Project 2: Header HTML

<!DOCTYPE html>

<head>

</head>

<body>

<header>

<img src="/assets/jeff.png">

<h1>Jeff's Blog</h1>

<ul>

<li><a href="#">About Me</a></li>

<li><a href="#">Best Poems</a></li>

<li><a href="#">Worst Poems</a></li>

</ul>

</header>

</body>

CSS Implementation: Rather than using the <style> tag, the <link> tag lets you include CSS from an external file- rather than keeping it all written out in your HTML file. This is crucial when you work on larger projects, with thousands of lines of code.

In the <head> tag, write <link href=”/normalize.css” rel=”stylesheet”>

The link tag always needs two attributes:

1. href –a URL where the CSS file lives
2. rel – which should always be set to “stylesheet”

What is normalize.css?

It is a CSS file that, according to its website, makes browsers “render all elements more consistently and in line with modern standards”. That way your page will look the same, no matter what browser you use. It’s a good idea to include this file in all your web projects.

NAVIGATION STYLES

To get started, let’s add a style tag and center all the text in the header

<head>

<link href="/normalize.css" rel="stylesheet">

<style>

header {

text-align: center;

}

ul {

padding: 10px;

}

</style>

To make all the links line up next to each other in a row:

li {

display: inline;

}

What does “display” do?

Every HTML element has a default value for display. Usually it’s either “block” or “inline”.

Block: Elements like headings, paragraphs, and list items display “block” by default, which means they stretch the *whole width of the page* and have *line breaks* before and after.

Inline: Links (the <a> tag) and a few other elements we haven’t learned about yet display ”inline” by default, which means they exist within the normal flow of the text they’re contained within- no *line breaks*, no taking the *whole width of the page*.

To position the padding of the links and make them further apart:

padding: 0px 10px 0px 10px;

When you pass padding just one value, as we’ve done before, (e.g. padding: 10px;) then you are telling the browser to add 10px of space on *all sides* of that element.

To control each side’s padding individually, simply give padding 4 values, separated by spaces:

Padding: top right bottom left

HEADER STYLES

After the header tag’s styles (just text-align: center; for now), add the background image and adjust the sizing.

Header {

Text-align: center;

Background: url(<http://dash.ga.co/assets/jeff-bg.png)>;

Background-size: cover;

Color: white;

If you want to change a link’s colour, you have to set it explicitly. Unlike headings and paragraphs, it won’t inherit from its parent element.

a {

color: white;

}

To make the headline standout in the visual hierarchy, we can make it bigger:

H1 {

Font-size: 70px;

}

To make some space between the top of the page and the image:

Img {

Margin: 40px 0px 0px 0px;

}

Margin creates space on the outside of an element, padding creates space on the inside.

Let’s make it pop more by adding a big white border and rounding the corners:

Img {

Margin: 40px 0px 0px 0px;

Border: 7px solid white;

Border-radius: 20px;

}

Make the background of the navigation bar black:

Ul {

Padding: 10px;

Background: black;

}

Part 3: Responsive design and Javascript

The goal of responsive design is to make your website look great no matter which device you’re on. Some special CSS tricks let us make our content adapt gracefully to different screen widths. Start by improving the structure of our content.

Wrap an <article> tag around each blog post like this:

<article>

<h2>VHS umami pop-up trust fund</h2>

<p>Marfa church-key kitsch bicycle rights, 8-bit mixtape cardigan gentrify Echo Park. Street art swag brunch, next level roof party Schlitz hella organic keffiyeh selfies. You probably haven't heard of them polaroid hashtag +1, meggings biodiesel Portland High Life cray tumblr retro.</p>

</article>

<article>

<h2>Sartorial synth Echo Park, roof party</h2>

<p>chambray you probably haven't heard of them pour-over viral selvage umami skateboard VHS post-ironic selfies. Wes Anderson gentrify fanny pack twee, bicycle rights bitters blog keffiyeh plaid flannel. Tonx irony cliche sustainable mlkshk bitters. Four loko leggings chambray Vice.</p>

</article>

<article>

<h2>Forage food truck keytar master cleanse</h2>

<p>ethical thundercats sustainable locavore quinoa Neutra. Aesthetic tacky sweater single-origin coffee, bicycle rights organic lo-fi street art american apparel ennui four loko ethnic Brooklyn small batch. Forage YOLO polaroid</p>

</article>

</body>

The article tag lets us group together multiple HTML elements that form a single piece of content.

If the posts stretch the whole width of the page, you can make them narrower and centred so they’re easier to read. Make a new style for the article tags, then set the width and padding:

Article {

Width: 500px;

Padding: 20px;

}

The post is still aligned to the left of the screen. Fix this by adding margin: 0 auto; to the article’s style.

Article {

Width: 500px;

Padding: 20px;

Margin: 0 auto;

}

Margin controls the amount of space between the outside of an element and the other elements around it. So when we set margin to “0 auto”, it has 0 margin on the top and bottom, and “auto” margin on the left and right. Since both the left and right’s margin stretch all the way to the edge of the page (that’s what *auto* means), the effect is that the blog post is centred in the page.

To make it responsive replace “width: 500px;” with “max-width: 500px;” for the article

Using “max-width” instead of “width” means our article elements can be smaller than 500px, but not any larger.

When the browser is really small, the title and navigation look kind of messed up. We can fix this using a media query, a technique that allows us to set CSS styles that only activate when the browser is a certain width.

The syntax for media queries looks like this

@media (max-width: 500px) { 🡪 this is basically setting a condition- in this case that the browser should be smaller than 500px- and when that condition is true the CSS inside it gets activated.

Article {

Max-width: 500px;

Padding: 20px;

Margin: 0 auto;

}

@media (max-width: 500px) {

body {

background: red;

}

}

This makes the body’s background red if the browser width is smaller than 500px.

STYLES FOR NARROWER WIDTHS

The heading font should be smaller when the browser is small:

@media (max-width: 500px) {

h1 {

font-size: 36px;

}

li {

display: block;

padding: 5px;

}

}

The navigation links should sit on top of each other, rather than inline… see above.

ADVANCED COLOURS

Hex colour codes can improve the page.

Change the unordered list (ul) element’s background from black to #f00 (red)

Anatomy of a Hex Colour Code

# Hex colours always start with a hash like this

The first character controls the amount of redness. It ranges from 0 (no red at all) to F (100% red).

The second character controls the amount of green-ness. Just like the first character, it ranges from 0 to F.

The third character controls the amount of blueness. Together, they are RGB.

Hex codes work by virtually “mixing” the three primary colours.

For example, #000 is black because there is no colour, and #FFF is white because all the colours are present. #F00 is red because red is 100% and the other colours are at 0%.

Hex codes aren’t the only way to do colour in CSS. Another popular method is rgba(). This lets you add alpha (transparency) to the mix.

ul {

padding: 10px;

background: rgba(0, 0, 0, 0.5);

RGBA colours are a lot like hex colours with a slightly less confusing syntax:

Rgba(255, 255, 255, 1)

Instead of a 0-F scale, RGBA colours use a 0-255 scale, and the last digit (transparency) uses a 0-1 scale. The example above would make white.

0,0,0 Black

255,0,0 red

0,255,0 green

0,0,255 blue

255,255,255 white

0,255,255, cyan

255,0,255, magenta

255,255,0 yellow

Javascript

This can make your webpages interactive

Our goal is to make a “like” button for Jeff’s blog, so he can easily gauge the popularity of each poem. For now it won’t actually save any data, we’d need to program a back-end server to do that, but we will make a button that reacts when clicked.

Before we write any javascript to add interactivity, let’s make the button itself using html.

Inside each article at the bottom, right before the closing </article> tag, write this: <button>Like</button>

We can include Javascript in our site similar to the way we include CSS, by using a special tag. Instead of style, we use a <script> tag.

Write <script></script> at the bottom, just above the </body> tag.

To check and make sure Javascript is working, let’s write a simple script to pop up an alert when the page loads.

<script>alert("Javascript works!")</script>

Now let’s try to make that happen when you click one of the buttons:

Every time you move your mouse, click something, mouse over something, press a key or scroll, your browser fires off an *event*. By default, these events go unnoticed. But we can use Javascript to *listen* for specific events and take action when they happen. In this case, we want to listen for the *click* event on the *button* element.

Here’s an example of some Javascript that would listen for a click event on a button element:

$(“button”).on(“click”, function() {

alert(“clicked!”)

});

$("button").on("click",function(){ alert("clicked!") })

This code uses jQuery, a popular library of useful javascript code that, among other things, make it easy to select elements and listen for events.

Write it out inside your script tag exactly.

How this javascript works:

First we select the element(s) whose events we want to listen to. This part is very similar to the way we select elements in CSS, except the selector goes inside quotes and parentheses with a $ to the left of it🡪 $(“button”)

For example, you could select paragraphs with $(“p”) or headings with$(“h1”)

Next, we call the on() function, which sets up an event listener for our button element. Inside it’s parentheses are two options – sometimes called *parameters* or *arguments*.

$(“button”).on(“click”, function() { alert(“A button was clicked!”) });

Notice how the basic format is $(element).on(event-type, thing-to-be-done);

The first parameter is the type of event to listen to. In this case it’s “click”, but we could also listen for hover, scroll, etc. It must be in quotes, because it’s a string, which basically means it’s text (as opposed to numbers, or lists, or functions).

$(“button”).on(“click”, function() { alert(“A button was clicked!”) });

The second parameter is a function containing the thing to be done when our event happens. In this case, we’ll pop up an alert with the text “A button was clicked!”, but you can put any text you like inside there. *Functions* group together chunks of code and allow them to be executed at a later time.

<!DOCTYPE html>

<head>

<link href="/normalize.css" rel="stylesheet">

<style>

header {

text-align: center;

background: url('http://www.worldofwanderlust.com/wp-content/uploads/2018/10/Paris.jpeg');

background-size: cover;

color:white;

}

a {

color: white;

}

h1 {

font-size: 75px;

}

img {

margin: 50 50px 50px 50px;

border: 8px solid black;

border-radius: 30px;

}

ul {

padding: 20px;

background: rgba(20,20,0,0.5);

}

li {

display: inline;

padding: 0px 10px 0px 10px;

}

article {

max-width: 500px;

padding: 20px;

margin: 0 auto;

}

@media (max-width: 500px) {

h1 {

font-size: 30px;

padding: 5px;

}

li {

padding: 5px;

display: block;

}

}

</style>

</head>

<body>

<header>

<img src="https://cdn130.picsart.com/237776213020202.jpg?c256x256">

<h1>Beth's Blog</h1>

<ul>

<li><a href="#">About Me</a></li>

<li><a href="#">Favourite Products</a></li>

<li><a href="#">Disappointing Products</a></li>

</ul>

</header>

Google css selectors

Css tricks

Use **justify-content** again to help these frogs get to their lilypads. Remember that this CSS property aligns items horizontally and accepts the following values:

* **flex-start**: Items align to the left side of the container.
* **flex-end**: Items align to the right side of the container.
* **center**: Items align at the center of the container.
* **space-between**: Items display with equal spacing between them.
* **space-around**: Items display with equal spacing around them.

Now use **align-items** to help the frogs get to the bottom of the pond. This CSS property aligns items vertically and accepts the following values:

* **flex-start**: Items align to the top of the container.
* **flex-end**: Items align to the bottom of the container.
* **center**: Items align at the vertical center of the container.
* **baseline**: Items display at the baseline of the container.
* **stretch**: Items are stretched to fit the container.

The frogs need to get in the same order as their lilypads using **flex-direction**. This CSS property defines the direction items are placed in the container, and accepts the following values:

* **row**: Items are placed the same as the text direction.
* **row-reverse**: Items are placed opposite to the text direction.
* **column**: Items are placed top to bottom.
* **column-reverse**: Items are placed bottom to top.

Notice that when you set the direction to a reversed row or column, start and end are also reversed.

Notice that when the flex direction is a column, **justify-content** changes to the vertical and **align-items** to the horizontal.

Sometimes reversing the row or column order of a container is not enough. In these cases, we can apply the **order** property to individual items. By default, items have a value of 0, but we can use this property to set it to a positive or negative integer value.

Use the **order** property to reorder the frogs according to their lilypads.

justify-content: flex-end;

flex-direction: row-reverse;