

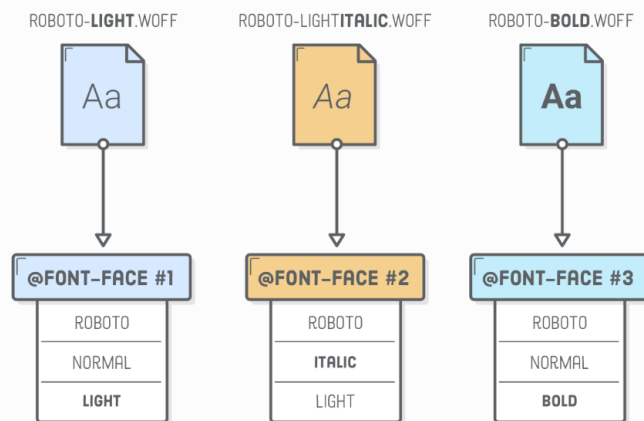
distinguish between our light, italic, and bold faces, we'll add `font-style` and `font-weight` properties to the at-rule. Replace all the `@font-face` declarations in `typo.css` with the following:

```
@font-face {
  font-family: 'Roboto';
  src: url('Roboto-Light-webfont.woff') format('woff');
  font-style: normal;
  font-weight: 300;
}

@font-face {
  font-family: 'Roboto';
  src: url('Roboto-LightItalic-webfont.woff') format('woff');
  font-style: italic;
  font-weight: 300;
}

@font-face {
  font-family: 'Roboto';
  src: url('Roboto-Bold-webfont.woff') format('woff');
  font-style: normal;
  font-weight: 700;
}
```

file. The first `@font-face` is saying it's a Roboto font that's roman (normal) and has a font weight of 300 (aka "light"). The second says it's also in the Roboto family and has a weight of 300, but it's italic. Finally, the third at-rule lets our the browser know that `Roboto-Bold-webfont.woff` contains the 700-weight (aka "bold") roman face.



Letting the browser know that our font faces are related makes our CSS much more intuitive. We can set the default font family and weight in our body selector. Then, when we want to use italics or bold for a particular element, we can simply specify a `font-style` or `font-weight` and the browser will pull the corresponding `.woff` file:

```
body {
  font-family: 'Roboto', sans-serif;
  font-weight: 300;
  /* ... */
}

em {
  font-style: italic;
}

strong {
  font-weight: bold; /* Or 700 */
}
```

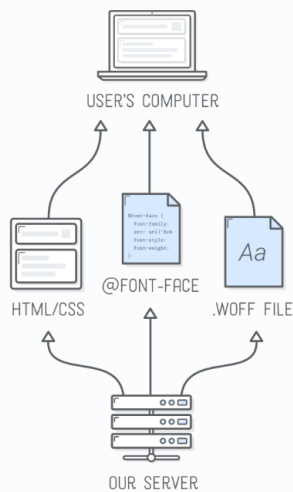
These happen to be the default `font-style` and `font-weight` values for `` and `` elements, so we don't *really* need to include the last two rules here. Note that the only human-friendly keywords available for `font-weight` are `normal` (400) and `bold` (700). Any other boldness levels need to

set numerically.

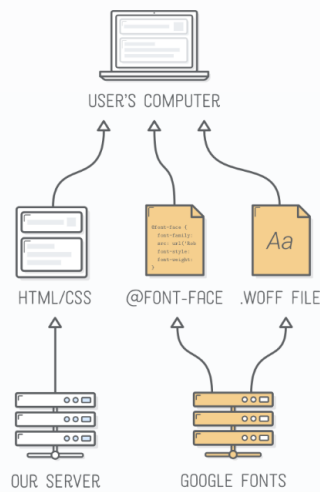
— EXTERNALLY HOSTED WEB FONTS —

Ok! That was complicated. Next, we're going to explore the easier method of using web fonts: externally hosted via [Google Fonts](#). This lets us skip the first two steps of locally hosted fonts. Instead of adding `.woff` files to our project and embedding them with `@font-face`, we can let Google Fonts do this part for us.

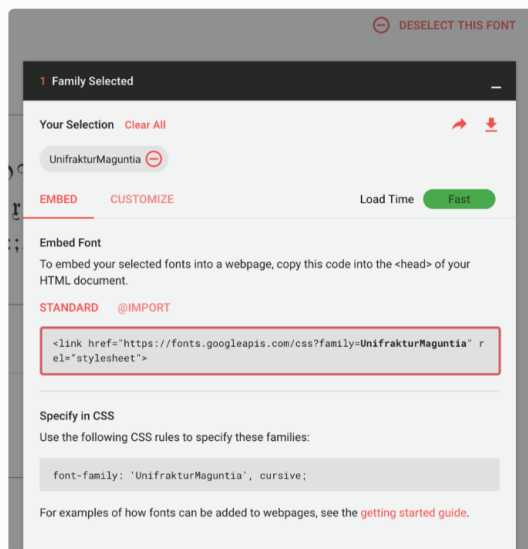
LOCALLY HOSTED FONTS



EXTERNALLY HOSTED FONTS



In this section, we're going to be working on `history.html`, so open up that file in both your text editor and a web browser. If you want a brief history of typography going all the way back to the first printing press, take a quick read through the example text. Right now, each section in `history.html` is using Roboto Light, but we're going to change all of them to be representative of the period they're talking about.



Let's begin by changing the font for the *Gothic/Blackletter* section. In [Google Fonts](#), search for **UnifrakturMaguntia**. It should look like something a monk wrote in the middle ages. Click **Select this font**. In the pop-up menu, you'll see a `<link/>` element. Copy this into the `<head>` of `history.html`, above the `<link/>` element that includes our `typo.css`

stylesheet.

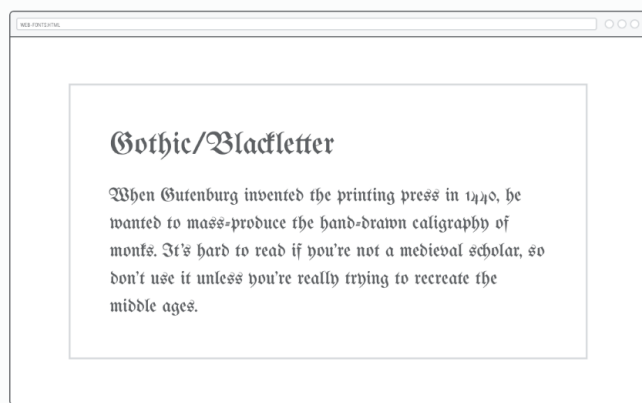
```
<link href="https://fonts.googleapis.com/css?family=UnifrakturMaguntia" rel="st
```

Remember that `<link/>` is how we [include an external stylesheet](#), and that's exactly what the above HTML is doing. However, instead of linking to a local CSS file, it's including some CSS defined by Google Fonts. If you paste the href value into your browser, you'll find the same `@font-face` declaration that we used in the previous section—except we didn't actually have to write it this time. Yay!

Now that we've embedded our UnifrakturMaguntia web font, we should be able to use it to style any HTML element we want. Add the following to the `<head>` of `history.html`:

```
<style>
  .blackletter {
    font-family: 'UnifrakturMaguntia', cursive;
  }
</style>
```

That first section has a `class='blackletter'` attribute, so it should now be printed in gothic letters:



Google Fonts are a quick and easy solution, but professional sites should typically use locally hosted web fonts. This gives you a lot more flexibility (you're not limited to Google's font offering) and can have performance/reliability gains if you've optimized the rest of your site correctly.

TOO MANY FONT FILES

Speaking of performance, let's do something awful. There's another 10 sections on our `history.html` page, and we want to give each one its own web font. We can embed multiple fonts in a single `<link/>` element, so change our Google Fonts stylesheet to include the rest of them:

```
<link href="https://fonts.googleapis.com/css?family=Alfa+Slab+One|Droid+Sans+Mo
```

Note that you can generate this in Google Fonts by selecting multiple fonts before copying the `<link/>` element. Next, add all these new fonts to the `<style>` element of `history.html`:

```
.old-style {
  font-family: 'Sorts Mill Goudy', serif;
}
.transitional {
  font-family: 'Libre Baskerville', serif;
}
.didot {
  font-family: 'Rufina', serif;
}
```

```

.slab {
  font-family: 'Rokkitt', serif;
}
.fat-face {
  font-family: 'Alfa Slab One', cursive;
}
.grotesque {
  font-family: 'Roboto', sans-serif;
}
.geometric {
  font-family: 'Questrial', sans-serif;
}
.humanist {
  font-family: 'Lato', sans-serif;
}
.display {
  font-family: 'Lobster', cursive;
}
.monospace {
  font-family: 'Droid Sans Mono', monospace;
}

```

Now, each section of `history.html` is rendered in a font from the era it's describing. This serves as a nice introduction to the historic significant of different fonts, but you should **never, ever include this many web fonts on a real web page**.

Don't forget that each web font is actually a `.woff` or `.woff2` file that your browser needs to load before it can render the page. More fonts means longer load times. The key to using web fonts effectively is to find a balance between performance (fewer web fonts) and a beautifully typeset document (more web fonts).

And that's more than you could ever want to know about web fonts. The rest of this chapter shifts gears into basic typographic principles. These are simple guidelines (with simple CSS implementations) that often make the difference between a professional web page and an amateur one.

PARAGRAPH INDENTS

Separating paragraphs from one another is one of the most fundamental functions of typography. There's two generally accepted solutions: either use a first-line indent *or* a margin between the paragraphs. Your readers (hopefully) aren't stupid—they don't need two signs that a new paragraph is happening, so never use *both* an indent and a margin. That would be redundant.

INDENTS

Dis nec nascetur adipiscing a nec sed scelerisque urna sem dignissim vestibulum eget lorem vestibulum.

Parturient elementum eros a scelerisque felis velit fames hac hendrerit mi sociis.

A molestie semper nam eget laoreet placerat blandit consetetur vel dignissim.

MARGINS

Dis nec nascetur adipiscing a nec sed scelerisque urna sem dignissim vestibulum eget lorem vestibulum.

Parturient elementum eros a scelerisque felis velit fames hac hendrerit mi sociis.

A molestie semper nam eget laoreet placerat blandit consetetur vel dignissim.

NEVER BOTH

Dis nec nascetur adipiscing a nec sed scelerisque urna sem dignissim vestibulum eget lorem vestibulum.

Parturient elementum eros a scelerisque felis velit fames hac hendrerit mi sociis.

A molestie semper nam eget laoreet placerat blandit consetetur vel dignissim.

The CSS `text-indent` property defines the size of the first-line indent of a particular element (usually a `<p>`). We can explore this in our `indents.html` page. Go ahead and change the existing bottom margin styles in the first section to an indent by adding the following rules to the `<style>` element:

```

<style>
  .paragraph-indent p {

```

```

    text-indent: 1em;
    margin-bottom: 0;
}
.paragraph-indent p:first-of-type {
    text-indent: 0;
}
</style>

```

Note that the first paragraph after a heading should never be indented because, well, it's usually pretty obvious that it's a new paragraph. This is a pretty good use case for the `:first-of-type` pseudo-class.

And here's a negative example so we remember what *not* to do. Add this to the page-specific styles in `indents.html`:

```

/* DESIGNERS WILL JUDGE YOU FOR THIS */
.never-both p {
    text-indent: 1em;
    margin-bottom: 1em;
}

```

It might seem silly, but we're not kidding when we say that good designers *will* judge you for this.

TEXT ALIGNMENT

The alignment of text has a subconscious impact on how you read it. You've probably never noticed it before, but your eyes don't move in a smooth motion as they read over a paragraph—they jump from word to word and from line to line. Your eyes fixate on certain spots and skip over other ones.

YOU READ LIKE THIS

Dis nec nascetur adipiscing a nec sed
 scelerisque urna sem dignissim vestibulum
 eget lorem vestibulum. Parturient
 elementum eros a scelerisque felis velit
 fames hac hendrerit mi sociis.

NOT LIKE THIS

Dis nec nascetur adipiscing a nec sed
 scelerisque urna sem dignissim vestibulum
 eget lorem vestibulum. Parturient
 elementum eros a scelerisque felis velit
 fames hac hendrerit mi sociis.

In a well-designed HTML document, text alignment is never an arbitrary decision. It takes into account this little bit of human physiology. Good text alignment actually makes it easier for users to read your content by giving their eyes an anchor to jump to when they move from line to line.

The next few sections explain the proper times to use left, center, right, and justified text alignment. All of these examples rely on the `text-align` property, which controls the text alignment of a particular HTML element. We set up the `alignment.html` page in our example project with some convenient scenarios.

LEFT ALIGNMENT

Most of your text should be left-aligned because it gives the reader a vertical anchor to jump back to on every line. Long runs of text, in particular, should almost always be left-aligned. Short runs of text and headings have a little bit more leeway.

```

Dis nec nascetur adipiscing a nec sed
scelerisque urna sem dignissim vestibulum
eget lorem vestibulum. Parturient
elementum eros a scelerisque felis velit
fames hac hendrerit mi sociis.

```

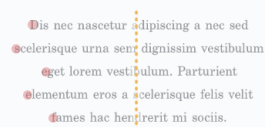
Left alignment is the default value for `text-align`, but if we wanted to be explicit, we could add the following rule to the `<style>` element of our `alignment.html` file:

```
<style>
  .left {
    text-align: left;
  }
</style>
```

Of course, if you're working on a website that's in a language that's written right-to-left instead of left-to-right (like Arabic), you can go ahead and swap all this advice with the *Right Alignment* section below.

CENTER ALIGNMENT

Center-aligned text doesn't have that anchor, so it's easier for the eye to get lost when it tries to jump to the next line. It's best suited for short line lengths (more on that later) and for special kinds of content like poems, lyrics, and headings.



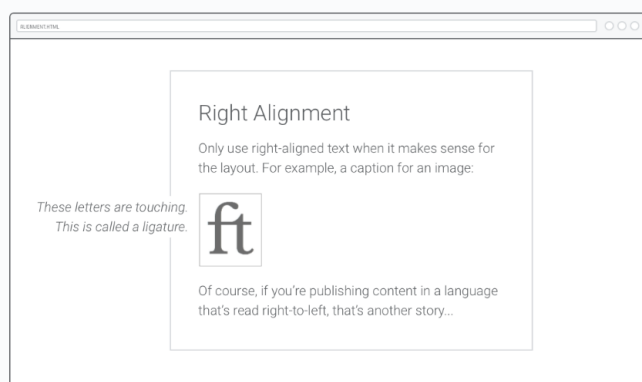
Go ahead and center-align the second paragraph in `alignment.html` with another page-specific style:

```
.center {
  text-align: center;
}
```

Notice how the page now feels a little disjointed. The center-aligned second paragraph breaks the flow of the left-aligned first paragraph. Generally speaking, text alignment should be consistent throughout a web page. If you're going to center a heading, center *all* of your headings.

RIGHT ALIGNMENT

Another consideration when choosing text alignment is the relationship it creates with the surrounding elements. For instance, take a look at that third section in `alignment.html`. We want to move the image's caption to the left of the image and right-align it to make it look like it's attached to the image:



Our example image is wrapped in a `<figure>` and the caption text is in a `<figcaption>`, so adding the following to the `<style>` element of `alignment.html` should result in the above layout.

```
figcaption {
  display: none;
```

```

}
@media only screen and (min-width: 900px) {
  figure {
    position: relative;
  }
  figcaption {
    display: block;

    font-style: italic;
    text-align: right;
    background-color: #FFFFFF;

    position: absolute;
    left: -220px;
    width: 200px;
  }
}

```

This also happens to be a good example of [advanced positioning](#). The relative position of the `<figure>` sets the coordinate system for the `<figcaption>`'s absolute positioning. By nudging the caption left by 220px and giving it an explicit width of 200px, we get a nice 20-pixel margin between the image and its caption.

Like centered text, right alignment should usually be reserved for these kinds of special design scenarios because its jagged left edge makes it harder for the reader to find the next line.

JUSTIFIED TEXT

Justified text is created by subtly adjusting the space between words/letters and splitting long words with hyphens until each line is the same width. Without a high-quality hyphenation engine, justified text results in awkwardly large spaces between words. These uneven spaces make it harder for the eye to move horizontally across the text.

BAD HYPHENATION

Dis nec nascetur adipiscing a nec sed
scelerisque urna sem dignissim vestibulum
eget lorem vestibulum. Parturient
elementum eros a scelerisque felis velit
fames hac hendrerit mi sociis.

Dis nec nascetur adipiscing a nec sed
scelerisque urna sem dignissim vestibulum
eget lorem vestibulum. Parturient
elementum eros a scelerisque felis velit
fames hac hendrerit mi sociis.

GOOD HYPHENATION

Dis nec nascetur adipiscing a nec sed
scelerisque urna sem dignissim vestibulum eget
lorem vestibulum. Parturient elementum eros
a scelerisque felis velit fames hac hendrerit
mi sociis.

Dis nec nascetur adipiscing a nec sed
scelerisque urna sem dignissim vestibulum eget
lorem vestibulum. Parturient elementum eros
a scelerisque felis velit fames hac hendrerit
mi sociis.

Unfortunately, most browsers don't have *any* kind of built-in hyphenation engine, so you're better off avoiding justified text in HTML documents. We can take a look by adding one more `text-align` rule to our `alignment.html` file:

```

.justify {
  text-align: justify;
}

```

Compare this with the left-aligned paragraph. It's subtle, but the left-aligned paragraph is more uniform and inviting.

Just as alignment isn't an arbitrary decision, neither is the space between text. In this section, we're concerned with the responsible use of three CSS properties:

- `margin-top` (or `padding-top`)
- `margin-bottom` (or `padding-bottom`)
- `line-height`

The first two should be pretty familiar by now, and they define the vertical space between separate paragraphs. The new `line-height` property determines the amount of space between lines *in the same paragraph*. In traditional typography, `line-height` is called “leading” because printers used little strips of lead to increase the space between lines of text.

Together, these properties control the “vertical rhythm” of a web page. There's all sorts of techniques to figure out the “optimal” vertical rhythm for a given layout, but the general principles are:

- Give things enough space to breathe.
- Use *consistent* spacing throughout the page.

To demonstrate this, we're going to destroy the vertical rhythm in the second half of our `spacing.html` page. Go ahead and add the following page-specific styles to `spacing.html`

```
<style>
.messy {
  line-height: 1.2em;
}
.messy h2 {
  line-height: .9em;
}
.messy:last-of-type {
  line-height: 1.5em;
}
.messy:last-of-type h2 {
  margin-bottom: .3em;
}
.messy .button:link,
.messy .button:visited {
  margin-top: 0;
}
</style>
```

A few small changes to line height, paddings, and margins can have a dramatic impact on the quality of a page:

Give Your Elements Enough Room to Breathe

One of the easiest ways to make your web pages look more professional is to add bigger margins or padding between everything.

By everything, we mean the space between headings, paragraphs, images, form controls, the vertical space between lines, the horizontal space between the edge of the page, and pretty much every other margin or padding you can think of.

Consistent Spacing

Using a consistent margin between headings, paragraphs, images, and other elements gives a vertical rhythm to the page. See how this button doesn't seem to break the flow of the page?

[Learn More About Typography](#)

Consistent spacing makes your web page feel more like a cohesive whole, rather than a bunch of unrelated graphical elements.

Without Enough Space, Things Feel Messy

One of the easiest ways to make your web pages look less professional is to reduce margins or padding between everything.

By everything, we mean the space between headings, paragraphs, images, form controls, the vertical space between lines, the horizontal space between the edge of the page, and pretty much every other margin or padding you can think of.

Inconsistent Spacing

Using an inconsistent margin between headings, paragraphs, images, and other elements destroys the vertical rhythm of a page. See how this button seems to break the flow of the page?

[Learn More About Typography](#)

The difference between the top and bottom margin makes it feel like it doesn't fit in, and your web page is no longer a cohesive whole.

There's a surprising amount of math and psychology that goes into

calculating the vertical rhythm of a page, but that's a job for your designer. As a developer, you need to know the CSS properties to implement what they're asking for. More importantly, you have to understand that your designer really cares about this kind of stuff, so you should be paying very careful attention to your margin, padding, and line-height properties.

LINE LENGTH

If the vertical spacing of your text isn't arbitrary, it should be no surprise that the horizontal spacing isn't, either. "Line length" or "measure" refers to the horizontal length of your text. You can think of it as the number of characters or words that fit into a single line. Measure has everything to do with the following CSS properties:

- width
- margin-left (or padding-left)
- margin-right (or padding-right)

A good rule-of-thumb is to limit the number of characters on a single line to around 80. Like alignment, this subtly affects the readability of your content. It takes energy for your eye to move from the left edge of a paragraph to the right, and the farther it has to scan, the faster it gets tired. Longer lines also make it easier to get lost when you finish a line and need to jump back to the beginning of the next line.

SHORT LINE LENGTH

Dis nec nascetur adipiscing a nec
sed scelerisque urna sem
dignissim vestibulum eget lorem
vestibulum. Parturient elementum
eros a scelerisque felis velit fames
hac hendrerit mi sociis.

LONG[ER] LINE LENGTH

Dis nec nascetur adipiscing a nec sed scelerisque urna sem
dignissim vestibulum eget lorem vestibulum. Parturient elementum
eros a scelerisque felis velit fames hac hendrerit mi sociis.

These are the reasons why so many websites (including this one) use [fixed-width layouts](#) or split content into multiple columns on wider screens. Without constraining the width of the page or dividing it into manageable columns, line length becomes unacceptably long.

In our example project, the `line-length.html` file has decent measure. Let's see what happens when we break the bottom half of the page by adding the following to its `<head>`:

```
<style>
  @media only screen and (min-width: 580px) {
    .not-so-manageable {
      max-width: 100%;
      margin-left: 2em;
      margin-right: 2em;
    }
  }
</style>
```

Now, the second section stretches to fill the full width of the browser window. It feels a little bit more unapproachable due to the long line length. Again, the goal of good web typography is to make it as easy as possible for visitors to digest your content.

OTHER BASIC TYPOGRAPHY GUIDELINES

That should be enough to get you on your way towards quality web typography. Typography is a whole industry, and we've barely scratched the surface. However, getting any deeper into it would be more design than web development, so we'll just leave you with a few final guidelines:

- Use a font-size between 14px and 20px for the body element.
- Use “curly quotes” and apostrophes with the `’`, `‘`, `”`, and `“`; HTML [entities](#).
- Use proper dashes (`–`, `—`) and other symbols (`©`).
- Don't use text-decoration: underline except for hover states.
- Use *real* italic fonts over synthesized ones if not it's too much of a performance burden.

If you find this stuff fascinating, [Practical Typography](#) has a fantastic list of general rules to follow when typesetting a document.

SUMMARY

The goal of this chapter was twofold:

- Learn the mechanics of web fonts and basic CSS typography properties.
- Understand how designers think about typography.

You might not be able to create a beautifully typeset web page from scratch after reading this chapter, but that wasn't the point. It was to make you *aware* of the invisible art of typography. You should now have the vocabulary to talk about things like font families, faces, weights, and styles, as well as leading, measure, and vertical rhythm.

The most important thing you should take away from this chapter is the fact that nothing is arbitrary in a well-designed web page. The font sizes, indent style, text alignment, line height, margins, and every other tiny facet of the page was carefully considered. There was a *purpose* behind all of these decisions.

All of the CSS properties we've covered throughout this tutorial are actually kind of simple. When it comes down to it, we've really just been moving a bunch of boxes around, changing some colors, and altering the appearance of our text. The *meaning* behind these things comes from the underlying design and the business goals of the website you're implementing.

But, that's for another tutorial. Believe it or not, you've reached the end of [HTML & CSS is Hard](#). We've covered all the HTML elements and CSS properties you need to build professional web pages. The only thing missing is *experience*. Your next step is to practice all these new skills by building a bunch of web pages from scratch.

Stay tuned for more tutorials!

[TABLE OF CONTENTS >](#)



InternetingIsHard.com is an independent publisher of premium web development tutorials. All content is authored and maintained by Oliver James. He loves hearing from readers, so [come say hello!](#)

More tutorials are coming. (Lots more.)

Enter your email above, and we'll let you know when they get here.