

SESSION THREE  
SEPTEMBER 12, 2023

# INTRO TO CSS

ENDLESS APPRECIATION  
TO ERIC LI +  
MICHAEL FEHRENBACH  
FOR THE MATERIAL

# AGENDA

- > Some changes to the schedule
- > A note about late work
- > A note on file nomenclature
- > HTML questions
- > CSS
- > Exercises
- > Homework

# AGENDA

- > Some changes to the schedule
- > A note about late work
- > A note on file nomenclature
- > HTML questions
- > CSS
- > Exercises
- > Homework

# CHANGES TO THE SCHEDULE

8/29

The Internet

9/05

HTML

9/12

CSS

9/19

Layouts

9/26

Web Hosting  
+ Responsive  
Design

10/3

Advanced  
Layouts

10/10

Even More  
CSS

10/17

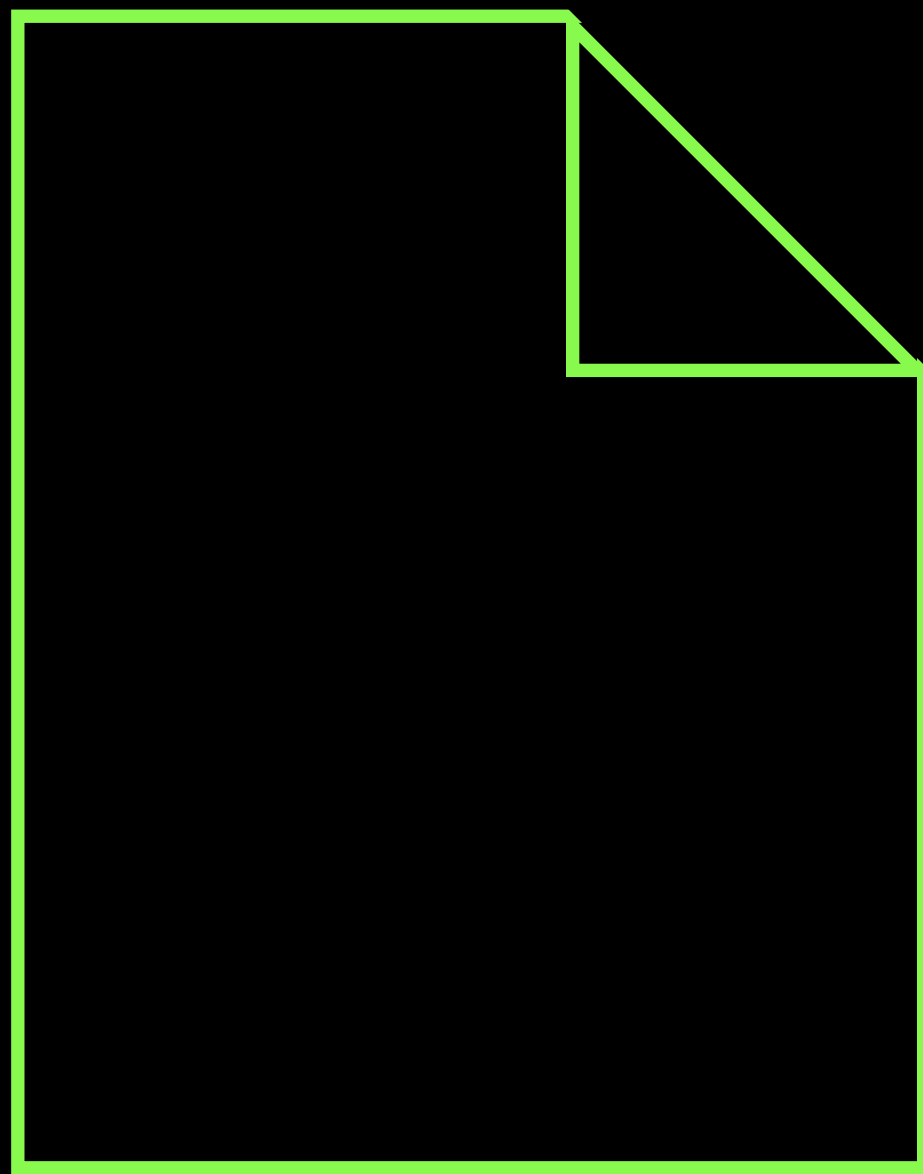
Midterms

# CHANGES TO THE SCHEDULE

For 10-17 (Session 8), you will have:

- > 5 Harmonic Collection entries **completed and revised**
- > Your Harmonic Collection hub that links the pages together
- > A short written response (more on this later)
- > A brief presentation to share with the class (more on this later)

# A NOTE ON FILE NOMENCLATURE



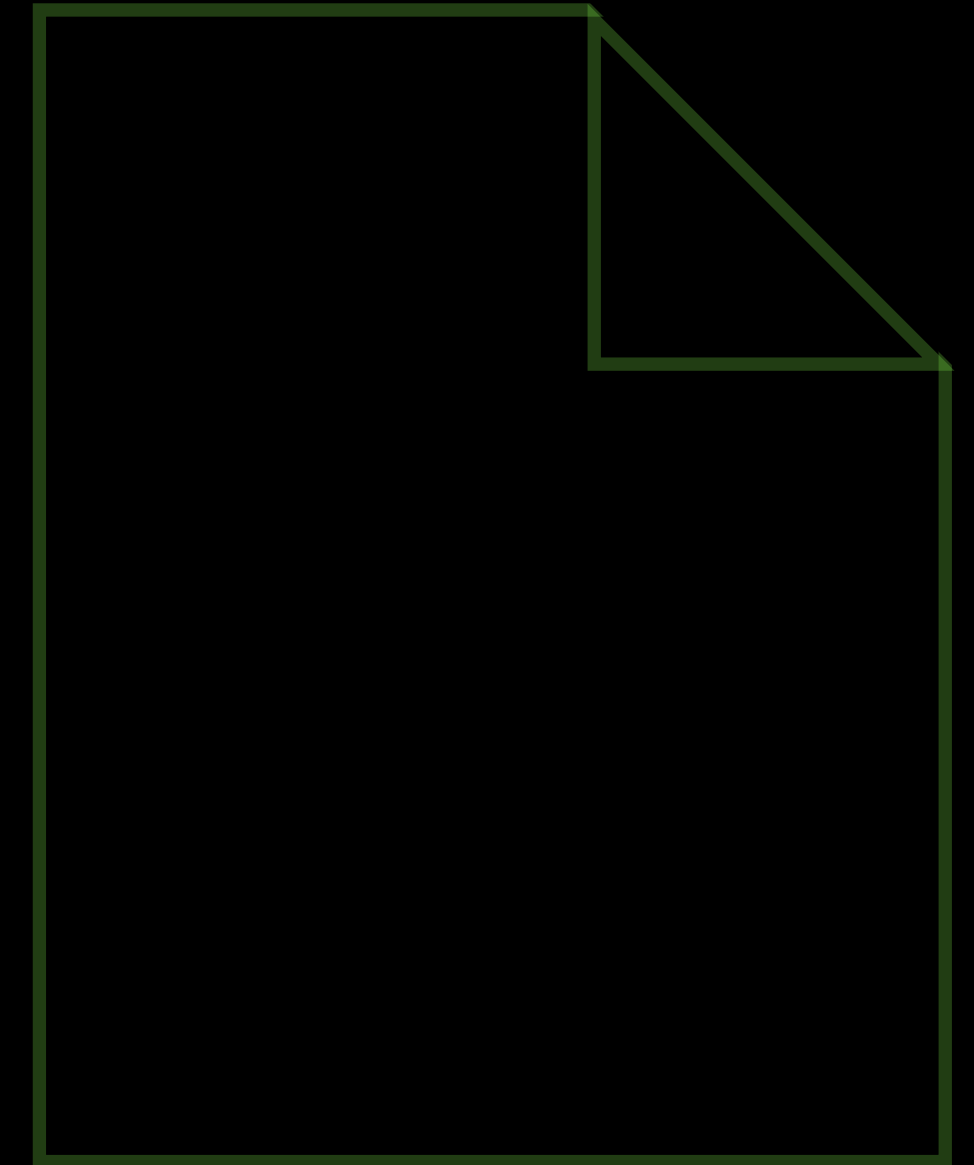
**index.html**



**about.html**



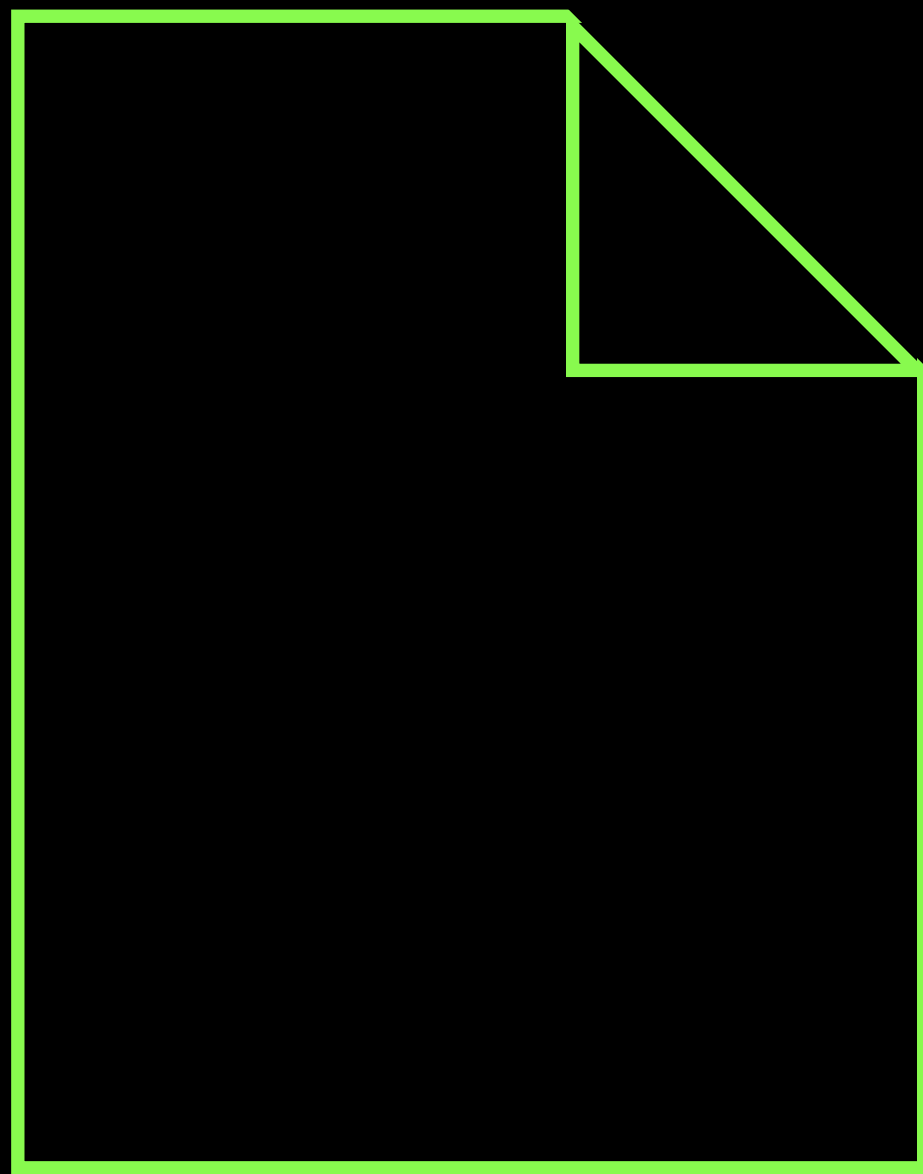
**contact.html**



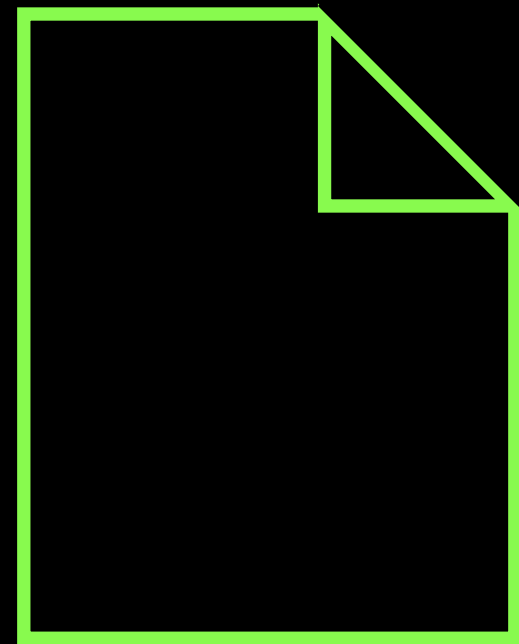
**style.css**

Every website needs an html file named `index.html`—this is the first page looked for by a computer.

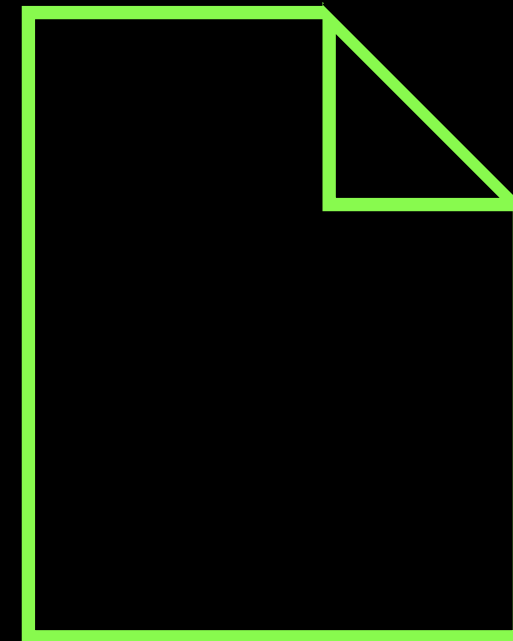
# A NOTE ON FILE NOMENCLATURE



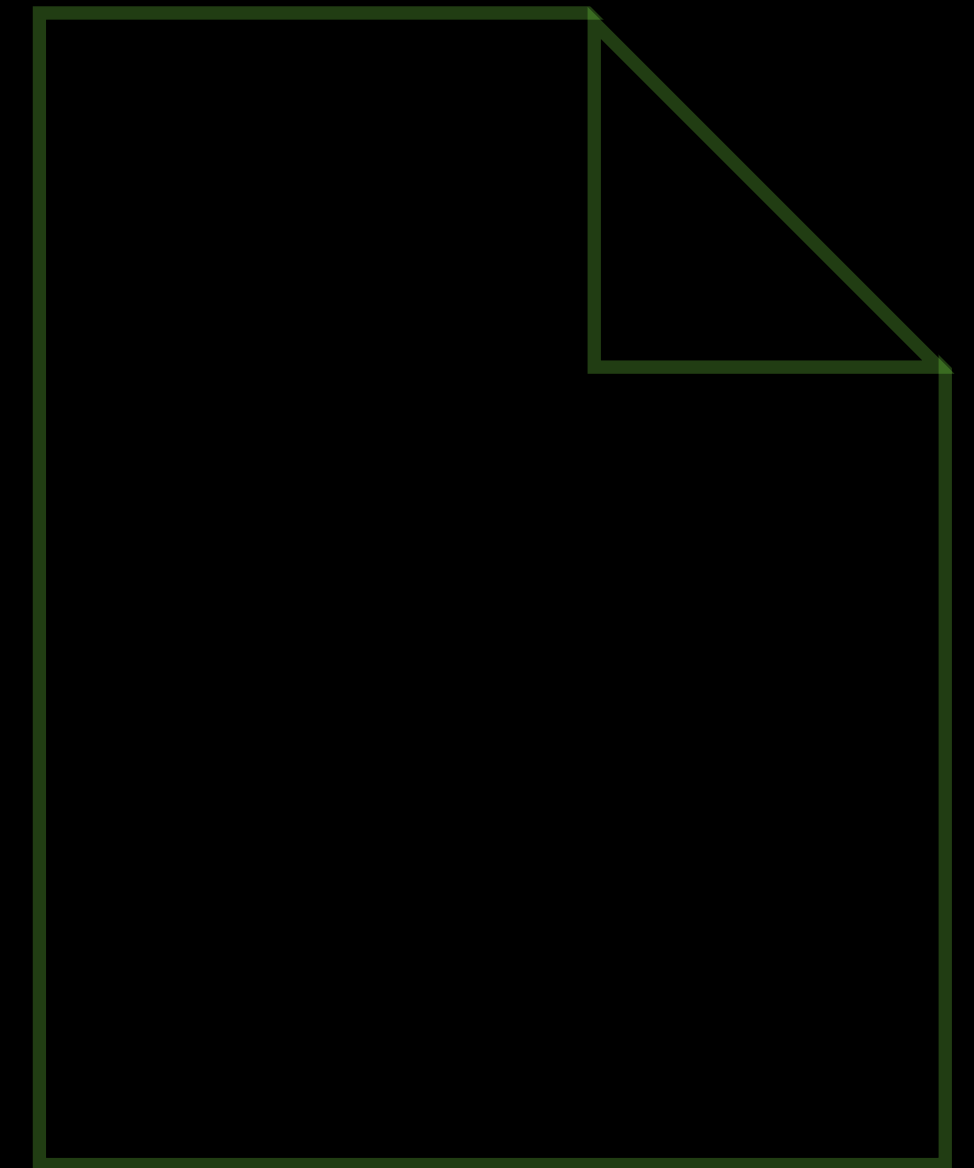
`index.html`



`about.html`



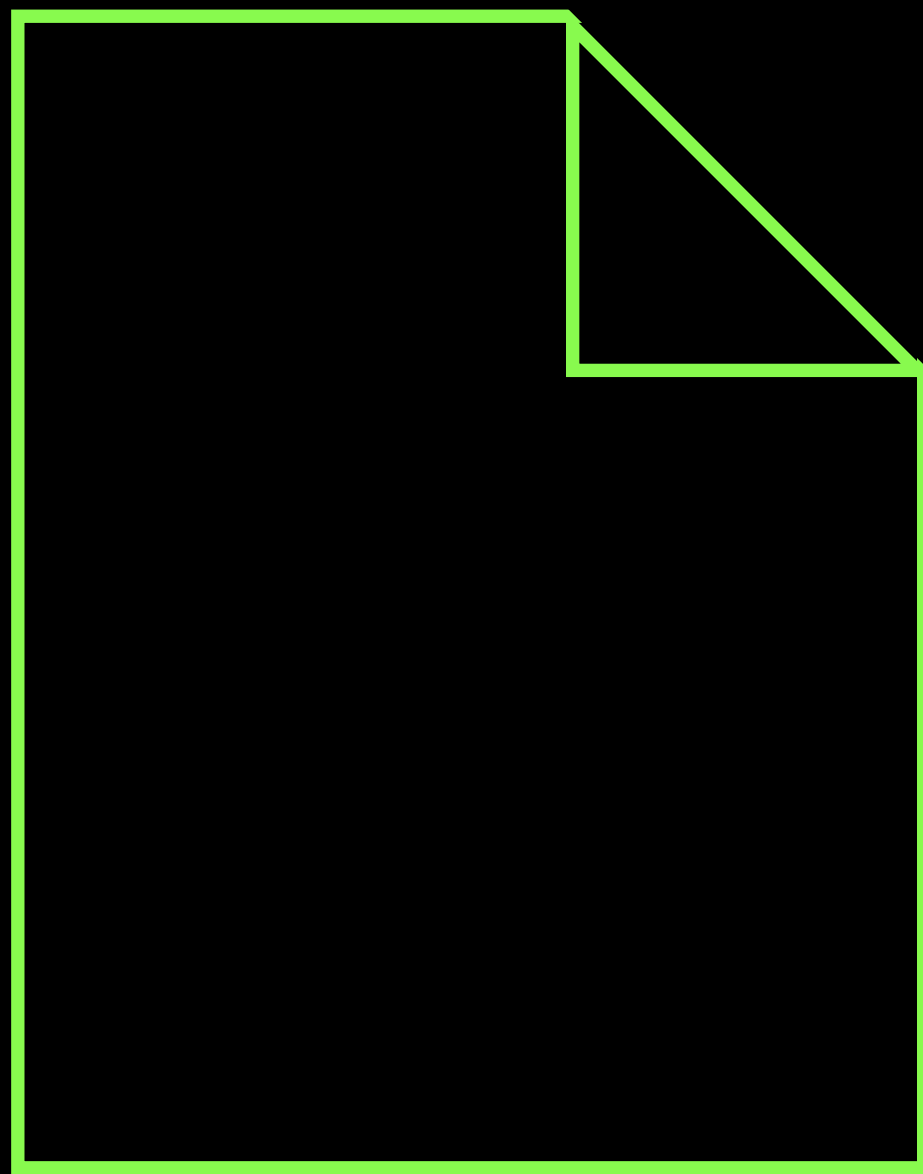
`contact.html`



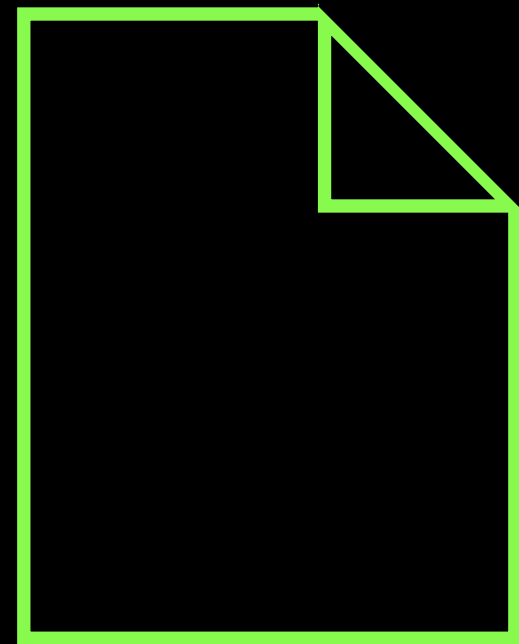
`style.css`

However!!! Not every html file needs to be, or **should**, be named index!  
Descriptive names for pages in a larger site are good.

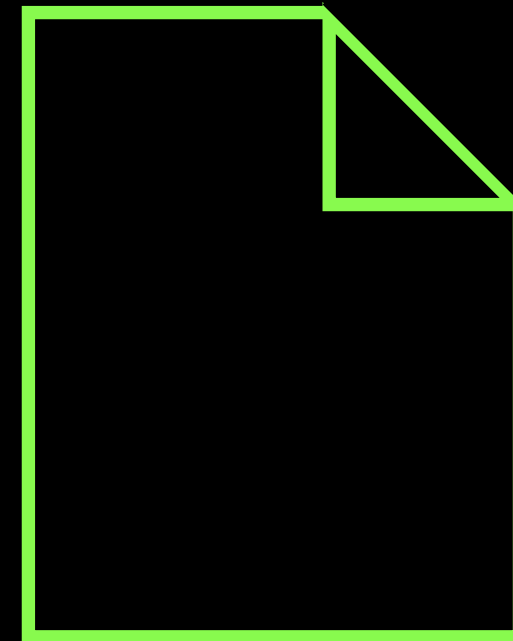
# A NOTE ON FILE NOMENCLATURE



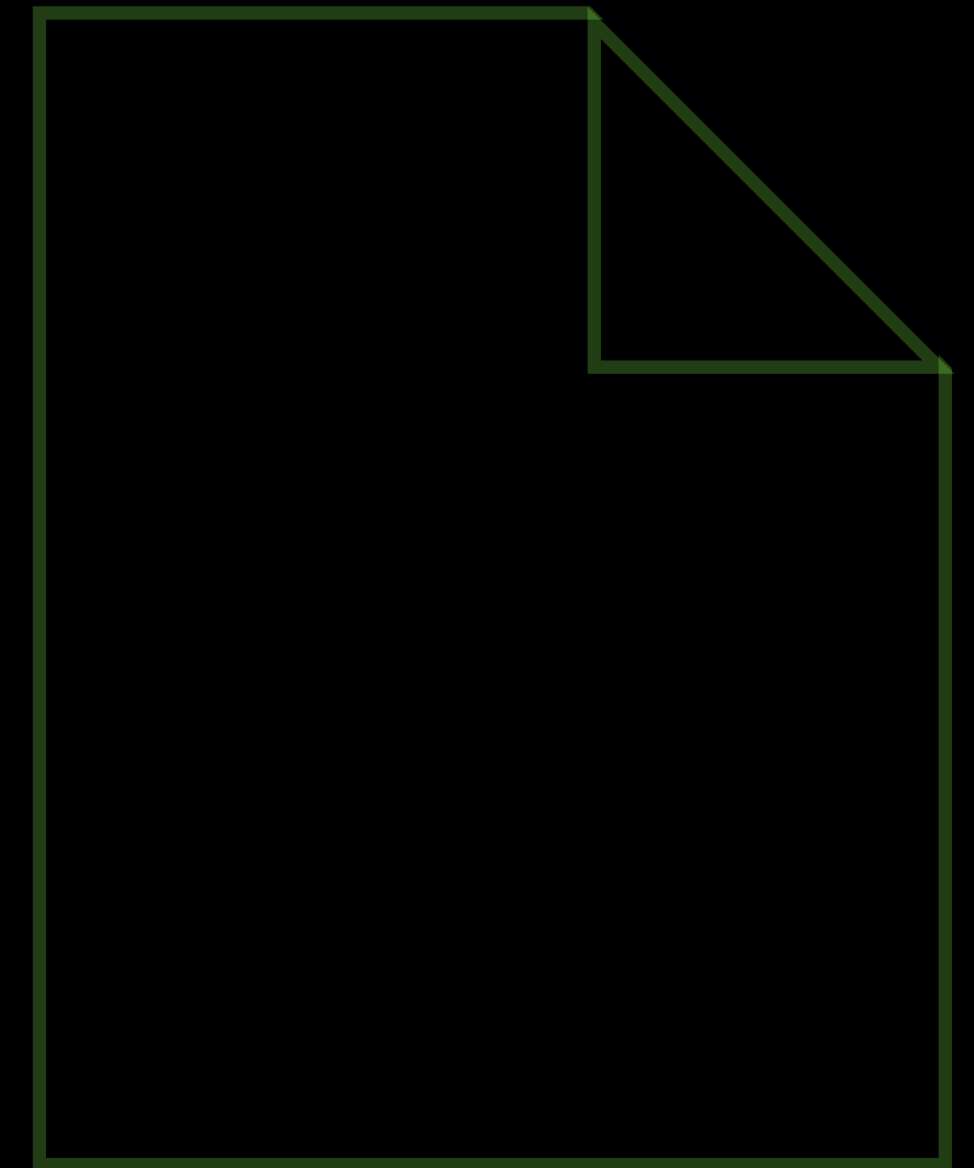
**index.html**



**about.html**



**contact.html**

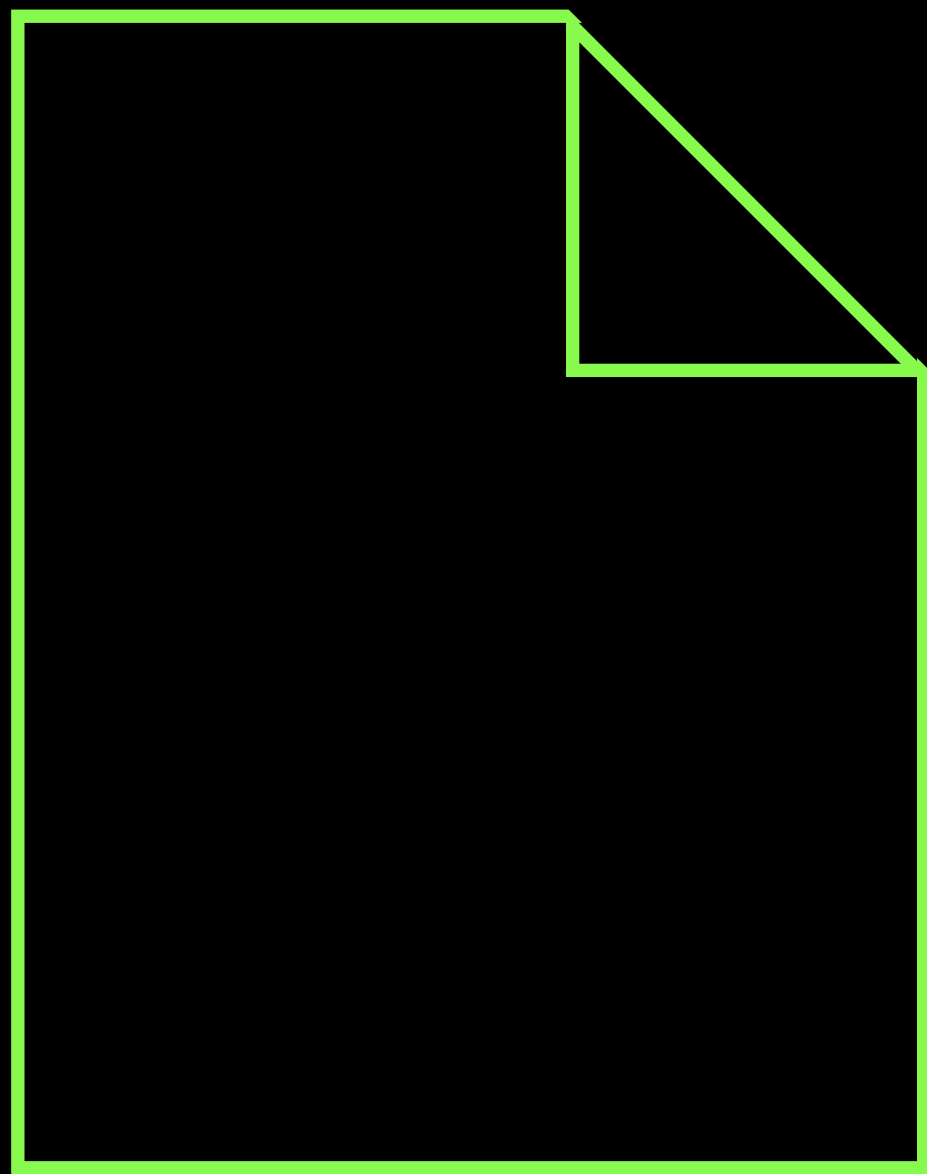


**style.css**

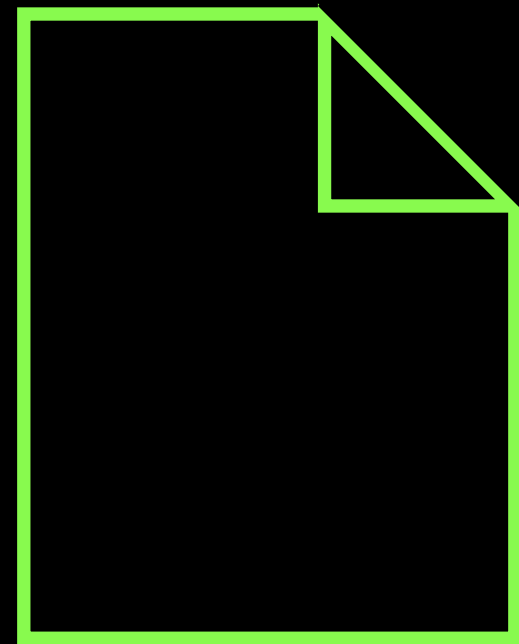
For this class—exercises etc—name the files whatever will be helpful for you. e.g. hierarchypractice.html



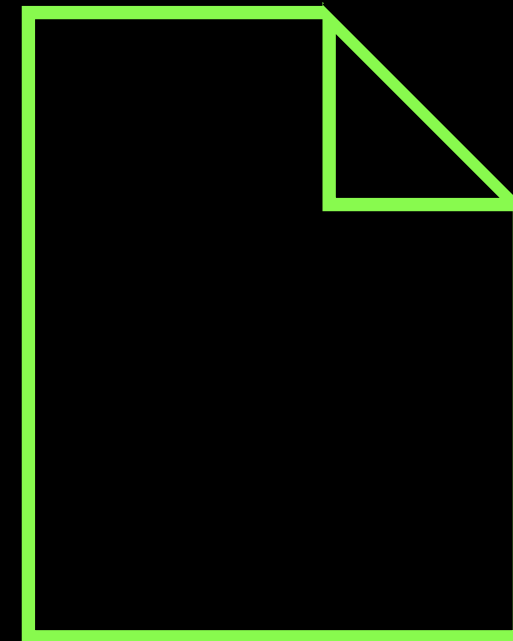
# A NOTE ON FILE NOMENCLATURE



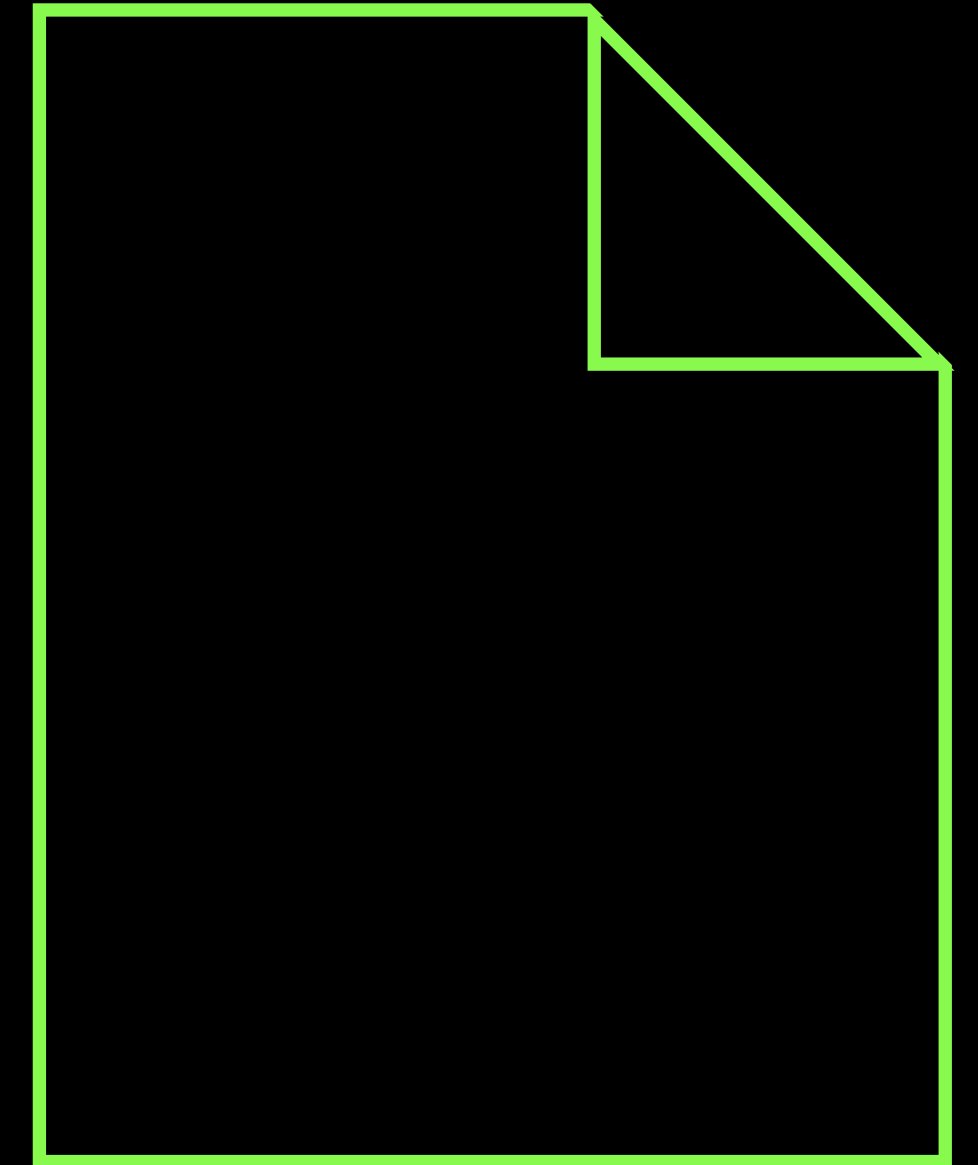
`index.html`



`about.html`



`contact.html`



`style.css`

Today we're talking about css files—these can also be named **anything**, as long as they're linked properly. I will generally use `style.css` for ease.

# HTML Questions?

CSS was first proposed by **Håkon Wium Lie** in 1994 as a way to exert more control over the presentation of web pages.

# CSS

It's had three major revisions,  
Although CSS 3 has been continuously  
updated and adapted since 1999.

# CSS

CSS can live in three places:

**1** Inline in elements

**2** In `<style>` elements in HTML documents

**3** In separate, linked .css files via `<link>`

# 1. INLINE

The original way to apply CSS,  
inline code applies styles directly  
to elements as **attributes**.

# 1. INLINE

Attribute



`<p style="color: red;">`  
`This text will be red</p>`

# 1. **INLINE**

But! This method requires you to independently style every element—it's redundant and repetitive code (not to mention hard to update).



## 2. `<style>`

`<style>` wraps css, which will apply to the whole document.  
It lives in the `<head>` of the html

## 2. `<style>`

`<style>` wraps css, which will apply to the whole document.  
It lives in the `<head>` of the html

## 2. <style>

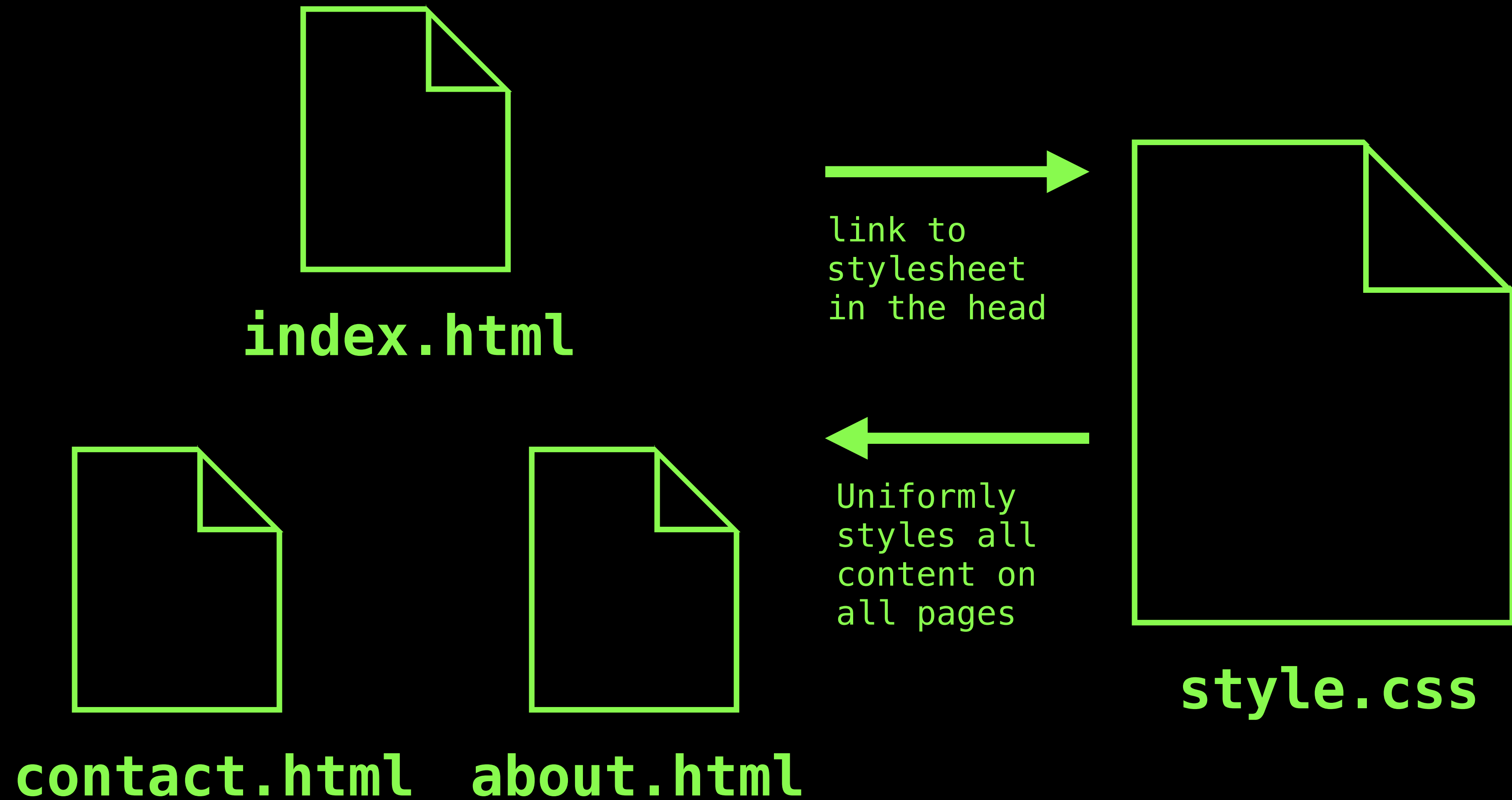
```
<!DOCTYPE html>
<html>
  <head>
    <title>Page title</title>
    <style>
      p {
        color: red;
        font-family: sans-serif;
      }
    </style>
  </head>
  <body>
    <p>This is a paragraph.</p>
  </body>
</html>
```

This is a paragraph.

### 3. `<link>`

But, best practice is to `<link>` a separate css document and silo all of your style code there. This allows you to apply css across an entire website—not just page by page.

# 3. <link>



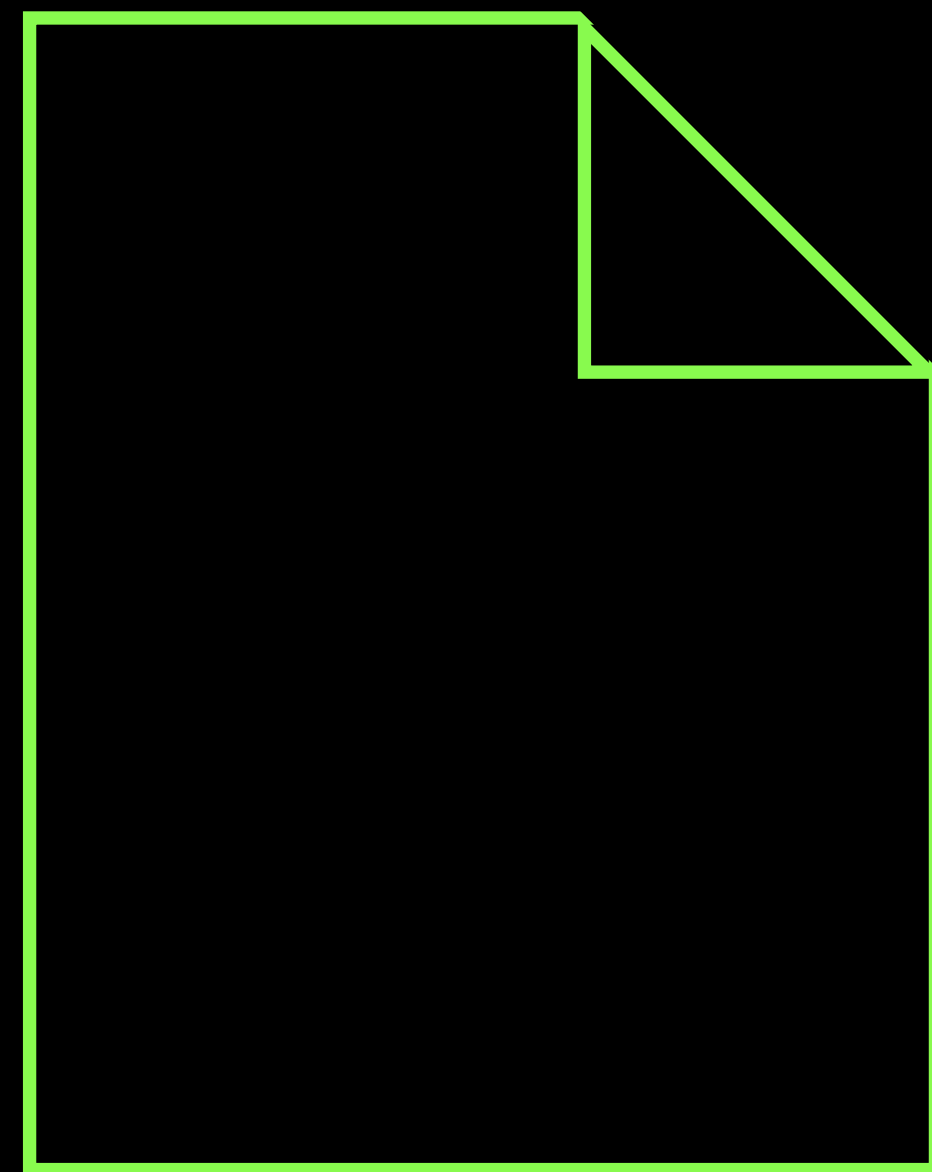
### 3. <link>

```
<head>
```

```
  <link rel="stylesheet"  
    href="style.css">
```

```
</head>
```

# 3. `<link>`

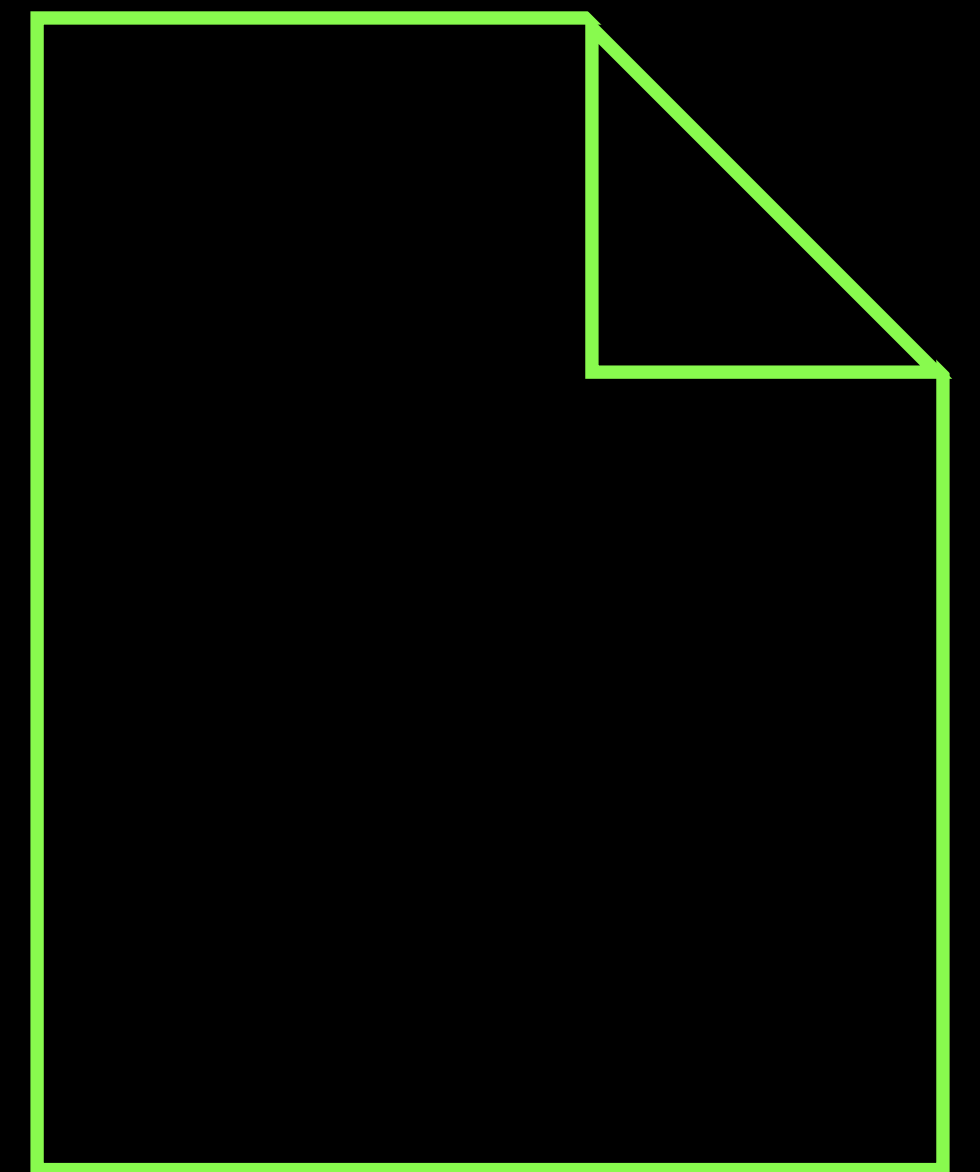


**index.html**

Element (general or specific) exists as an entity



Tells the computer how that element should look & act



**style.css**

# CSS RULES

The **syntax** of CSS is entirely separate from that of HTML.



# CSS RULES



# CSS RULES

```
p {  
  color: red;  
}
```

# CSS RULES

Like HTML, line breaks & capitalization\* are not important in CSS—but indents help you organize your code and make it more legible to others.

\*EXCEPT when identifying classes and IDs

# SELECTORS

CSS targets specific HTML elements in order to apply styles, via **selectors**.  
There are three primary types:  
elements, classes, and IDs.

# SELECTORS: ELEMENTS

Remember, elements are the units of html—e.g. `<h1>`. Use this type of CSS selector to change **all elements** of the same type.

# SELECTORS: ELEMENTS

```
<!DOCTYPE html>
<html>
  <head>
    <title>Element</title>
    <link href="style.css" rel="stylesheet">
  </head>
  <body>
    <h1>This heading will be gray</h1>
    <p>This paragraph will be red.</p>
    <blockquote>This blockquote will be
blue.</blockquote>
    <p>This paragraph will also be red.</p>
    <footer>
      <blockquote>This blockquote, also blue,
nested in a gray footer.</blockquote>
      <a href="#">A gray link, from the
wildcard</a>
    </footer>
  </body>
</html>
```

```
/* This is a "wildcard,"
targeting any element. */
* { color: gray; }

p { color: red; }

blockquote { color: blue; }

footer { background-color:
lightgray; }
```

# SELECTORS: ELEMENTS

This heading will be grey

This paragraph will be red

This blockquote will be blue

This paragraph will also be red

This blockquote, also blue, nested in a grey footer.

A stray link from the wildcard

```
/* This is a "wildcard,"  
targeting any element. */
```

```
* { color: gray; }
```

```
p { color: red; }
```

```
blockquote { color: blue; }
```

```
footer { background-color:  
lightgray; }
```

# SELECTORS: CLASSES

If you want to style only **specific instances** of an element, use **classes** to select them.



# SELECTORS: CLASSES

A class is added as an **attribute** in your .html document, which can then be selected in the .css document

# SELECTORS: CLASSES

```
<p class=  
"example">
```

```
p.example{ }
```

# SELECTORS: CLASSES

```
<!DOCTYPE html>
<html>
  <head>
    <title>Class</title> <link
    href="style.css" rel="stylesheet">
  </head>
  <body>
    <h1 class="highlight">This heading
    will be blue</h1>
    <p>This paragraph will remain black,
    since it has no class on it.</p>
    <p>So will this one.</p>
    <p class="faded">This paragraph will
    be faded.</p>
    <p class="highlight faded">This
    paragraph will be blue and faded, with
    both classes.</p>
  </body>
</html>
```

```
.highlight { color: blue; }
```

```
.faded { opacity: 33%; }
```

# SELECTORS: Classes

## **This heading will be blue**

This paragraph will remain black, since it has no class on it.

So will this one.

This paragraph will be faded

This paragraph will be blue and faded, with both classes.

```
.highlight { color: blue; }
```

```
.faded { opacity: 33%; }
```

# SELECTORS: CLASSES

Elements can have multiple classes,  
and classes can be used as many  
times as you need.

# SELECTORS: CLASSES

Class names can be anything,  
but try and be descriptive—as always,  
it's all to help you and others  
decipher your code for access and  
debugging.

# SELECTORS: IDs

You can also select using an `id`—  
an attribute which can only  
be used once in an HTML document.

# SELECTORS: IDs

Use this to apply css to a specific element—e.g. your nav or a heading



# SELECTORS: IDs

```
<!DOCTYPE html>
<html>
  <head>
    <title>ID</title>
    <link href="style.css" rel="stylesheet">
  </head>
  <body>
    <h1 id="title">This heading will be
    red</h1>
    <p>This paragraph will be black, with no
    ID.</p>
    <p id="introduction">This paragraph will
    be blue.</p>
  </body>
</html>
```

```
#title { color: red; }
```

```
#introduction { color: blue; }
```

# SELECTORS: IDs

**This heading will be red**

This paragraph will be black, with no ID.

This paragraph will be blue.

```
#title { color: red; }
```

```
#introduction { color: blue; }
```

# SELECTORS: IDs

These id tags can also be used as link destinations, to jump down to a specific part of a page.

```
<a href="#section1">
```

# SELECTORS: SELECTOR LISTS

You can apply css to several elements, classes, or ids at a time using selector lists.

# SELECTORS: SELECTOR LISTS

```
<!DOCTYPE html>
<html>
  <head>
    <title>Combinations and groups</title>
    <link href="style.css" rel="stylesheet">
  </head>
  <body>
    <h2 id="first-heading">This heading will
    be black</h2>
    <p class="warning">This paragraph with be
    red.</p>
    <blockquote class="warning">This
    blockquote will black, despite having the
    same class.</blockquote>
    <h2 id="second-heading">This heading will
    be blue</h2>
    <footer>
      <p>Just another paragraph, nested in a
      footer.</p>
    </footer>
  </body>
</html>
```

```
/* Paragraphs with the class
`warning`. */
p.warning { color: red; }

/* The `h2` isn't really necessary
here... */
/* Since the `id` is already
unique! */
h2#second-heading { color: blue; }

/* This applies to both selectors!
D.R.Y. */
blockquote,
footer { background-color:
lightgray; }
```

# SELECTORS: SELECTOR LISTS

**This heading will be black**

**This paragraph will be red**

This blockquote will be black, despite having the same class.

**This heading will be blue**

Just another paragraph, nested in a footer

```
/* Paragraphs with the class  
`warning`. */
```

```
p.warning { color: red; }
```

```
/* The `h2` isn't really necessary  
here... */
```

```
/* Since the `id` is already  
unique! */
```

```
h2#second-heading { color: blue; }
```

```
/* This applies to both selectors!  
D.R.Y. */
```

```
blockquote,  
footer { background-color:  
lightgray; }
```

# SELECTORS: ATTRIBUTES

You can select by attribute, as well.  
This is similar to selecting classes.  
It can be useful for differentiating  
between internal and external links  
(for instance).

# SELECTORS: ATTRIBUTES

```
<!DOCTYPE html>
<html>
  <head>
    <title>Attributes</title> <link
      href="style.css" rel="stylesheet">
  </head>
  <body>
    <p> You might use these to differentiate
    <a href="/">an internal link</a> from an
    <a href="https://newschool.edu">an
    external one</a>, or ones for <a href="/
    assets/PUCD_2035.pdf">PDF files</a>. </p>
    <details>
      <summary>Or an open/close state for
      details/summary</summary>
      <p>This won't be visible to start, but
      should be teal when you can see it.</p>
    </details>
  </body>
</html>
```

```
/* Links that start with `https://`
` are external. */
```

```
a[href^='https://'] { color: teal;
}
```

```
/* Links with `.pdf` anywhere in
them. */
```

```
a[href*='.pdf'] { color: green; }
```

```
/* The `open` attribute is added
when you toggle. */
```

```
details[open] { color: gray; }
```



# SELECTORS: ATTRIBUTES

You might use these to differentiate an [internal link](#) from an [external one](#), or ones for [PDF files](#).

Or an open/close state for details/summary

```
/* Links that start with `https://`  
` are external. */
```

```
a[href^='https://'] { color: teal;  
}
```

```
/* Links with `.pdf` anywhere in  
them. */
```

```
a[href*='.pdf'] { color: green; }
```

```
/* The `open` attribute is added  
when you toggle. */
```

```
details[open] { color: gray; }
```

# SELECTORS: PSEUDOCLASSES

Some HTML elements have, through interaction or default, states that can be used as CSS selectors. e.g. open for a detail

# SELECTORS: PSEUDOCLASSES

Some HTML elements have, through interaction or default, states that can be used as CSS selectors. e.g. open for a detail

# SELECTORS: PSEUDOCASSES

```
<!DOCTYPE html>
<html>
  <head>
    <title>Pseudoclasses</title>
    <link href="style.css" rel="stylesheet">
  </head>
  <body>
    <p>This is a paragraph with <a
href="#">a link</a> in it.</p>
    <p>And another one with <a href="/topic/
css">another link</a>.</p>
  </body>
</html>
```

```
/* A link that hasn't been visited.
*/
a:link { color: red; }

/* After you've visited the link. */
a:visited { color: purple; }

/* Change the color when the mouse
is over it. */
a:hover { color: fuchsia; }

/* And when the mouse is clicked. */
a:active { color: maroon; }
```

# SELECTORS: PSEUDOCASSES

This is a paragraph with a link in it.

And another one with another link.

```
/* A link that hasn't been visited.
*/
```

```
a:link { color: red; }
```

```
/* After you've visited the link. */
```

```
a:visited { color: purple; }
```

```
/* Change the color when the mouse
is over it. */
```

```
a:hover { color: fuchsia; }
```

```
/* And when the mouse is clicked. */
```

```
a:active { color: maroon; }
```

# SELECTORS: PSEUDOCLASSES

```
<!DOCTYPE html>
<html>
  <head>
    <title>Pseudoclasses</title>
    <link href="style.css" rel="stylesheet">
  </head>
  <body>
    <div>
      <p>This should be blue.</p>
      <p>Standard black, here.</p>
      <p>Also black.</p>
      <p>This one is purple!</p>
      <p>Back to black.</p>
      <p>This will be fuchsia.</p>
    </div>
    <div>
      <p>This should be red.</p>
    </div>
  </body>
</html>
```

```
/* The paragraph is the first
descendent of its parent. */
p:first-child { color: blue; }

/* Paragraphs that are the 4th
children. */
p:nth-child(4) { color: purple; }

/* Last one. */
p:last-child { color: fuchsia; }

/* Only one. */
p:only-child { color: red; }

/* You can invert these, too. */
div:not(:first-child) { background-
color: lightgray; }
```

# SELECTORS: PSEUDOCLASSES

This should be blue.

Standard black, here.

Also black.

This one is purple!

Back to black.

This will be fuchsia.

This should be red.

```
/* The paragraph is the first  
descendent of its parent. */  
p:first-child { color: blue; }
```

```
/* Paragraphs that are the 4th  
children. */  
p:nth-child(4) { color: purple; }
```

```
/* Last one. */  
p:last-child { color: fuchsia; }
```

```
/* Only one. */  
p:only-child { color: red; }
```

```
/* You can invert these, too. */  
div:not(:first-child) { background-  
color: lightgray; }
```



# SELECTORS: PSEUDOELEMENTS

Alternately, some selectors allow you to style a specific *part* of an element; often this looks like  
:before or :after



# SELECTORS: PSEUDOELEMENTS

```
<!DOCTYPE html>
<html>
  <head>
    <title>Pseudoelements</title>
    <link href="style.css" rel="stylesheet">
  </head>
  <body>
    <h1>Pseudo-elements are clever</h1>
    <p>The first letter of this paragraph
    will be red, and the first line maroon,
    no matter where it wraps or how long it
    ends up being.</p>
    <blockquote> You can use these to <a
    href="https://newschool.edu">embellish
    links</a> </blockquote>
  </body>
</html>
```

```
/* These are inserted before and after
the heading. */
h1:before,
h1:after {
  color: red;
  content: ' ... ';
}

p:first-letter {
  color: red;
  font-size: larger;
  font-weight: bold;
}

p:first-line { color: maroon; }

/* After external links. */
a[href^='https://']:after { content: '
↗'; }
```

# SELECTORS: PSEUDOELEMENTS

## ●●● Pseudoelements are clever ●●●

**T**he first letter of this paragraph will be red, and the first line maroon, no matter where it wraps or how long it ends up being.

You can use these to [embellish links](#) ↗

```
/* These are inserted before and after
the heading. */
h1:before,
h1:after {
  color: red;
  content: ' ... ';
}

p:first-letter {
  color: red;
  font-size: larger;
  font-weight: bold;
}

p:first-line { color: maroon; }

/* After external links. */
a[href^='https://']:after { content: '
↗'; }
```

# SELECTORS: COMBINATORS

Lastly, you can select based on relationships—a sibling or parent. Combinators combine other selectors to accomplish this.

# SELECTORS: COMBINATORS

```
<!DOCTYPE html>
<html>
  <head>
    <title>Combinators</title>
    <link href="style.css" rel="stylesheet">
  </head>
  <body>
    <header>
      <h1>Combinators are handy</h1>
      <p>This paragraph should be <em>blue</em>,
      since it is inside the header.</p>
    </header>
    <p>This paragraph should be <em>gray</em>, as an
    immediate child of <strong>body</strong>.</p>
    <p>This one should be <em>teal</em> though, as it
    follows another paragraph.</p>
    <p>Same for this one.</p>
    <h2>This heading doesn't have a style itself</h2>
    <p>But will make this following paragraph
    <em>green</em>.</p> <p>And this one too, still
    after the heading.</p>
  </body>
</html>
```

```
/* The "descendant" combinator. */
/* Only paragraphs inside the header. */
header p { color: blue; }
```

```
/* The "child" combinator. */
/* Only immediate children paragraphs of
`body`. */
body > p { color: gray; }
```

```
/* "Adjacent sibling" combinator. */
/* Only immediately preceded by another
paragraph. */
p + p { color: teal; }
```

```
/* "General sibling" combinator. */
/* Only paragraphs following `h2`. */
h2 ~ p { color: green; }
```

# SELECTORS: COMBINATORS

## Combinators are handy

This paragraph should be blue, since it is inside the header.

This paragraph should be gray, as an immediate child of body.

This one should be teal though, as it follows another paragraph.

Same for this one.

## This heading doesn't have a style itself

But will make this following paragraph green.

And this one too, still after the heading.

```
/* The “descendant” combinator. */  
/* Only paragraphs inside the header. */  
header p { color: blue; }
```

```
/* The “child” combinator. */  
/* Only immediate children paragraphs of  
`body`. */  
body > p { color: gray; }
```

```
/* “Adjacent sibling” combinator. */  
/* Only immediately preceded by another  
paragraph. */  
p + p { color: teal; }
```

```
/* “General sibling” combinator. */  
/* Only paragraphs following `h2`. */  
h2 ~ p { color: green; }
```

# SELECTORS: COMBINATORS

combinators can only “see” elements before and above themselves—meaning their (older) *siblings* or their *parents*.

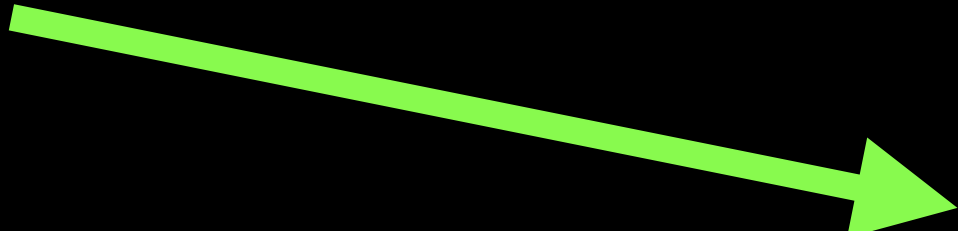
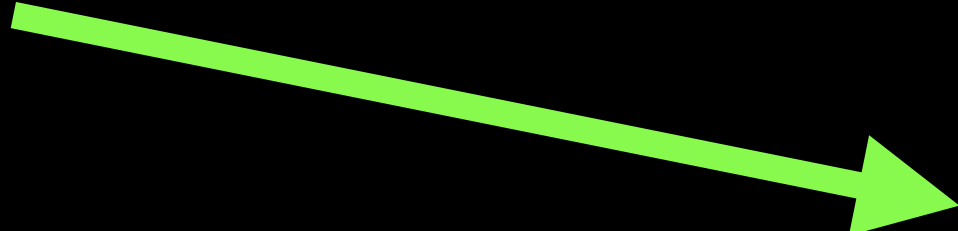


# SELECTORS: COMBINATORS

You can use `:has` to apply a style to a parent based on its children.

```
div:has(p) { background-color: red; }
```

# SPECIFICITY

We've discussed selectors  
in order of specificity, where  
elements are broader than  
 classes are broader than  
 ids



# SPECIFICITY

More specific selectors  
override less specific selectors.

If I had a paragraph `<p class="blue">`  
with the css `p{red}` and  
`p.water{blue}` ,  
the paragraph would be blue.

# SELECTORS: SPECIFICITY

```
<!DOCTYPE html>
<html>
  <head>
    <title>Specificity</title>
    <link href="style.css" rel="stylesheet">
  </head>
  <body>
    <p>This paragraph will be red, from the
    <em>element</em>.</p>
    <p class="maroon">This paragraph will be maroon,
    from the <em>class</em>.</p>
    <p>This paragraph will be red again, no
    <em>class</em>.</p>
    <h2 class="maroon" id="intro">This heading will
    be blue, from the <em>ID</em>, even though it has
    the <em>class</em> too</h2>
    <p id="warning">This paragraph will be teal, from
    the <em>ID</em>, beating the <em>element</em>.</
    p>
    <p class="maroon" style="color: gray;">This
    paragraph will be gray, from an <em>inline
    style</em>, which beats everything.</p>
  </body>
</html>
```

```
p { color: red; }

/* Will "win" over an element selector.
*/
.maroon { color: maroon; }

/* But IDs will supersede classes. */
#intro { color: blue; }

#warning { color: teal; }
```

# SELECTORS: SPECIFICITY

This paragraph will be red, from the element.

This paragraph will be maroon, from the class.

This paragraph will be red again, no class.

This heading will be blue, from the ID, even though it has the class too

This paragraph will be teal, from the ID, beating the element.

This paragraph will be gray, from an inline style, which beats everything.

```
p { color: red; }
```

```
/* Will “win” over an element selector. */
```

```
.maroon { color: maroon; }
```

```
/* But IDs will supersede classes. */  
#intro { color: blue; }
```

```
#warning { color: teal; }
```

# CASCADE

The first C in CSS is **cascade**, and in this context that just means when there are two competing properties, the one that is **physically lowest** in the .css document or `<style>` wins.

# CASCADE

```
<!DOCTYPE html>
<html>
  <head>
    <title>Cascade</title>
    <link href="style.css" rel="stylesheet">
  </head>
  <body>
    <p class="note">This paragraph will be
    gray, with just the "note" class.</p>
    <p class="warning note">This paragraph
    will be red though, with both classes.</
    p>
    <p class="note">This paragraph will also
    be gray, with just "note" again.</p>
  </body>
</html>
```

```
.note { color: gray; }

/* Because this is lower, it will
"win" the tie. */
.warning { color: red; }
```

# CASCADE

This paragraph will be gray, with just the “note” class.

**This paragraph will be red though, with both classes.**

This paragraph will also be gray, with just “note” again.

```
.note { color: gray; }
```

```
/* Because this is lower, it will  
“win” the tie. */
```

```
.warning { color: red; }
```

# INHERITANCE

Some CSS properties applied to parents will also apply to their children:  
for instance, type styles.

# INHERITANCE

```
<!DOCTYPE html>
<html>
  <head>
    <title>Inheritance</title>
    <link href="style.css" rel="stylesheet">
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph.</p>
    <p>This is another paragraph.</p>
    <p>This is <span>third</span>
    paragraph.</p>
  </body>
</html>
```

```
body {
  color: gray;
  font-family: sans-serif;
}

span {
  color: blue;
  font-style: italic;
  font-weight: bold;
}
```



# INHERITANCE

This is a heading

This is a paragraph.

This is another paragraph.

This is *third* paragraph.

```
body {  
    color: gray;  
    font-family: sans-serif;  
}  
  
span {  
    color: blue;  
    font-style: italic;  
    font-weight: bold;  
}
```

# SOME PROPERTIES: COLOR

A little pivot: We've talked a lot about *applying* css properties, but now... let's discuss the properties themselves. First up, color.

# COLOR

```
<!DOCTYPE html>
<html>
  <head>
    <title>Color</title>
    <link href="style.css" rel="stylesheet">
  </head>
  <body>
    <ol>
      <li>This paragraph will be <em>dodgerblue</em>.</li>
      <li>Which can also be written as <em>#1e90ff</em>.</li>
      <li>Or as <em>rgb(30, 144, 255)</em>.</li>
      <li>And as <em>hsl(210, 100%, 56%)</em>.</li>
      <li>This has an <em>alpha</em> (transparency) value in the color.</li>
      <li>Same for this one.</li>
      <li>This uses a separate <em>opacity</em> property.</li>
      <li>All of these work for backgrounds, too!</li>
    </ol>
    <style>li:not(:first-child) {
      margin-top: 0.666em }</style>
  </body>
</html>
```

```
/* There are a bunch of named colors. */
li:nth-child(1) { color: dodgerblue; }

/* Or you can specify a hex value. Brands love these.
*/
li:nth-child(2) { color: #1e90ff; }

/* Or RGB for Red, Green, Blue, if you think like a
screen. */
li:nth-child(3) { color: rgb(30, 144, 255); }

/* Or Hue, Saturation, Lightness, which is more human.
*/
li:nth-child(4) { color: hsl(210, 100%, 56%); }

/* RGB and HSL have a fourth value, for Alpha
(transparency). */
li:nth-child(5) { color: rgba(30, 144, 255, 50%); }
li:nth-child(6) { color: hsla(210, 100%, 56%, 0.5); }

/* Alpha as a percentage, 0–100%, or a decimal, 0–1. */
li:nth-child(7) { color: dodgerblue; opacity: 0.5;
/* This will affect everything, not just text! */ }

li:nth-child(8) { color: white; background-color:
dodgerblue; /* Same for backgrounds! */ }
```

# COLOR

1. This paragraph will be dodgerblue.
2. Which can also be written as #1e90ff.
3. Or as rgb(30, 144, 255).
4. And as hsl(210, 100%, 56%).
5. This has an alpha (transparency) value in the color.
6. Same for this one.
7. This uses a separate opacity property.

All of these work for backgrounds, too!

```
/* There are a bunch of named colors. */
li:nth-child(1) { color: dodgerblue; }

/* Or you can specify a hex value. Brands love these.
*/
li:nth-child(2) { color: #1e90ff; }

/* Or RGB for Red, Green, Blue, if you think like a
screen. */
li:nth-child(3) { color: rgb(30, 144, 255); }






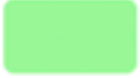
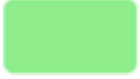

/* Or Hue, Saturation, Lightness, which is more human.
*/
li:nth-child(4) { color: hsl(210, 100%, 56%); }

/* RGB and HSL have a fourth value, for Alpha
(transparency). */
li:nth-child(5) { color: rgba(30, 144, 255, 50%); }
li:nth-child(6) { color: hsla(210, 100%, 56%, 0.5); }

/* Alpha as a percentage, 0–100%, or a decimal, 0–1. */
li:nth-child(7) { color: dodgerblue; opacity: 0.5; }
/* This will affect everything, not just text! */ }

li:nth-child(8) { color: white; background-color:
dodgerblue; /* Same for backgrounds! */ }
```

# SOME PROPERTIES: COLOR

	GreenYellow	#ADFF2F	rgb(173, 255, 47)
	Chartreuse	#7FFF00	rgb(127, 255, 0)
	LawnGreen	#7CFC00	rgb(124, 252, 0)
	Lime	#00FF00	rgb(0, 255, 0)
	LimeGreen	#32CD32	rgb(50, 205, 50)
	PaleGreen	#98FB98	rgb(152, 251, 152)
	LightGreen	#90EE90	rgb(144, 238, 144)
	MediumSpringGreen	#00FA9A	rgb(0, 250, 154)

There are 147 named colors, which you can find online, or you can use hex/rgb/hsla values to display any color in the gamut.

# SOME PROPERTIES: COLOR

Any color value can also be plugged into background-color, which will flood the entirety of the element.

# SOME PROPERTIES: TYPE

One of the most important sets of  
css properties (for us) is about  
customizing your typography.  
The web is all about text, after all.



# SOME PROPERTIES: TYPE

One of the most important sets of css properties (for us) is about customizing your typography.  
The web is all about text, after all.



# FONTS

```
<!DOCTYPE html>
<html>
  <head>
    <title>Color</title>
    <link href="style.css" rel="stylesheet">
  </head>
  <body>
    <ol>
      <li>Default serif</li>
      <li>Default sans-serif</li>
      <li>Default monospace</li>
      <li>Default cursive, you don't see this much</li>
      <li>Default fantasy, maybe don't do this</li>
      <li>Futura, on a Mac; Verdana, on a PC</li>
      <li>Some condensed Roboto, imported from Google Fonts</li>
      <li>This will be in Playfair, using @font-face</li>
    </ol>
    <style>li:not(:first-child) { font-size: 18px;
margin-top: 0.666em }</style>margin-top:
0.666em }</style>
  </body>
</html>
```

```
/* Importing from Google Fonts. These have to be at the top!
*/
@import url('https://fonts.googleapis.com/css2?
family=Roboto+Condensed:wght@300&display=swap');

/* These will depend on your user's platform and browser. */
li:nth-child(1) { font-family: serif; }
li:nth-child(2) { font-family: sans-serif; }
li:nth-child(3) { font-family: monospace; }
li:nth-child(4) { font-family: cursive; }
li:nth-child(5) { font-family: fantasy; }

/* You can also choose a specific typeface. */
li:nth-child(6) { font-family: 'Futura', 'Verdana', sans-
serif; }

/* But the user has to have them installed on their device. */
/* Otherwise it will "fall back" to the next/right value. */
/* The `@import` above lets you use this on any computer. */
li:nth-child(7) { font-family: 'Roboto Condensed', sans-serif;
}

/* You can also host fonts along with your site files. */
@font-face { font-family: 'Playfair'; src: url('playfair--
regular.woff2') format('woff2'); }

/* And then reference them, as above. */
li:nth-child(8) { font-family: 'Playfair', serif; }
```

# FONTS

1. Default serif
2. Default sans-serif
3. Default monospace
4. *Default cursive, you don't see this much*
5. Default fantasy, maybe don't do this
6. **Futura, on a Mac; Verdana, on a PC**
7. Some condensed Roboto, imported from Google Fonts
8. This will be in Playfair, using @font-face

```
/* Importing from Google Fonts. These have to be at the top!
*/
@import url('https://fonts.googleapis.com/css2?
family=Roboto+Condensed:wght@300&display=swap');

/* These will depend on your user's platform and browser. */
li:nth-child(1) { font-family: serif; }
li:nth-child(2) { font-family: sans-serif; }
li:nth-child(3) { font-family: monospace; }
li:nth-child(4) { font-family: cursive; }
li:nth-child(5) { font-family: fantasy; }

/* You can also choose a specific typeface. */
li:nth-child(6) { font-family: 'Futura', 'Verdana', sans-
serif; }

/* But the user has to have them installed on their device. */
/* Otherwise it will "fall back" to the next/right value. */
/* The `@import` above lets you use this on any computer. */
li:nth-child(7) { font-family: 'Roboto Condensed', sans-serif;
}

/* You can also host fonts along with your site files. */
@font-face { font-family: 'Playfair'; src: url('playfair--
regular.woff2') format('woff2'); }

/* And then reference them, as above. */
li:nth-child(8) { font-family: 'Playfair', serif; }
```

# SOME PROPERTIES: TYPE

FONT LICENSING IS A BIG DEAL.  
In general! It is **copyright infringement**  
to just upload a typeface for use  
on your webpage! You can purchase  
specific web licenses of typefaces,  
and occasionally get student/academic  
free licenses, but for now **please** use  
google fonts or adobe fonts for web use  
unless you have an actual license!



# FONTS

```
<!DOCTYPE html>
<html>
  <head>
    <title>Color</title>
    <link href="style.css" rel="stylesheet">
  </head>
  <body>
    <p>Everything here will be in <em>Work Sans</em>,
    inherited from the <em>body</em> rule.</p>
    <ol>
      <li>This will be much larger</li>
      <li>And then much smaller</li>
      <li>This will have a much looser <em>line-height</em>
      (leading), when it wraps across lines</li>
      <li>This will be very light</li>
      <li>And then very heavy</li>
      <li>You can force italics, without an em tag</li>
      <li>Or uppercase, without editing</li>
      <li>This will have looser <em>letter-spacing</em>
      (kerning)</li>
      <li>This will be tighter</li>
      <li>You can underline some text</li>
      <li>Or cross it out</li> <li>And center it</li>
      <li>Or you could even justify it</li> <li>Or right-
      align it</li>
    </ol>
    <style>li:not(:first-child) { margin-top: 16px }</
    style>
  </body>
</html>
```

```
/* Importing a typeface with a bunch of weights. */
@import url('https://fonts.googleapis.com/css2?
family=Work+Sans:ital,wght@0,100;0,200;0,300;0,400;0,500;0,600;0,700;0,800;0,900
;1,100;1,200;1,300;1,400;1,500;1,600;1,700;1,800;1,900&display=swap');

/* Apply this family to everything. */
body { font-family: 'Work Sans', sans-serif; }

/* Size is the first thing you'll want to adjust. */
li:nth-child(1) { font-size: 28px; }
li:nth-child(2) { font-size: 10px; }
/* Browsers usually default to 16px. */

/* Then the space between your lines, relative to your font-size. */
li:nth-child(3) { line-height: 200%; }
/* This could also be written as "32px", or just "2". */

/* You can specify different weights. */
li:nth-child(4) { font-weight: 100; }
li:nth-child(5) { font-weight: 900; }
/* These go from 100–900, depending on what your family has. */
/* Default is 400, and you'll see "bold"—which maps to 700. */

li:nth-child(6) { font-style: italic; } /* Default is "normal". */

li:nth-child(7) { text-transform: uppercase; } /* YOU CAN YELL */

/* This uses a relative unit, based on the font-size. */
li:nth-child(8) { letter-spacing: 0.1em; }
li:nth-child(9) { letter-spacing: -0.1em; } /* Negative values. */
/* Keep in mind these don't apply semantic meaning... */

li:nth-child(10) { text-decoration: underline; } /* Even non-links. */
li:nth-child(11) { text-decoration: line-through; }
/* Decoration should be paired with a tag, like <em> or <del>. */

li:nth-child(12) { text-align: center; } /* Default is "left". */
li:nth-child(13) { text-align: justify; }
li:nth-child(14) { text-align: right; }
```



# FONTS

Everything here will be in *Work Sans*, inherited from the *body* rule.

## 1. This will be much larger

2. And then much smaller

3. This will have a much looser *line-height* (leading), when it wraps across lines

4. This will be very light

**5. And then very heavy**

6. *You can force italics, without an em tag*

7. OR UPPERCASE, WITHOUT EDITING

8. This will have looser *letter-spacing* (kerning)

9. This will be tighter

10. You can underline some text

11. ~~Or cross it out~~

12. And center it

13. Or you could even justify it

14. Or right-align it

```
/* Importing a typeface with a bunch of weights. */
@import url('https://fonts.googleapis.com/css2?
family=Work+Sans:ital,wght@0,100;0,200;0,300;0,400;0,500;0,600;0,700;0,800;0,900
;1,100;1,200;1,300;1,400;1,500;1,600;1,700;1,800;1,900&display=swap');

/* Apply this family to everything. */
body { font-family: 'Work Sans', sans-serif; }

/* Size is the first thing you'll want to adjust. */
li:nth-child(1) { font-size: 28px; }
li:nth-child(2) { font-size: 10px; }
/* Browsers usually default to 16px. */

/* Then the space between your lines, relative to your font-size. */
li:nth-child(3) { line-height: 200%; }
/* This could also be written as "32px", or just "2". */

/* You can specify different weights. */
li:nth-child(4) { font-weight: 100; }
li:nth-child(5) { font-weight: 900; }
/* These go from 100–900, depending on what your family has. */
/* Default is 400, and you'll see "bold"—which maps to 700. */

li:nth-child(6) { font-style: italic; } /* Default is "normal". */

li:nth-child(7) { text-transform: uppercase; } /* YOU CAN YELL */

/* This uses a relative unit, based on the font-size. */
li:nth-child(8) { letter-spacing: 0.1em; }
li:nth-child(9) { letter-spacing: -0.1em; } /* Negative values. */
/* Keep in mind these don't apply semantic meaning... */

li:nth-child(10) { text-decoration: underline; } /* Even non-links. */
li:nth-child(11) { text-decoration: line-through; }
/* Decoration should be paired with a tag, like <em> or <del>. */

li:nth-child(12) { text-align: center; } /* Default is "left". */
li:nth-child(13) { text-align: justify; }
li:nth-child(14) { text-align: right; }
```

# JUST WAIT... NEXT WEEK

But CSS can do so much more.  
Next week, we'll discuss using css  
to lay out webpages, and  
start to create more complex  
digital compositions.

**Break: 10 minutes**



# EXERCISE 1

Let's set up a pair of  
html & css documents as a class,  
and use @fontface  
to import a google font.



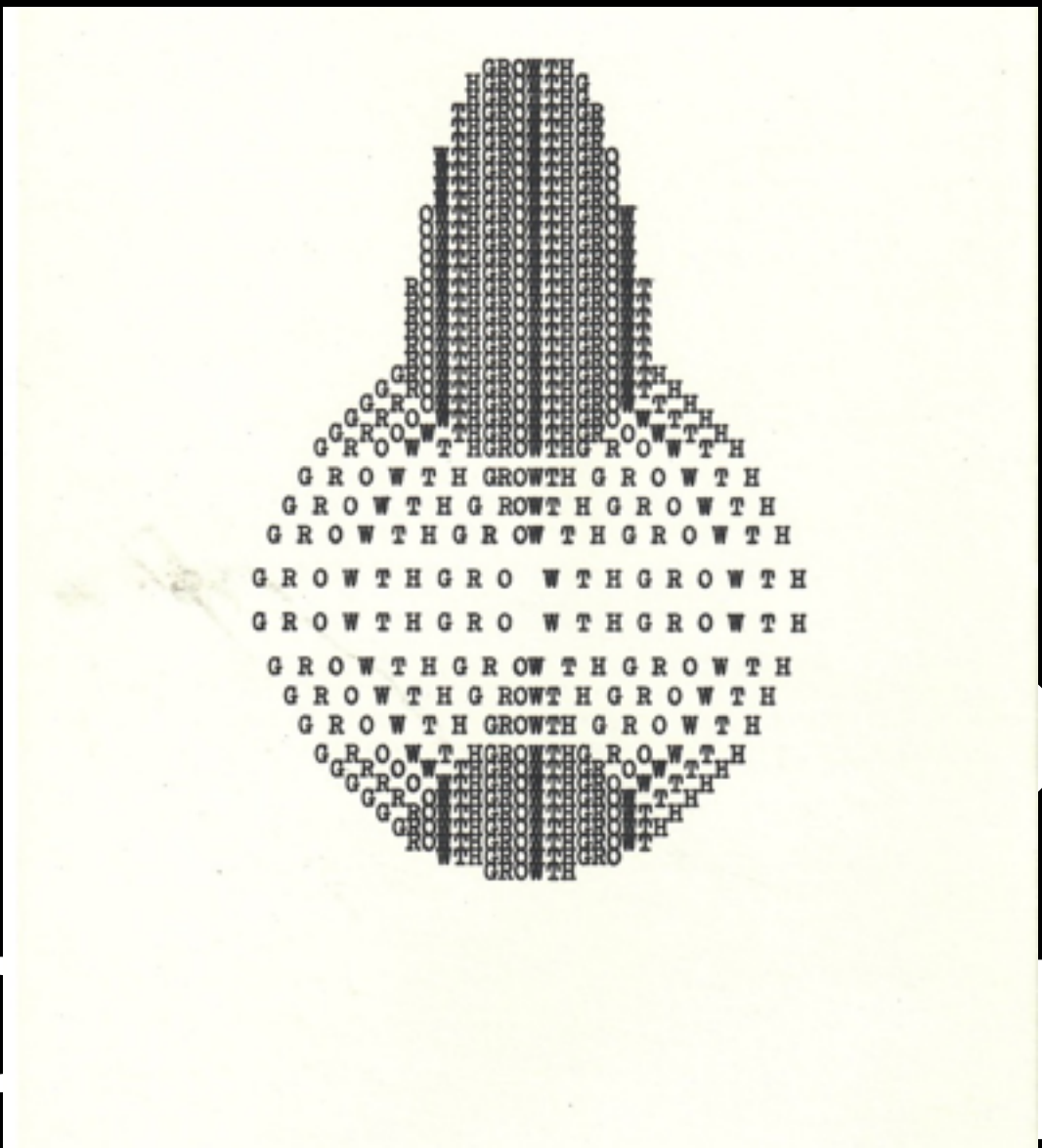
# EXERCISE 2

Compose concrete poetry on a web page. Feel free to source or write your own words, but if you pull text, make sure to attribute it explicitly or in the metadata:  
<meta name="author" content="John Doe">

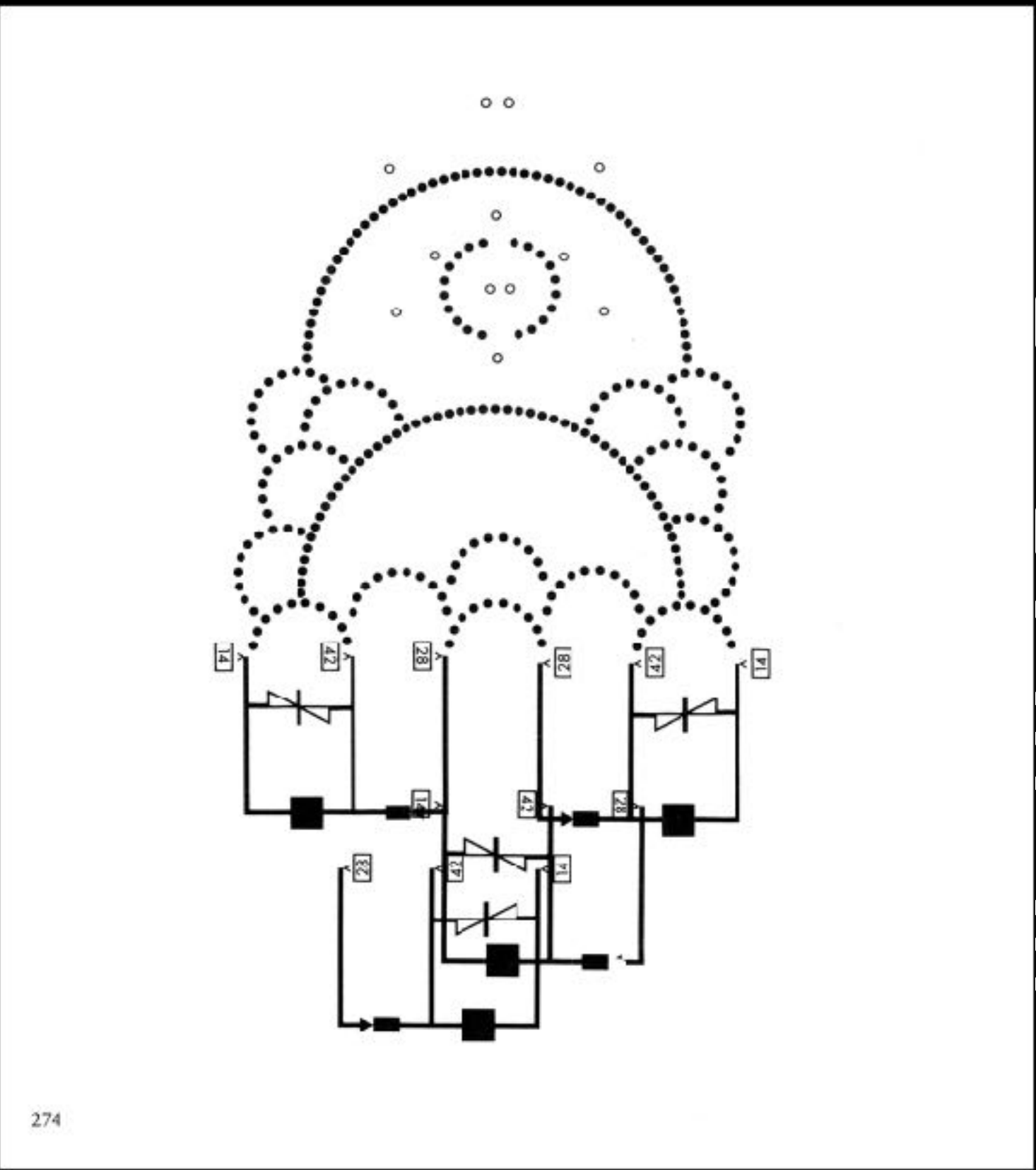
# EXERCISE 2



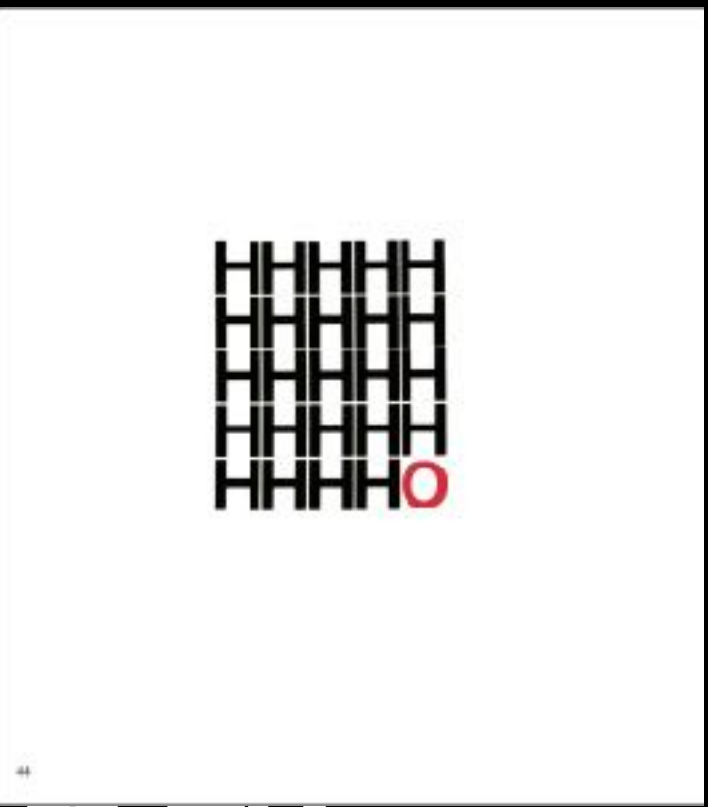
EL LISSITZKY



C/O SACKNER ARCHIVE



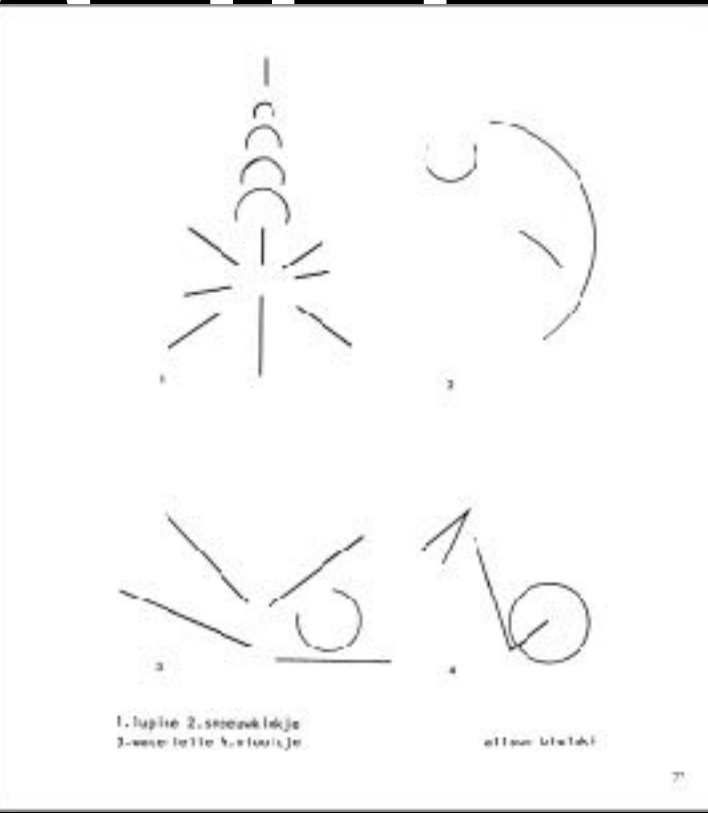
LILIANE LIJN



MIRELLA BENTIVOGLIO



AMANDA BERENGUE



ALISON BIELSKI

**SHARE?**

# QUESTIONS?

# **HOMEWORK FOR YOU**

- > Practice CSS in codecademy's Learn CSS course—part 1, Syntax and Selectors, and part 2, visual rules.
- > Watch Laurel Schwulst's Basics of CSS

# **FOR ME**

- > Harmonic Collection Entry 2

SESSION THREE  
SEPTEMBER 12, 2023

THANK YOU