Font stacks and other explorations on typography on the web

I have dealt with some of these topics before in different places but now want to weave a more coherent narrative around these topics and try to find a way how to tie them together so it makes sense.

What is a font-stack and why it matters

It wasn't until I started reading blogs and books by Jason Pamental, Bram Stein, and Tim Brown among others that I realized that web typography is a lot more than choosing the right fonts.

The font stack is the set of fonts that we use for our site or application. The browser's rendering engine will pick the first font on the stack that is installed and available.

To install the font we use CSS's `@font-face` declaration like this:

```
@font-face {
  font-family: 'Ubuntu'; 'Ubuntu'/* regular */url('Ubuntu-R-webfont.woff2
  font-weight: normal;
  font-style: normal;
}
@font-face {
  font-family: 'Ubu') format('@font-facesrc: url('Ubuntu-RI-webfont.woff2
  font-weight: norm'Ubuntu-RI-webfont.ttf'url('Ubuntu-RI-webfont.ttf')ily
  src: url('Ubuntu-B-webfont.woff2') forma'Ubuntu'@font-face {
  font-family: 'Ubuntu'; /* bold */off'), url('Ubuntu-B-webfont.ttf') for
'), url('url('Ubuntu-B-webfont.woff')
      format('woff'), url('Ubuntu-B-webfont.ttf')old italic */
  src: url('Ubuntu-BI-webfont.woff2') format('woff2''Ubuntu-BI-webfont.wo
  font-weight: bold;
  font-style: ita'truetype'url('Ubuntu-BI-webfont.woff')
      format('woff'), url('Ubuntu-BI-webfont.ttf') format('truetype');
  font-weight: bold;
  font-style: italic;
}
```

To prevent faux bold and italics we load four fonts, one each for normal, italics, bold and bolditalics.

But in this `@font-face` demo we can already see one of the performance pitfalls of using web fonts: The size of the files.

Each of these font files can range from just a few tens of kilobytes to 400 kilobytes or more for a single weight of the font and 1.2 **mega bytes** for the four weights that we're using.

So try to find smaller sizes or subset the fonts.

Subsetting fonts will make the files smaller by only adding the glyphs (character, punctuations and others) that are needed to render the page. You must be careful when doing this as any glyph missing will be rendered in a backup font, either one you designate or one of the defaults for serif, sanserif and monospace.

I've discussed subsetting fonts in two posts: subsetting fonts and more on font subsetting.

This is what a font stack looks like:;

```
/* Times New Roman-based stack */
font-family: Cambria, 'Hoefler Text', Utopia,
  'Libera'Hoefler Text'imbus Roman No9 L Regular', Times, 'Times New Roman

', Times, '/* Modern Georgia-based serif stack */y: Constantia, 'Lucida Br
  'DejaVu Serif', 'Bitstream Vera Serif', 'Liberation Serif', Georgia, ser

/* Traditio'Lucida Bright'/* Traditional Garamond-based serif stack */noty
  'Book Antiqua', Baskerville, 'Bookman Old Style', 'Bitstream Charter',
  'Nimbus Roman No9 L', Garamond, 'Apple Garamond', 'ITC Garamond Narrow'
  'New Century Schoolbook', 'Century Schoolbook', 'Century Schoolbook L',
  Georgia, serif;

/* Helvetica/Arial-based s', Palatino, Palladio, '/* Helvetica/Arial-base
  'Gill Sans MT', 'Myriad Pro', Myriad, 'DejaVu Sans Condensed',
  'Liberation Sans', 'Nimbus Sans L', Tahoma, Geneva, 'Helvetica Neue',
  Helvetica, Arial, sans-serif;

/* Verdana-based sans serif stack */
font-family: 'Gill Sans'/* Verdana-based sans serif stack */ 'Lucida Sans
  'DejaVu Sans', 'Bitstream Vera Sans', 'Liberation Sans', Verdana,
  'Verdana Ref', sans-serif;

/* Monospace stack */
font-family: Consolas, 'Andale Mono WT''Lucida Sans'/* Monospace stack */
font-family: Consolas, 'Andale Mono WT', 'Andale Mono', 'Lucida Console',
  'Lucida Sans Typewriter', 'DejaVu Sans Mono', 'Bitstream Vera Sans Mono
  'Liberation Mono', 'Nimbus Mono L', Monaco, 'Courier New', Courier, mon
```

These are fairly complex stacks and the closer you move to the right the more likley you are to find an equivalent until you get to the final font of the stack, one of five generic fonts: serif, sans-serif, cursive, fantasy, and monospace. The generic fonts are different than the other fonts in the stack, rather than trying match a specific font, they ask the browser to provide a default fallback.

According to the CSS Fonts Level 3 Specification generic fonts section:

> All five generic font families must always result in at least one matched font face, for all CSS implementations. However, the generics may be composite faces (with different typefaces based on such things as the Unicode range of the character, the language of the containing element, user preferences and system settings, among others). They are also not guaranteed to always be different from each other.
>
> User agents should provide reasonable default choices for the generic font families, which express the characteristics of each family as well as possible, within the limits allowed by the

> underlying technology. User agents are encouraged to allow users to select alternative choices for the generic fonts.

Making sure your fallback fonts work well

One problem with fallback fonts is that they may not match the web font they replace 100%… and this will definitely become a problem when working on how will the layout look in your fallback font and whether it'll work at all or not.

Monica Dinculescu created a Font Style Matcher to work around this issue. We match the fonts as close as possible as show in the figure below and then copy the two CSS blocks.

Font Matcher Demo of Work In Progress

Figure 1: Font Matcher Demo of Work In Progress

What we get when we copy the CSS is one block for the fallback font (Georgia) and one for the web font (Merriweather). Since CSS will apply both sequences in order it'll jump from Georgia to Merriweather but, since we took care that the fonts would match, the jarring change will be minimized.

```css
font-family: Georgia
font-size: 16px
line-height: 1.6;

font-family: Merriweather
font-size: 16px
line-height: 1.6;
```

System Fonts: Use what's given to

you

OK, we have our font stack, we understand what it is and we can customize it, after a fashion, to reduce any jarring text change between fallback fonts and the web fonts we've downloaded.

Another way provide a better experience is to use system fonts; the same fonts that the Operating System uses on your machine. We use the following CSS to load system specific fonts across platforms, knowing that the CSS parser will stop when it hits the first match.

Also note that we use the long form, with different attributes for each of `font-family`, `font-size` and `line-height` to work around a bug that may cause the `-apple-system` font (used in Safari) to be considered a named prefixed and cause the rest of the declaration (fonts for all other systems) to be ignored.

```css
html {
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto',
    'Oxygen', 'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans',
    'Helvetica Neue', sans-serif;
  font-size: 16px;
  line-height: 1.2;
}
```

CSS fonts module Level 4 introduces a new generic font family. The system-ui generic font family will always map to the default user-interface font on all platforms.

There is no browser support for system-ui yet, when it arrives on browsers, you'll be able to reduce the above font stack to a single item:

```css
html {
  font-family: system-ui;
}
```

This is cool but what's the use case for system fonts?

System fonts are best used for the UI of your application. This will give users a familiar feell by using the same fonts throughout the Operating System.

Variable Fonts: New Swiss Army Knife?

Opentype 1.8 introduced the concept of variable fonts. In these variable fonts we get multiple axes representing six predefined axes: weight, width, optical size, italic, slant; plus an unlimited numbers of custom axes to define your own properties in the font. This is all packaged together in one file rather than have multiple files to represent the different font faces we use.

If you have variable fonts avaialble you can use Axis Praxis a preview and experimentation tool. You can work with any of the available fonts or upload your own variable font to play with, all available axes will appear in the tool to play with.

One thing to notice about Variable Fonts is that there are only six pre-defined axes (duiscussed above) and no limit to the number of custom axes a font can have. This means that we can have a single font that works for body copy, heaadings, and text-decorations if needed.

For examples of what you can do with variable fonts check out Variable Fonts

Demo and Explainer for work with early versions of variable fonts.

Another idea worth exploring is whether we can use a single variable font for our site. This will depend on what Axes the variable font makes available and how complex is the typography for your site. It should be possible but requires careful planning and font selection.

Choosing Fonts

Choosing your font stack is a tricky balancing act of many conflicting and, sometimes, contradictory priorities. We'll look at what I consider the most important design and non-design considerations.

Design Considerations

These are some of the design consideratons that go into my font selection process.

If you think I missed anything, ping me on twitter (@elrond25)

Font Selection

Display faces are designed to be used large, such as in headlines and are usually less readable at smaller sizes and should not be used for body copy. These are called display fonts or faces.

Other typefaces are designed specifically to be used in large areas of smaller body copy. These are called text, body, or copy fonts.

Identifying fonts for your specific needs is the first step but then comes the big question: **Serif or Sans-Serif?**

Sans Serif Font Serif font with serif lines show in red

Figure 2: Comparison of sans-serif font (left) and serif font

showing serif lines in red (right)

I normally work with whatever font I think will work best for the project I'm working on either on its own if I'm using it for both headings and copy or paired with another font.

So which is more legible: serif or sans-serif typefaces? In print it appears that serif fonts are considered more legible. Serif fonts allow the eye to flow more easily over the text, improving reading speed and decreasing eye fatigue.

For the web the situation is somewhat different. We will never set as much text on the web as we would in a book or magazine and there are sans-serif fonts that work just as well as serif for the amount of text that we use on the web.

This is one subject that is debated whenever typography people talk about fonts – some people argue that serif typefaces make text harder to read in smaller screens. Others believe that there's no difference. My position is to test the fonts and then have people read it.

Font Sizes

We need to make sure that the text is not too small or too large for the device users are viewing the content on. The default size in browsers is 16px. I like using that as my base and then working with Modular Scales.

> A modular scale, like a musical scale, is a prearranged set of harmonious proportions.
>
> Robert Bringhurst

I use Tim Brown and Scott Kellum's modular scale builder where we can play with a base value (for example 16px, the default for most browsers) and a set of predefined ratios (I normally use the golden ratio, 1.618) to create a series of values that are ratios of the base number.

For the values of 16px and 1:1.618 ratios the scale looks like this:

```
2206.258em
1363.571em
842.751em
520.86em
321.916em
198.959em
122.966em
75.999em
46.971em
29.03em
17.942em
11.089em
6.854em
4.236em
2.618em
1.618em
1em
0.618em
0.382em
0.236em
0.146em
0.09em
0.056em
```

So you can take these values and plug them (up or down) to your CSS. If the Golden Ratio doesn't work you can experiment with different ratios available on the tool; they will each produce different values you can plug in to your stylesheets.

This is how I like to define the basic font size for the document. I rely on the fact that most browsers have a default font size of 16px; so I set the default value to 16px and then use `em` or `rem` units in other selectors to make sure that the sizing is relative to the parent element (when using em) and to the root element (with rem).

I've also chosen not to support IE6 and 7. If you do need to support those or older browsers, then Richard Rutter's How to Size Text in CSS provides additional guidance for your work.

```
body {
  font-size: 16px;
  line-height: 1.2em;
}
.bodytext p {
  font-size: 1em;
}
.sidenote {
  font-size: 0.75em;
}
```

Measure

Sometimes we allow the lines of text to become to wide; that makes them harder to read as the eye looses sight of where the line ends and when should the user return to the left edge of the screen. The horizontal distance is referred to as measure (also called line length).

So what should the maximum width a text block be? Well, it all depends on the size of the font. The larger the font size, the longer the line can be; that said you don’t want to go beyond 80 characters or 40ems (since block width is easier to measure than counting characters). We can ensure the text grows no wider than our specified width using the `max-width` css attribute. It looks like this:

```
body {
  max-width: 40em;
}
```

Leading

Leading (pronounced “ledding”) is so called because, in mechanical presses, strips of lead are placed between lines of type to space the lines apart. CSS handles leading through the `line-height` property.

Line-height is unique among CSS properties in that it doesn’t require a unit attached to its value, like in the example below the line-height is 1.25 times the value of the font size (18px in this case):

```
p {
  font-family: 'Minion Pro', 'Minion Web', serif;
  font-size: 16px;
  line-height: 1.25;
}
```

I prefer to use the long hand syntax for my CSS and explictly write all attributes separately. You could write the declaration above as:

```
p {
  font: 16px/1.25 'Minion Pro', 'Minion Web', serif;
}
```

Spacing/Line Height

One of the most common typographic “mistakes” I see on the web today is improper type spacing. What I’m referring to here is instances where a block text isn’t given enough margin, subheads and correlated body text which aren’t visually grouped together, and so on. Proper spacing (combined with hierarchy) allows the reader to scan the text and access it at the desired points.

It seems to me that the relationship of paragraph spacing (additional spacing placed before or after a paragraph), the space around a block of type, and letter spacing can be related proportionally to the line height of a paragraph. Line height is defined as the vertical distance between lines of text. So for instance, if the line height of one paragraph is set to 2em and a paragraph with the same size text is set to 1.5em, the first paragraph will require more paragraph spacing and probably more margin around it.

Much of this is done by eye rather than an exact formula, but I do use a good rule of thumb when it comes to the relationship of paragraph spacing to line height. When working with web content we need to make sure that we do equal spacing on both the top and the bottom margins, otherwise the first and last paragraphs will look weird and your content will not flush to the top, as you probably intended.

```
p {
  line-height: 1.5;
  margin-top: 1.5em;
  margin-bottom: 1.5em;
}
```

We can modify the styles for the first and last paragraphs to remove the top or bottom margin as needed using `:first-child` and `last-child` pseudo-classes. The code look lie this.

```
p:first-child {
  margin-top: 0;
}

p:last-child {
  margin-bottom: 0;
}
```

The final part of spacing that I consider is paragraph indentation. I don't intend the first paragraph, either I'm using a drop cap or want to follow convention and not indent.

The code uses and adjacent sibling selector to indent paragraphs that are next to each other. This will fail for the first and last paragraphs since there is no paragraph before/after it.

```
p {
  font-size: 1em;
}

p + p {
  text-indent: 2em;
}
```

Contrast

It may sound obvious that good type contrast is essential for readability but we always try to push the boundaries of contrast.

We forget that even sighted users may have problems with our “clever” light gray text on white background design assuming that what’s good for us is good for everyone or that everyone will understand the message.

The Web Content Accessibility Guidelines have the following to say about color and contrast.

Don’t use color as the only way to procide information or eliciting a response (Rule 1.4.1)

> Color is not used as the only visual means of conveying information, indicating an action, prompting a response, or distinguishing a visual element. (Level A)
>
> Rule 1.4.1

Use high contrast between text and background color, with few exception indicated in the rule.

> The visual presentation of text and images of text has a contrast ratio of at least 4.5:1, except for the following: (Level AA)
>
> - Large Text: Large-scale text and images of large-scale text have a contrast ratio of at least 3:1;
> - Incidental: Text or images of text that are part of an inactive user interface component, that are pure decoration, that are not visible to anyone, or that are part of a picture that contains significant other visual content, have no contrast requirement.
> - Logotypes: Text that is part of a logo or brand name has no minimum contrast requirement.
>
> Rule 1.4.3

When planning your site’s color pallete (font, links, headers, additional text colors) you can use tools like Lea Verou’s Contrast Ratio tool.

The other aspect is a matter of cultural awareness. In addition to avoiding using color as the only way to convey meaning we have to be mindful about conveying the right meaning. Colors have different meanings for different cultures.

The table below (From Creating Culturally Customized Content) shows what different colors mean in 5 different countries with vastly different cultures. It is obvious you're conveying a very different message to each of those audiences, even if they live in the same country.

COLOR	USA	China	India	Egypt	Japan
Red	Danger Love Stop	Good fortune Luck Joy	Luck Fury Masculine	Death	Anger Danger
Orange	Confident Dependable Corporate	Fortune Luck Joy	Sacred (the Color Saffron)	Virtue Faith Truth	Future Youth Energy
Yellow	Coward Joy Hope	Wealth Earth Royal	Celebration	Mourning	Grace Nobility
Green	Spring Money New	Health Prosperity Harmony	Romance New Harvest	Happiness Prosperity	Eternal life
Blue	Confident Dependability Corporate	Heavenly Clouds	Mourning Disgust Chilling	Virtue Faith Truth	Villainy
Purple	Royalty Imagination	Royalty	Unhappiness	Virtue	Wealth
White	Purity Peace Holy	Mourning	Fun Serenity Harmony	Joy	Purity Holiness
Black	Funeral Death Evil	Heaven Neutral High Quality	Evil	Death Evil	

Same font for all text or combined?

Does the font you're using work well for body and headings or will you have to combine it with another font that works better for headings and larger text. Not all fonts lend themselves to this dual usage so you'll have to be careful in your research to test the font or fots work as intended.

If you use specimens, particularly those that live in the web, use them to test visually if your font or fonts work well for your design.

Media queries are your friend

When we work with media queries we can change every detail of our content, including text layout and attributes. This way when you change the layout for smaller screens. You may also consider adjusting size, leading and other typography items to better suit the smaller size.

Non Design Considerations

There are other considerations outside design that must be taken into account when selecting fonts… as much as I hate to admit it sometimes licensing or cost will drive me away from a font that, otherwise, would be awesome for the project.

These are some of the non-design things that come to mind.

Have you set your viewport correctly?

The viewport meta tag tells the browser how to render the page. This is required for responsive sites (and I'm assuming you're working on responsive sites, right?!)

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

This will likely render the width of the page at the width of its own screen. So if that screen is 320px wide, the browser window will be 320px wide, rather than way zoomed out and showing 960px or whatever that device does by default.

Do not, I repeat, **do not** use this meta tag if your site is not responsive. The results are unpredictable but ugly. You've been warned.

Are you subsetting your fonts? Pros and cons.

One way to make your fonts files smaller is to subset them. When you subset a font you take out all the characters you're not using for your content. Depending on the font this may reducen the file size significantly at the risk of having to redo the work every time you add content to your site.

I will not discuss subsetting in detail. I've written about it in Subsetting Fonts and More on Font Subsetting.

To load Roboto Regular with only the Basic Latin character range look like this.

```
@font-face {
  font-family: 'Ubuntu'; 'Ubuntu'/* regular */url('Ubuntu-R-webfont.woff2
  font-weight: normal;
  font-style: normal;
  unicode-range: U+0020-007f;
}
') format('
```

One thing that you need to be mindfult of when/if you subset your fonts is to make sure that you keep all the characters you will need for your content to render with the chosen font.

In the case of English using the first 127 characters (equivalent to the ASCII encoding standard) this is usually not a problem. However, if you're working with more than one language (like English and Spanish or English and Swedish) you will have to make sure that you subset the additional characters that you will need, like opening question (¿) and exclamation mark (▯) or some of the Diacritics and accents needed (like ñ in Spanish or ö in Swedish). These additional characters are distinct Unicode codepoints and need to be included in your subset for it to work.

Does the font have all the weights and styles that you need?

Related to the necessary glyphs are the necessary weights for each font that you use in your page. Ideally we'd use 4 files representing regular, bold, italic and bold italic. We want to avoid the browser creating "faux" synthetic styles that will adversely impact the way your content look.

In Say No to Faux Bold Alan Stearns shows what the impact of faux styles (bold in particular) can have in your content.

If you want more indepth information, check How to use @font-face to avoid faux-italic and bold browser styles. It goes deeper into the technical details than Alan's article and it's geared for a more technical audience.

Creating speciments of the fonts you use

An interesting idea is to start building specimen libraries for the fonts you have used in projects or plan to use in projects. If you have a good specimen template wil;l definitely help you in your design process.

In Real Web Type in Real Web Context Tim Brown states the neeed explicitly: "I need to know how my type renders on screens, in web browsers".

Depending on your needs you could use the W3C's element sampler and style the bare content with your styles and fonts.

Another option is to use Tim Brown's [http://webfontspecimen.com/](Web Font Specimen) to create more advanced specimens. I've created specimens for fonts with open sources licenses at Font Specimen Archive

Have you tested how will your fallback fonts affect your design?

So far we've made the assumption that our fonts will load and everything works well out of the box. But this is not always the case and we shouldn't assume it is.

This begs the question: ***How well will the fallback fonts work with your design***

If the fallback has a higher or lower x-height or thicker strokes than your web font, the layout will look different. This is why you must test your stack during development and make sure that your web fonts match as close as possible with the fallbacks. Monica Dinculescu's Font Style Matcher can help

Hosting locally or using a font service?

Are you hosting your fonts with a service like Google Fonts, Adobe Typekit or other font hosting providers or are you hosting your fonts in your own site?

If you choose to host fonts in your own site you will have to contend with network congestions and having your server become a bottleneck for your users. Yet it may be the only way to comply with some fonts restrictive licensing terms and conditions; as we'll discuss in the next section be very careful about the licensing of your font.

These are some of the font services I've used in the past.

- Google Fonts
- Adobe Typekit
- Fonts.com Web Fonts
- Typotheque

As with anything else on the web, your milleage may vary.

Do you have the right license to use the font on the web?

The last, and perhaps most important, non-design aspect of selecting and using fonts on the web is the licensing. Rightly or not some foundries are very restrictive in the terms of their license and some will release their fonts under open source licenses like the SIL OFL or Apache (seen in some early fonts from Google and other companies for whom liability may be an issue).

For most foundries, fonts are software users license rather than a product you can own. This has implications if you want to modify the fonts or want to convert them to other formats. Some foundries will go as far as block usage of Font Squirrel's Webfont Generator with their fonts.

These problems with Foundries lead me to prefer comissioning fonts specifically for a project. If that is not possible my next option is to choose Open Source Fonts and, only if I have no choice, I will work with foundries based on past experience or expressed instructions from the client.

This is a step we often overlook but is essential to cover yourself from liability.

Writing Modes and World Languages

Another thing that affects typography is the writing direction of your text. Either because the language requires it or because you're experimenting with new technologies like Jen Simmons does in this example from Layout Land Youtube Channel

It leverages CSS Grid, Transforms, and other modern technologies to create really impressive layouts.

Going back to writing modes. Different languages use different writing modes. Some things to consider:

- Most Latin and Cyrilic languages run the text from left to right and top to bottom
- Arabic and Hewbrew run the text from right to left and top to bottom
- Japanese is a special case
    - Japanese can run the text from left to right, top to bottom
    - Japanese can also run from top to bottom **and** right to left
    - Both writing modes for Japanese can be used in the same page

There are other languages and considerations but you get the idea.

But you can also use it for creative typographical effects that require CSS, not images or CSS translate, to accomplish the goal. A good, and subtle example is the “Rise To Success” text in the pen below:

If you look at the CSS code in the embedded pen, you'll see the following code:

```
h2 {
  writing-mode: vertical-rl;
  float: left;
  margin: 1.5rem 0 0 -3.8rem;
  font-size: 1.8em;
  background: #96e5fb;
  padding: 8px 0;
}
```

It's the `writing-mode: vertical-rl;` attribute that makes the text rotate while still allowing us to highlight it and keeping it in the document to be read by assistive technology.

UN Website in Eglish UN Website in Arabic

Figure 3: United Nations Website in English (top) and Arabic

(bottom). Notice how they are mirrors of each other.

Jen wrote an article for 24 ways in 2016, where she provides a thorough explanation of writing modes and how to make them work in your projects, today.

font-variant: Low Level Plumbing

CSS offers different levels of control over font features available from your font. The prefered way is to use individual `font-variant-*` attributes in a selector or use the shorthand `font-variant`.

Not all fonts providde all the features discussed in this section. As always research whether the chosen font or fonts have the features that you need.

This may be a good thing to include in your font specimen, if you have one.

The code looks for the shorthand looks like this:

```css
body {
  font-variant: common-ligatures annotations() slashed-zero;
}
```

The different values for the property are:

normal
: Specifies a normal font face; each of the longhand properties has an initial value of normal.

none
: Sets the value of the font-variant-ligatures to none and the values of the other longhand property as normal, their initial value.

<common-lig-values>, <discretionary-lig-values>, <historical-lig-values>, <contextual-alt-values>
: Specifies the keywords related to the font-variant-ligatures longhand property. The possible values are: common-ligatures, no-common-ligatures,

discretionary-ligatures, no-discretionary-ligatures, historical-ligatures, no-historical-ligatures, contextual, and no-contextual.

stylistic(), historical-forms, styleset(), character-variant(), swash(), ornaments(), annotation()
: Specifies the keywords and functions related to the font-variant-alternates longhand property.

small-caps, all-small-caps, petite-caps, all-petite-caps, unicase, titling-caps
: Specifies the keywords and functions related to the font-variant-caps longhand property.

<numeric-figure-values>, <numeric-spacing-values>, <numeric-fraction-values>, ordinal, slashed-zero
: Specifies the keywords related to the font-variant-numeric longhand property. The possible values are: lining-nums, oldstyle-nums, proportional-nums, tabular-nums, diagonal-fractions, stacked-fractions, ordinal, and slashed-zero.

<east-asian-variant-values>, <east-asian-width-values>, ruby
: Specifies the keywords related to the font-variant-east-asian longhand property. The possible values are: jis78, jis83, jis90, jis04, simplified, traditional, full-width, proportional-width, ruby.

We can also use these variables individually. The individual names are:

- font-variant-ligatures
- font-variant-alternates
- font-variant-caps
- font-variant-numeric
- font-variant-east-asian
- font-variant-position

The code using individual properties looks like this:

```
body {
  font-variant-ligatures: common-ligatures;
  font-variant-alternates: historical-forms;
  font-variant-numeric: slashed-zero;
}

.japanese {
  font-variant-east-asian: ruby full-width jis83;
}

.small {
  font-variant-caps: small-caps;
}

.sup {
  font-variant-position: sub;
}

.super {
  font-variant-position: super;
}
```

Performance: FontFace Observer and font-display

Whenever I need to make sure that a font has loaded before using it I work with Fontface Observer to load the fonts with a good fallback and timeouts.

> The process consists of the following sections:
>
> - Font Loading
> - Font Use
> - Javascript Loader

We first define our fonts in CSS like normal. We use the same declarations as we do normally to load fonts.

```
/* Regular font */
@font-face {
  font-family: 'notosans';
  src: 'notosans'url('../fonts/notosans-regular.woff2')off2'), url('../fo
  font-weight: normal;
  font-style: normal;
```

```
}
/* Bold font */
@font-face;
  src: url('../fonts/notosans-bold.woff2') format('woff2'), url('../fonts/
  fo'truetype'url('../fonts/notosans-bold.ttf')Italic Font */
@font-face {
  font-family: 'notosans';
  src: url'notosans'/* Italic Font */
@font-facemat('woff2'), url('../fonts/notosans-'woff2'url('../fonts/notos
      format('woff'), url('../fonts/notosans-italic.ttf')nt */
@font-face {
  font-family: 'notosans';
  src: url('../fonts/n'notosans'/* bold-italic font */
@font-face url('../fonts/notosans-bolditalic.woff')
      format('woff'), url('../fonts/notosans-bolditali'../fonts/notosans-b
      format('woff'), url('../fonts/notosans-bolditalic.ttf') format('true
  font-weight: 700;
  font-style: italic;
}
```

Next we prepare three versions of the default element styles. One for when there is no Javascript (body), one for when the fonts fail to load (`.fonts-failed body`) and one for when the fonts load successfully (`.fonts-loaded body`). Only one of these body declarations will be used for the page.

```
/* Default body style */
body {
  font-family: Verdana, sans-serif;
  font-size: 16px;
  line-height: 1.275;
  -webkit-text-decoration-skip: ink;
  -moz-text-decoration-skip: ink;
  -ms-text-decoration-skip: ink;
  text-decoration-skip: ink;
}

/*
```

```css
    This will match if the fonts failed to load.
    It is identical to the default but doesn't
    have to be
*/
.fonts-failed body {
  font-family: Verdana, sans-serif;
  font-size: 16px;
  line-height: 1.375;
  -webkit-text-decoration-skip: ink;
  -moz-text-decoration-skip: ink;
  -ms-text-decoration-skip: ink;
  text-decoration-skip: ink;
}
/*
    This will match when fonts load successfully
*/
.fonts-loaded body {
  font-family: notosans-regular, verdana, sans-serif;
  font-size: 16px;
  line-height: 1.375;
  -webkit-text-decoration-skip: ink;
  -moz-text-decoration-skip: ink;
  -ms-text-decoration-skip: ink;
  text-decoration-skip: ink;
}
```

The final piece is the Javascript file that will actually load the fonts. This assumes that `fontfaceobserver.js` has already been loaded.

We first define a constant for each of the fonts we want to load. We use the same name but add a second attribute to the `FontFaceObserver` object to indicate additional information about the font (weight and style)

we assign `document.documentElement` to a variable that we will work with later in the script.

We add the class `fonts-loading` to document element as a temporary place holder while we download the font.

Next we use `promise.all` to create an array of promises with each font's load method. Promise.all is an atomic function, either they will all succeed or the will all fail. This will help us make sure that all fonts are available.

If the fonts are successful the `then` branch is followed. This branch will remove the `fonts-loading` class and replace it with `fonts-loaded`. This is the CSS class that uses the web font we just downloaded and it will only be used if the fonts loaded successfully.

If the fonts fail to load the script follows the `catch` path. This path replaces `fonts-loading` with `fonts-failed`. This CSS class doesn't use the web font and is essentially identical to the body element definition.

```
const sans = new FontFaceObserver('notosans', {
  weight: normal,
  style: normal
});
const italic = new FontFaceObserver('notosans', {
  weight: normal,
  style: 'italic'
});
const bold = new FontFaceObserver('notosans', {
  weight: 700,
  style: 'normal'
});
const bolditalic = new FontFaceObserver('notosans', {
  weight: 700,
  style: 'italic'
});

let html = document.documentElement;

html.classList.add('fonts-loading');

Promise.all([sans.load(), bold.load(), italic.load() bolditalic.load()]).
  html.classList.remove('fonts-loading');
  html.classList.add('fonts-loaded');
}).catch(() =>{
  html.classList.remove('fonts-loading');
  html.classList.add('fonts-failed');
});
```

Yes, this is more work but think about it. You're already loading the fonts and we could optimize the loader script to use only two elements (body and .fonts-loaded). The only new things we do is load `fontfaceobserver.js` and run our loader script.

Another thing we can add to `@font-face` declarations to speed up font loading resolition is the `font-display` rule. The rule tells browsers how would you like it to handle loading web fonts.

The possible values are:

- **auto**: The default. Typical browser font loading behavior will take place. This behavior may be FOIT, or FOIT with a relatively long invisibility period. This may change as browser vendors decide on better default behaviors
- **swap**: Fallback text is immediately rendered in the next available system typeface in the font stack until the custom font loads, in which case the new typeface will be swapped in. This is what we want for stuff like body copy, where we want users to be able to read content immediately
- **block**: Like FOIT, but the invisibility period persists indefinitely. Use this value any time blocking rendering of text for a potentially indefinite period of time would be preferable. It's not very often that block would be preferable over any other value
- **fallback**: A compromise between **block** and **swap**. There will be a very short period of time (100ms according to Google) that text styled with custom fonts will be invisible. Unstyled text will then appear if the custom font hasn't loaded before the short blocking period has elapsed. Once the font loads, the text is styled appropriately. This is great when FOUT is undesirable, but accessibility is more important
- **optional**: Operates like **fallback** in that the affected text will initially be invisible for a short period of time, and then transition to a fallback if font assets haven't completed loading. The optional setting gives the browser freedom to decide whether or not a font should even be used, and this behavior depends on the user's connection speed. If you use this setting you should anticipate custom fonts may possibly not load at all

So, depending on the importance of the font to the layout and ease of read of the site you can play with the different values for `font-display` to see how it affects your site. Since you're likely to have a high speed connection and not throtle it it's important to test the site in your target devices.

Loading a font using `@font-face` and `font-display` looks like this:

```
@font-face {
  font-family: 'Ubuntu'; 'Ubuntu'/* regular */url('Ubuntu-R-webfont.woff2
  font-weight: normal;
  font-style: normal;
  font-display: swap;
}
') format('
```

I will not go into details on why testing on devices is important, I'll just leave you, again, with Alex Russell's video on web performance

One thing to keep in mind is that your fonts are subject to the web's [same origin policy](). This means that unless you configure your server with universal CORS access or you serve fonts from a CDN like Google Fonts or Typekit they will not load across different origin.

HTTP2 and Preload

We cam also tackle performance issues from the server side. I’m not talking about server side rendering but to use HTTP/2.

HTTP2 allows several requests to use the same network connection, reducingthe overhead of several individual requests significantly and makes inlining obsolete.

Browser support for HTTP/2 (and its predecessor SPDY) is excellent, so there’s no reason not to use HTTP/2.

We can preload resources from the server Using Apache as an example we preload assets when the browser loads index.html. We’re preloading both **woff** and **woff2** fonts to make sure cover modern browsers that will support either version. If we must support older browsers we should also push the **ttf** version of the font.

```
<If "%{DOCUMENT_URI} == '/index.html'">
  H2PushResource add css/site.css
  H2PushResource add js/site.js
  H2PushResource add font/font.woff2
  H2PushResource add font/font.woff
</If>
```

We can also customize what resources we push based in the URI of the resource. In the following example each time we match a URI we will load specific assets for that file and nothing else. We could also have a wildcard match that will load assets needed by **all** pages and use the system below for page specific assets.

```
<if "%{DOCUMENT_URI} == '/portfolio/index.html'">
  H2PushResource add /css/dist/critical-portfolio.css?01042017
</if>

<if "%{DOCUMENT_URI} == '/code/index.html'">
  H2PushResource add /css/dist/critical-code.css?01042017
</if>
```

Nginx also allows you to push resources to the browser. The same examples reworked for nginx. The first one will preload a set of resources.

```
server {
  location = /index.html {
    http2_push /css/style.css;
    http2_push /js/main.js;
    http2_push font/font.woff2;
    http2_push font/font.woff;
  }
}
```

And the second example pushing assets depending on the page we're trying to access:

```
location = /portfolio/index.html {
  http2_push /css/dist/critical-portfolio.css?01042017;
}

location = /code/index.html {
  http2_push /css/dist/critical-code.css?01042017;
}
```

If you don't have access to your server's configuration, don't want to depend on manually updating the cache busting string you can do the preload from the client side using link elements with the `preload` attribute.

```
<link rel="preload" href="https://example.com/fonts/font.woff2"
  as="font" crossorigin type="font/woff2">
<link rel="preload" href="https://example.com/fonts/font.woff"
  as="font" crossorigin type="font/woff">
<link rel="preload" href="https://example.com/css/main.css"
  as="style" crossorigin type="text/css">
<link rel="preload" href="https://fonts.example.com/js/site.js"
  as="script" crossorigin type="text/javascript">
```

The attributes of the link are:

- **rel** - the type of link it is. In this case the value is `preload`
- **href** – the URL to preload
- **as** – the destination of the response. This means the browser can set the right headers and apply the correct CSP policies.
- **crossorigin** – Optional. Indicates that the request should be a CORS request. The CORS request will be sent without credentials unless you add `crossorigin="use-credentials"` to the link
- **type** – Optional. Allows the browser to ignore the preload if the provided MIME type is unsupported.

I discuss link preloading along with other HTTP2 resource pushing and preloading strategies in HTTP/2 Server Push, Link Preload And Resource Hints

Service Worker Support

Service Workers are the core of progressive web applications. They work as a reverse network proxy that intercepts request for your site and performs actions based on its configuration. I've written about service workers in my blog before so I won't go into detail.

I will use workbox.js version 3, currently in beta, to illustrate how to cache fonts. You will most definitely want to add additional routes and caching strategies for your site.

At the root of your site use the following snippet inside a script tag to register the service worker.

We test if the `navigator` object has a serviceWorker method. If it does it means that Service Workers are supported and we can register it. If it doesn't then Service Workers are not supported and we bail accordingly .

Registering the Service Worker means that it'll work for all pages under its scope but not above it (This is why we put the service worker at the root of the application).

```
if ('serviceWorker' in navigator) {
  navigator.serviceWorker
    .register('sw.js')
    .then(function(registration) {
      console.log(
        'Service Worker registration successful with scope: ',
        registration.scope
      );
    })
    .catch(function(err) {
      console.log('Service Worker registration failed: ', err);
    });
}
```

The actual Service Worker script is fairly simple.

We import `workbox-sw`, the core of our Service Worker.

We check if Workbox loaded successfully and if it does then we register a route matching all possible font types and create a custom cache using a cache-first strategy (check the cache and if the resource is not there then fetch it from the network).

The cache will store 10 fonts for 30 days (as indicated in `maxEntries` and `maxAgeSeconds`). If more than 10 fonts are added the oldest will be removed first.

```
importScripts(
  'https://storage.googleapis.com/workbox-cdn/releases/3.0.0-beta.0/workbo
);

if (workbox) {
  workbox.routing.registerRoute(
    /.*\.(?:woff2,woff,ttf,otf,eot)/,
    workbox.strategies.cacheFirst({
      cacheName: 'font-cache/.*\.(?:woff2,woff,ttf,otf,eot)/workbox.expir
          maxEntries: 10,
 'font-cache'AgeSeconds: 30 * 24 * 60 * 60
        })
      ]
    })
  );
} else {
  console.log(`Boo! Workbox didn't load`);
}
```

Using a Service Worker to cache fonts using this method means that fonts will be loaded from cache in second and subsequent visits and when the browser is offline or connectivity is unreliable.

We could have precached the fonts but that would remove the possibility of customizing the cache. The size of fonts may also impact how long does it take to precache resources and the whole idea of precaching is to make the first load of the page work fast.

For more information, check Workbox 3 documentation.

Links and Resources

- General Information
  - What Is Beautiful Typography
  - The Experimental Layout Lab of Jen Simmons
  - Fonts and Layout for Global Scripts
- Typography
  - How to use @font-face to avoid faux-italic and bold browser styles
  - Say No to Faux Bold
- System Fonts
  - Shipping system fonts to GitHub.com
  - Using UI System Fonts In Web Design: A Quick Practical Guide
  - OS Specific Fonts in CSS
  - System Font Stack
  - Implementing system fonts on Booking.com — A lesson learned
- Choosing your font stack
  - Choosing Web Fonts: A Beginner's Guide
  - Font Family Reunion
  - Font style matcher
- `font-variant-*`
  - font-variant-* at MDN
- Performance
  - A Comprehensive Guide to Font Loading Strategies
  - 3 Tips for Faster Font Loading
  - https://css-tricks.com/font-display-masses/

- Media Queries
    - Using Media Queries For Responsive Design In 2018
- HTTP2
    - Specification
    - Wikipedia Entry
    - Can I Use HTTP2
- Service Workers
    - The offline cookbook
    - Jeremyt Keith's My first Service Worker
    - Making Resilient Web Design work offline
    - Service Worker notes
    - Making A Service Worker: A Case Study
    - Workbox 3 Beta
- Variable fonts
    - How to use variable fonts in the real world
    - Typographic Potential of Variable Fonts
    - Variable Fonts on the Web
    - Variable fonts for the win!
    - Variable Fonts Demo and Explainer
    - How to use variable fonts in the real world
    - How to use variable fonts in the real world
    - One File, Many Options: Using Variable Fonts on the Web
    - Variable fonts Codepen demos by Jason Pamental
    - Variable fonts Codepen demos by Mandy Michael
    - New variable fonts from Adobe Originals
- Document Order and Visual Order
    - HTML Source Order vs CSS Display Order
    - WCAG C27: Making the DOM order match the visual order
    - A Few Different CSS Methods for Changing Display Order
- Font Subsetting
    - unicode-range CSS descriptor
    - Unicode Character Ranges
    - How to subset fonts with unicode-range
    - Creating Custom Font Stacks with Unicode-Range
- Font Specimens
    - Real Web Type in Real Web Context
- Books
    - Tim Brown, Combining Typefaces
    - Cyrus Highsmith, Inside Paragraphs
    - Jason Santa Maria, On Web Typography
    - Robert Bringhurst, The Elements of Typographic Style

- Bram Stein, Webfont Handbook
- Richard Rutter, Web Typography
- Richard Rutter, The Elements of Typographic Style Applied to the Web
- Donny Truong, Professional Web Typography

Credits