* to fit well with the text, we edited the `line-height` of the letter. But we could have used another technique. We could have set the `position` of the *B* to `relative` and then used `top`, `bottom`, `left`, and `right` to alter its position relative to the rest of the text. With our drop cap created, we're going to turn our attention to the quote at the start of the page.

7.2 Styling the quote

The quote at the top of the page is rather drab at the moment and gets a little lost in the rest of the text. To make it stand out, we're going to use the same font we used for our drop cap. Because of the previously mentioned differences in size and line height, we're going to adjust those parameters so that the paragraphs and the quote are uniformly sized and spaced. Listing 7.5 shows the CSS we'll add to accomplish this task, and figure 7.6 shows the output.

Listing 7.5 `<blockquote>` formatting

```
blockquote {
  font-family: 'Passions Conflict', cursive;
  font-size: 2em;
  line-height: 1;
}
```

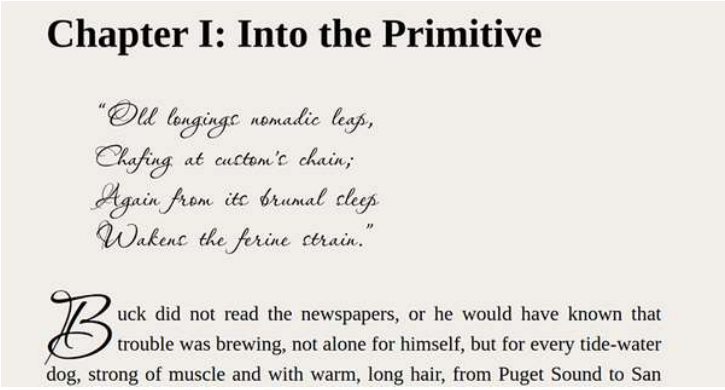


Figure 7.6 Styled `<blockquote>`

Again, we use relative units so that if the rest of the content's font size changes, so will the quote. You may have noticed that we used a `line-height` of `1` even though we stated earlier (section 7.1.1) that for optimal legibility, a line height of `1.5` to `2` is ideal. We make an exception here because by default the font already has a large line height; we don't need to increase the size. Occasionally, we'll encounter fonts that have naturally tall line heights by default, especially when we're dealing with cursive or display fonts. When this happens, sometimes we have to make an exception to the line-height-legibility guidance due to the design of the font.

Now, with our text taken care of, we can focus on the images.

7.3 Curving text around the compass

The first thing we need to do to make the text wrap around our compass image is float the compass to the right. Our compass is a PNG image, and because it's a rectangular image, the text follows a rectangular path in wrapping around the image. Figure 7.7 shows the floated compass. A border has been applied to the image to expose its bounding box.

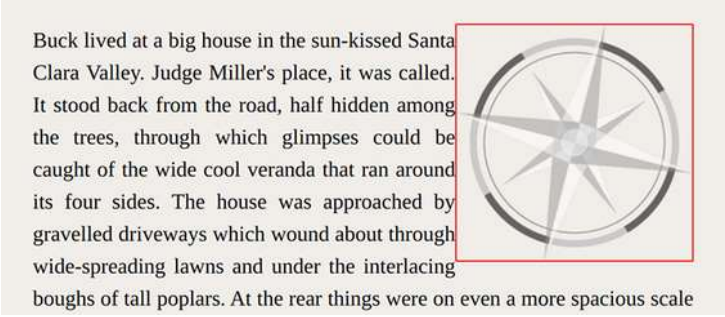


Figure 7.7 Square compass

7.3.1 Adding shape-outside: circle

To make the text follow the curve of the compass, we need to add a curve to the image for the text to wrap around. The property we'll use is `shape-outside`. This property allows us to define a shape around which the adjacent text will flow. The shape doesn't have to be rectangular; instead, it can be any of the following:

- Circle or ellipse
- Polygon
- Derived from an image (uses the alpha channel [transparency] of the image to determine what the shape should be)
- Path (in the specification but not implemented in any browser at this writing; see **http://mng.bz/aMWX**)
- Box model values (`margin-box`, `content-box`, `border-box`, and `padding-box`)
- Linear gradient

Because we have a circular graphic, the shape we're going to aim for is a circle. This decision gives us a couple of options:

- Use CSS shapes (**http://mng.bz/aMWX**).
- Use `border-radius`.

Let's first take a look at using shapes. To define our circle, we're going to use the `circle()` function, which takes an optional `radius` property and an optional `position` property to define where the center of the circle starts. If no `radius` is provided, the value defaults to `closest-side`. If the `position` property is omitted, the origin of the circle defaults to the center of the image:

```
circle(<shape-radius>, at <position> )
```

In our case, we want the center of the circle to be the middle of the image, so we won't pass a `position` property. We have to define a `radius`, however, and we're going to set it to `50%`.

How the math works

We want the radius to equal half the width of our image, which under the covers will resolve to the square root of our width squared plus our height squared divided by the square root of 2:

$$radius = \% \times \frac{\sqrt{height^2 + width^2}}{\sqrt{2}} = .5 \times \frac{\sqrt{175^2 + 175^2}}{\sqrt{2}} = 87.5$$

Because our image is square and has a width of 175, when we pass a `radius` of `50%`, it's logical that our radius would be 87.5. But if the image were rectangular, understanding how a percentage-based radius is calculated is important for understanding what the resulting output will look like.

If we had a landscape image of height `100px` and width `300px`, the radius needed to inscribe the circle when choosing a percentage-based

value is much less obvious. We can use the following formula to calculate what the radius would be:

$$radius = \% \times \frac{\sqrt{height^2 + width^2}}{\sqrt{2}} = .5 \times \frac{\sqrt{100^2 + 300^2}}{\sqrt{2}} \approx 111.8$$

Figure 7.8 shows how the radius would be applied to our square image versus a rectangular image when we use a value of `50%` in the `circle()` function.

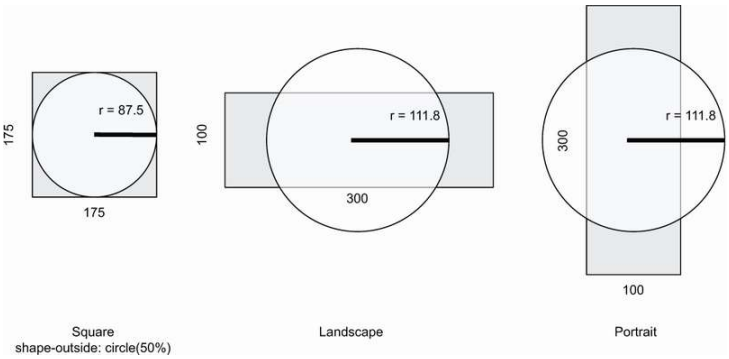


Figure 7.8 Radius applied to a square versus a rectangular image

Our image is square, so we use a `shape-outside` property with a value of `circle(50%)` for our image. Listing 7.6 shows the CSS rule. Our image is square, so it has an aspect ratio of 1 `(` width / height = 175 ÷ 175 = 1 `)`.

DEFINITION The *aspect ratio* of an image is the proportional relationship of the image's height and width calculated by dividing the width by the height.

Adding the aspect ratio isn't strictly necessary to create our shape but helps reduce layout shifts on load.

DEFINITION When an element is added to the page or its size is changed, everything after the element moves to make room for the element or fill the void left behind. The movement of elements on the page is referred to as a *layout shift*.

When the image has a set height and width or has a defined aspect ratio, the browser can save room for the image while it's being loaded, therefore reducing the layout shift. Accordingly, it's good practice to define aspect ratios and/or height and width for our images.

Listing 7.6 `shape-outside`

```
img.compass {
  aspect-ratio: 1;                ①
  float: right;                   ②
  shape-outside: circle(50%);     ③
}
```

① Aspect ratio

② Floats the image to the right

③ Adds our circle with a value of 50%

Figure 7.9 displays our output. The text wraps around the image and follows the curve, but the image isn't clipped in any way. This effect works because our image has a transparent background.

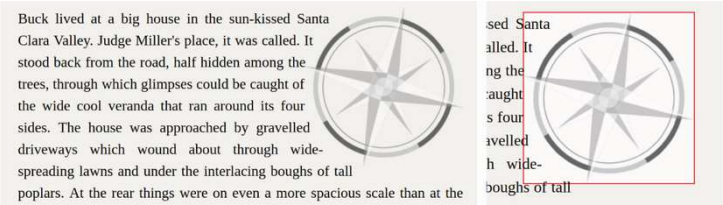


Figure 7.9 Floated compass with curved text

7.3.2 Adding a clip-path

We've curved the text, but the image is still square. If we add a background to the image, this fact becomes obvious. To make the image appear to be truly circular, we need to add a `clip-path`. The `clip-path` property also takes a shape, so we're going to pass it the same value we passed to `shape-outside`. We're also going to add some margin to our image to add a little breathing room between it and the text. Listing 7.7 shows the complete CSS for our image.

Listing 7.7 `clip-path`

```
img.compass {
  aspect-ratio: 1;
  float: right;
  shape-outside: circle(50%);
  clip-path: circle(50%);
  margin-left: 1rem;
}
```

We added a `clip-path` that matches our `shape-outside` and some margin to the left of the image to prevent the text from getting too close to the image, especially because the compass has arrows protruding from the circular outline that our `circle()` doesn't create. Figure 7.10 shows the finished output.

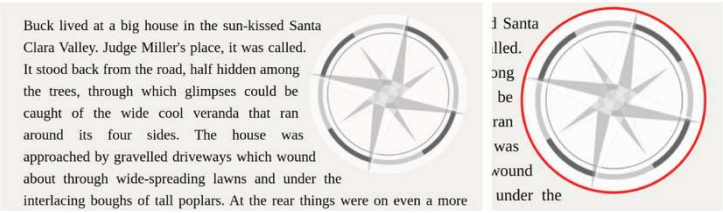


Figure 7.10 Round floated compass

When we add the `clip-path`, we observe that now the image itself, including the background, appears to be round. The corners have been clipped, and the previously square background is circular. Also, the added margin moves the text around our compass arrow, making it look less crowded.

We've demonstrated that we can create a circle by using CSS shapes. Now let's look at how to make the circle by using `border-radius`.

7.3.3 Creating a shape using border-radius

We can create a CSS shape from an element's contours when we use `border-radius` to shape the element. We still use `shape-outside`, but instead of passing in a shape, we specify the level of the box model at which we want the shape to form. Our options are

- `margin-box` —Shape follows the margins.
- `border-box` —Shape follows the borders.
- `padding-box` —Shape follows the padding.
- `content-box` —Shape follows the content.

Let's start with a clean slate, with our image floated to the right and some margin added to keep the text from crowding the image. Listing 7.8 contains our starting CSS, and figure 7.11 shows the current display.

Listing 7.8 Starting point

```
img.compass {
  aspect-ratio: 1;
  float: right;
  margin-left: 1rem;
}
```

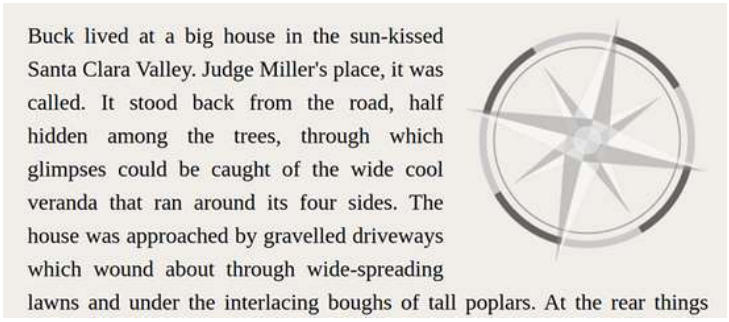


Figure 7.11 Resetting to float and adding a margin

Now let's add a `border-radius` of `50%`, which will make our image a circle. At this point, though, the text doesn't follow the curve. We still need to add the `shape-outside` property.

Our image has a margin that (ideally) we want the shape to respect, so we're going to use the `margin-box` value. The next listing shows this concept applied in code.

Listing 7.9 Adding `border-radius` and `shape-outside`

```
img.compass {
  aspect-ratio: 1;
  float: right;
  margin-left: 1rem;
  border-radius: 50%;
  shape-outside: margin-box;
}
```

Figure 7.12 shows the output with a white background and border added to emphasize the shape of the image.

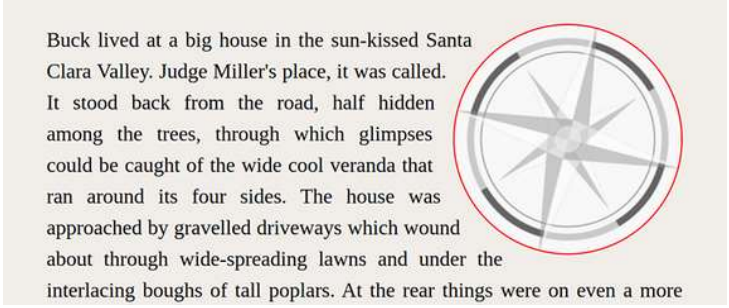


Figure 7.12 Compass shape with `border-radius` of `50%` and a `shape-outside` value of `margin-box`

Unlike when we used `shape-outside` with the `circle()` function, our image is already cropped into a circular shape, eliminating the need to use `clip-path`. This outcome is a direct result of using `border-radius`, which is doing the clipping for us.

We've seen two different ways to accomplish the same result. CSS offers more than one way to approach many problems, including this one. Neither option is particularly superior to the other. `border-radius` requires slightly less code, which gives it a slight edge, but in this case the choice is a matter of preference.

Now that we've handled the compass image, we're going to move on to wrap the text around the dog.

7.4 Wrapping text around the dog

Unlike the compass, which is a standard shape, the dog has an irregular outline. This image is line art composed of a single path, so we might be tempted to grab the path from the SVG file and use the `path()` function to create our shape. As we're about to see, however, although it's defined in the CSS specification (**https://www.w3.org/TR/css-shapes**), this technique won't work.

7.4.1 Using path() . . . or not yet

Let's open the image file in an editor to inspect the code. The following listing shows the image code redacted for brevity to highlight the important information.

Listing 7.10 `dog.svg`

```
<svg xmlns="http:/ /www.w3.org/2000/svg" viewBox="0 0 152 193">
  <defs>
    <style>
      .cls-1{
        fill:none;
        stroke:#000;
        stroke-miterlimit:10;
        stroke-width:2px;
      }
    </style>
  </defs>
  <path class="cls-1" d="M21.9135,115.62c-17.2115,4.7607-37.3354,..."/>
</svg>
```

We have the `<defs>` element, which includes the styles for the image. This part defines what the individual elements in the SVG will look like. Then we have a `<path>`, which is the element displaying the dog. This element is 1,988 characters long and quite complex, and when `shape-outside: path('M21.913...');` is pasted into the `path()` function, it doesn't seem to do anything. The reason is that when this book was written, no browser fully implemented `path()`.

When this feature is implemented, creating our paths with a graphics editor and copying them to create our shapes will be a valuable technique. But this method will have a drawback: the paths can get quite long, making maintainability dubious. In the meantime, we have a couple of alternatives:

- Creating a polygon shape that roughly matches our image, similar to the technique we used for the circle
- Using the `url()` function, which pulls in the image and bases the shape on the alpha channel

We're going to go with the second option: the `url()` function.

7.4.2 Floating the image

As we did when we handled the compass image (section 7.3), we're going to start by floating the image, but this time we'll float it to the left to break up the visual monotony of our page. Then, to create the shape, we'll use the `url()` function and pass the path to the image to it. Listing 7.11 shows the CSS applied to the dog image.

Serving the image file

When using URLs with `shape-outside`, we need to make sure we're running our code through a server so that the image is getting fetched by the browser, not read directly from the file system. This approach is related to Cross-Origin Resource Sharing (CORS) and security policies set by the browser. You can find a detailed explanation in the CSS specification at **http://mng.bz/pdMw**.

To mitigate this problem, the sample code in the GitHub repository uses `http-server`, serving the files on `localhost:8080` to accomplish this task. Another option would be to reference the hosted file in GitHub by using `shape-outside: url("https:/ /raw.githubusercontent.com/michaelgearon/Tiny-CSS-Projects/ main/chapter-07/before/img/dog.svg")`.

Listing 7.11 Dog floated left

```
img.dog {
  aspect-ratio: 126 / 161;
  float: left;
  shape-outside: url("https:/ /raw.githubusercontent.com/michaelgearon/Tiny-CSS-Projects/main/chapter-07/before/img/
}
```

We float the image left and then add our `shape-outside`, passing in a reference to the image itself. The browser will look at the transparency of the image and determine where to create the shape based on where the transparency ends. Figure 7.13 shows our output.

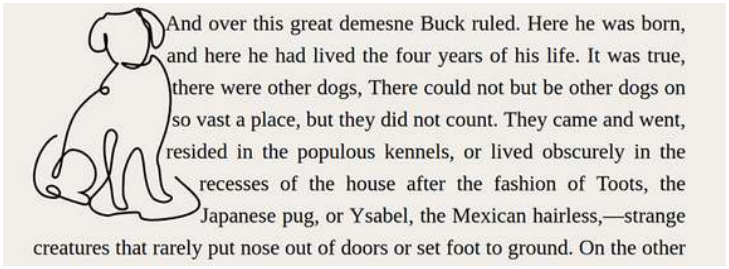


Figure 7.13 Floated dog

Because our image has an opaque line with a transparent background, the cutoff is straightforward. If our image had a gradient that went from opaque to transparent, we could tailor the cutoff by using the `shape-image-threshold` property. This property takes a value between `0` (fully transparent) and `1` (fully opaque).

7.4.3 Adding shape-margin

The next step is adding some margin to move the text away from the image because it looks rather crowded. We can't simply add a margin to the image, as we did when we floated to the right; if we try, we'll notice that the margin is ignored. Instead, we need to use `shape-margin`. The `shape-margin` property allows us to adjust the amount of space between our shape and the rest of the content. We're going to add `1em` worth of space, as shown in the following listing and figure 7.14.

Listing 7.12 Adding `shape-margin` to our rule

```
img.dog {
  aspect-ratio: 126 / 161;
  float: left;
  shape-outside: url("https:/ /raw.githubusercontent.com/michaelgearon/Tiny-CSS-Projects/main/chapter-07/before/img/
  shape-margin: 1em;
}
```

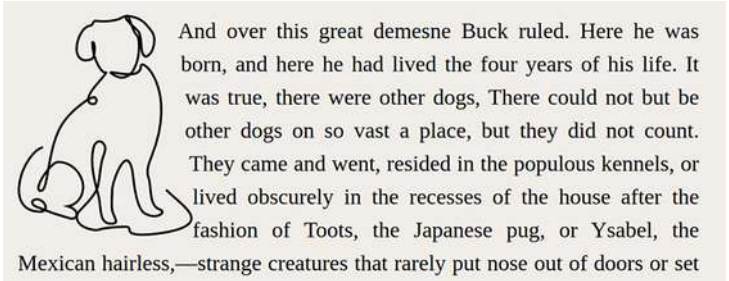


Figure 7.14 `shape-margin` applied to the image

The text at the bottom of the image is still quite close. At this point, we can add some margin to increase the space as long as the margin added is less than or equal to the `shape-margin` amount. If the value is greater than the `shape-margin` amount, the margin will still take effect, but only as much as the `shape-margin` amount. Keeping this caveat in mind, we'll add `1em` of margin to the right of the image. The next listing shows the completed CSS for the dog image.

Listing 7.13 Completed dog image

```
img.dog {
  aspect-ratio: 126 / 161;
  float: left;
  shape-outside: url('img/dog.svg');
  shape-margin: 1em;
  margin-right: 1em;
}
```

The combination of `shape-margin` and `margin-right` pushes the text away from our image, creating the polished result we see in figure 7.15.

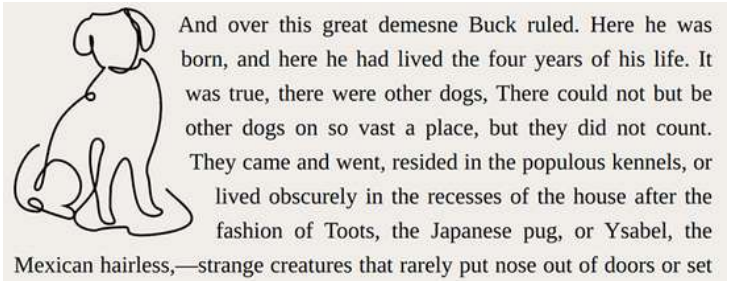
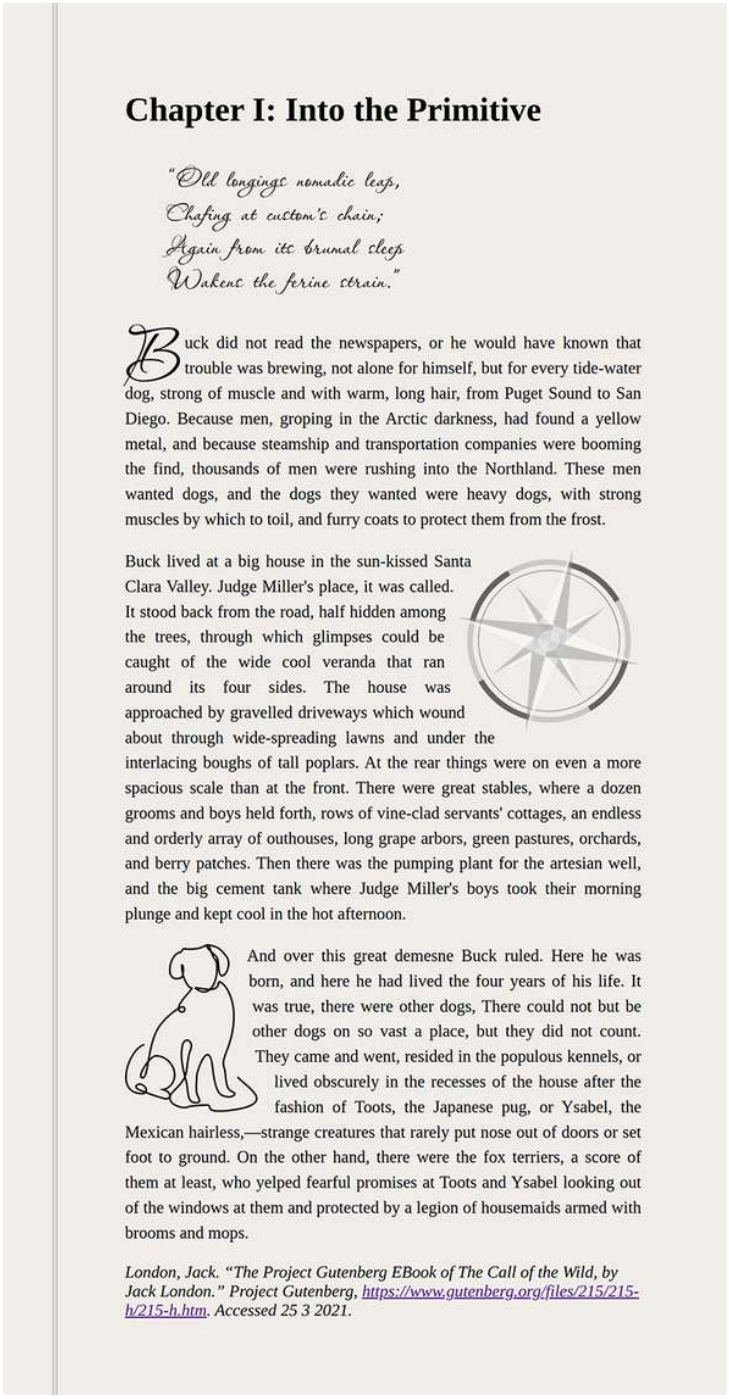


Figure 7.15 Finished floated dog

With this last piece completed, we've finished styling our page (figure 7.16). We have a layout that's visually appealing and much more interesting than the one we started with.



Chapter I: Into the Primitive

"Old longings nomadic leap,
Chafing at custom's chain;
Again from its brumal sleep
Wakens the ferine strain."

Buck did not read the newspapers, or he would have known that trouble was brewing, not alone for himself, but for every tide-water dog, strong of muscle and with warm, long hair, from Puget Sound to San Diego. Because men, groping in the Arctic darkness, had found a yellow metal, and because steamship and transportation companies were booming the find, thousands of men were rushing into the Northland. These men wanted dogs, and the dogs they wanted were heavy dogs, with strong muscles by which to toil, and furry coats to protect them from the frost.

Buck lived at a big house in the sun-kissed Santa Clara Valley. Judge Miller's place, it was called. It stood back from the road, half hidden among the trees, through which glimpses could be caught of the wide cool veranda that ran around its four sides. The house was approached by gravelled driveways which wound about through wide-spreading lawns and under the interlacing boughs of tall poplars. At the rear things were on even a more spacious scale than at the front. There were great stables, where a dozen grooms and boys held forth, rows of vine-clad servants' cottages, an endless and orderly array of outhouses, long grape arbors, green pastures, orchards, and berry patches. Then there was the pumping plant for the artesian well, and the big cement tank where Judge Miller's boys took their morning plunge and kept cool in the hot afternoon.

And over this great demesne Buck ruled. Here he was born, and here he had lived the four years of his life. It was true, there were other dogs, There could not but be other dogs on so vast a place, but they did not count. They came and went, resided in the populous kennels, or lived obscurely in the recesses of the house after the fashion of Toots, the Japanese pug, or Ysabel, the Mexican hairless,—strange creatures that rarely put nose out of doors or set foot to ground. On the other hand, there were the fox terriers, a score of them at least, who yelped fearful promises at Toots and Ysabel looking out of the windows at them and protected by a legion of housemaids armed with brooms and mops.

London, Jack. "The Project Gutenberg EBook of The Call of the Wild, by Jack London." Project Gutenberg, https://www.gutenberg.org/files/215/215-h/215-h.htm. Accessed 25 3 2021.

Figure 7.16 Finished layout

We've created a layout made possible by the use of float. We couldn't have achieved the same result by using Flex or Grid easily. Whether we use it on its own, as in our drop-cap example, or in conjunction with shapes (which, granted, are rather new as well), float continues to be a valuable asset for us to keep in our toolbox.

Summary

- Leading, the amount of space between lines, is important for legibility.
- Float can be used in conjunction with `::first-letter` to create drop caps.
- Not all typefaces have the same size and line heights when given the same size value.

- The `shape-outside` property uses CSS shapes to alter the shape of an element.
- Circular shapes can be created with `border-radius`.
- Inline content adjacent to a floated CSS shape will follow the shape.
- When we use `url()` with `shape-outside`, the image file must be fetched by the browser (hosted or via `http-server` or the equivalent).
- The `shape-margin` property sets the margin of a shape.
- Some layouts can't be created without the use of float.