CS122 Class 17: Visual & Web Design

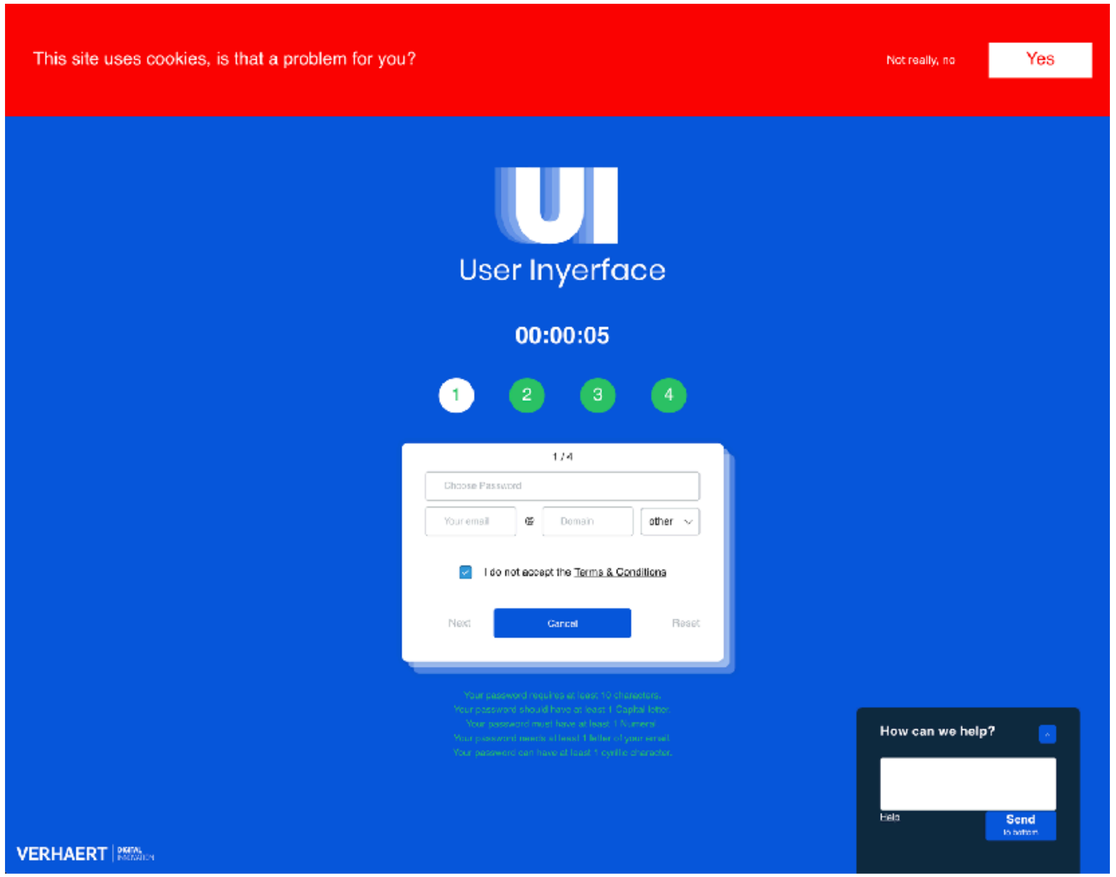
Class 17 agenda

- Zipcrit
- Lecture: Visual Design
- Lecture: Web Design
- Activity: Browse the HTML Review

Visual Design

Why do we need visual design?

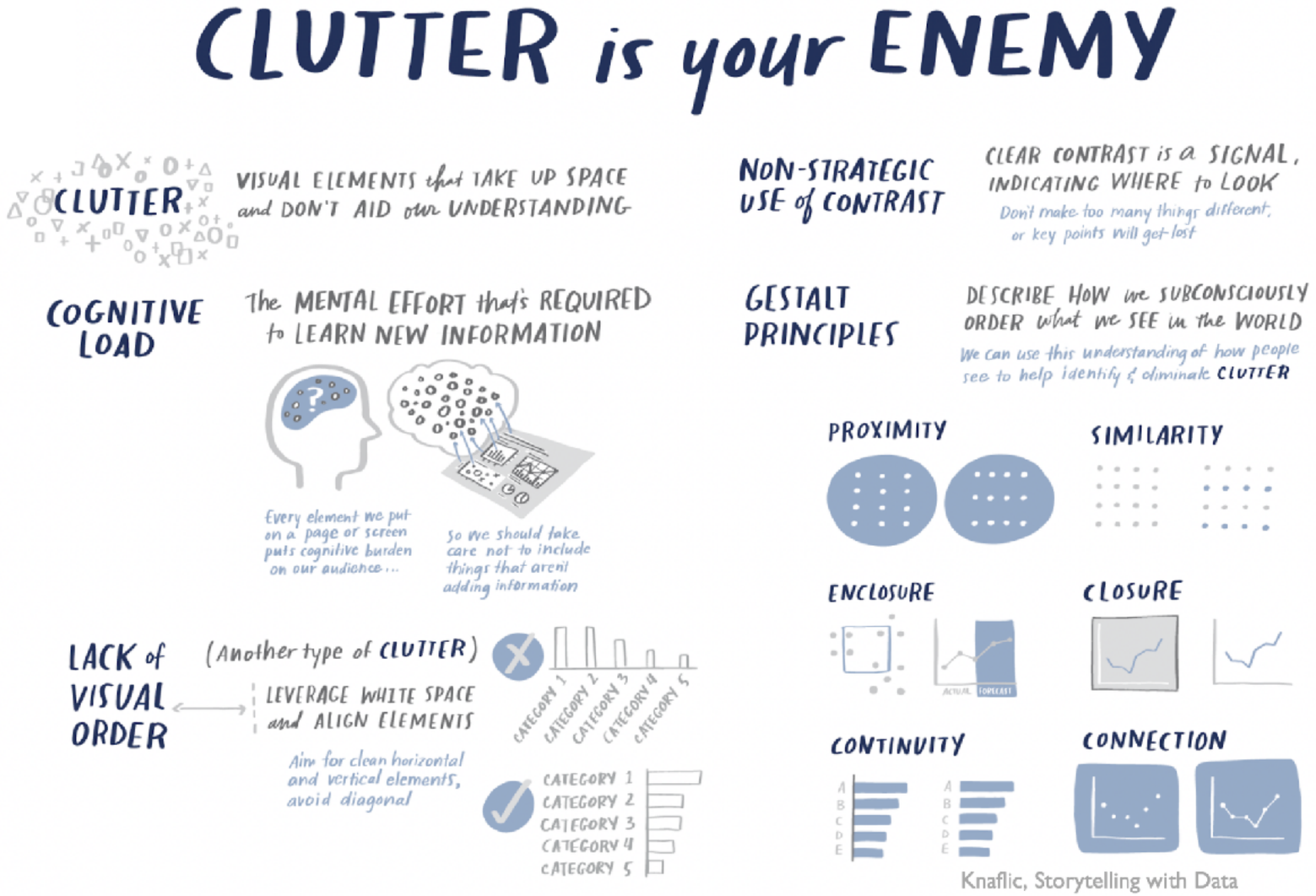
- “Aesthetic-usability effect”: people perceive that “aesthetic” interfaces are often more intuitive than less aesthetically pleasing ones
- Build user trust
- Communicate a norm or in-group aesthetic
- Visual design is a language and, like all languages, needs practice to be fluent



https://userinyerface.com/game.html

Principle: Avoid clutter

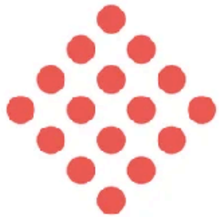
- “Clean” and “professional” designs minimize cognitive load by reducing clutter
- As mentioned in a previous class, may be a Western norm
- How to avoid clutter? We’ll cover basics of hierarchy, grids, color, typography



Knaflic, Storytelling with Data

Gestalt principles

- Gestalt principles, from psychology, explain how our brains perceive visual signals “at a glance”



Good Figure

Objects groupped together tend to be perceived as a single figure. Tendency to simplify.

Proximity

Objects tend to be grouped together if they are close to each other.

Similarity

Objects tend to be grouped together if they are similar.

Continuation

When there is an intersection between two or more objects, people tend to perceive each object as a single uninterrupted object.

Closure

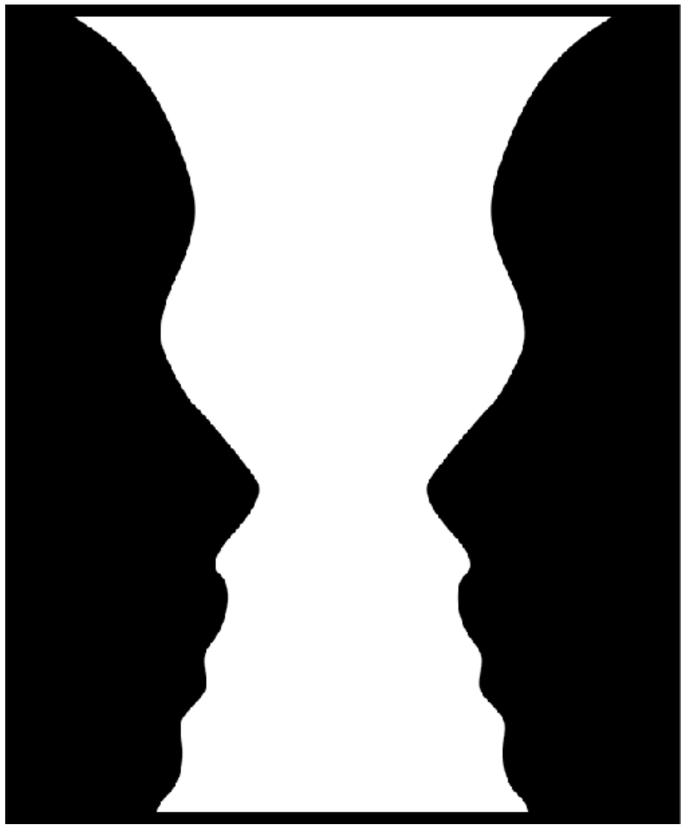
Visual connection or continuity between sets of elements which do not actually touch each other in a composition.

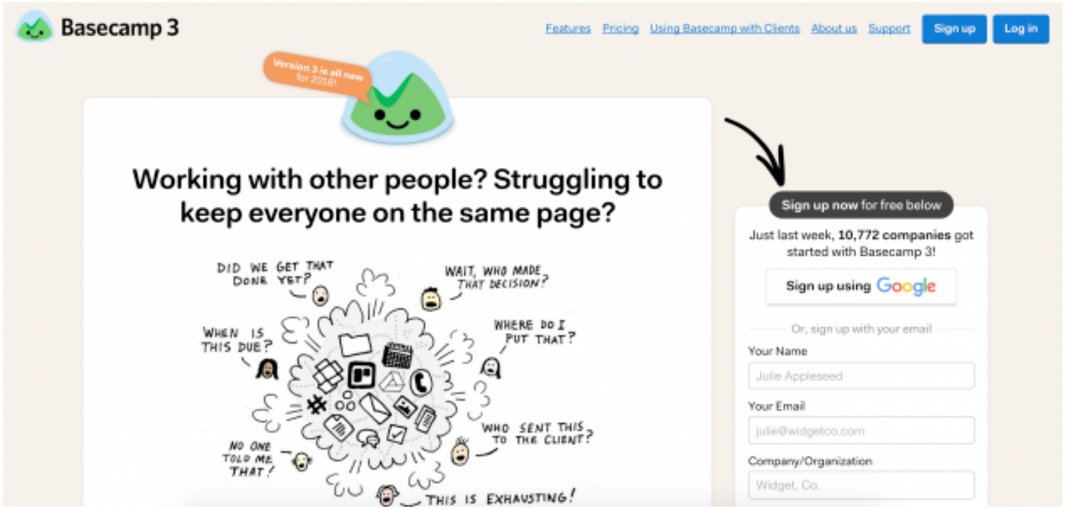
Symmetry

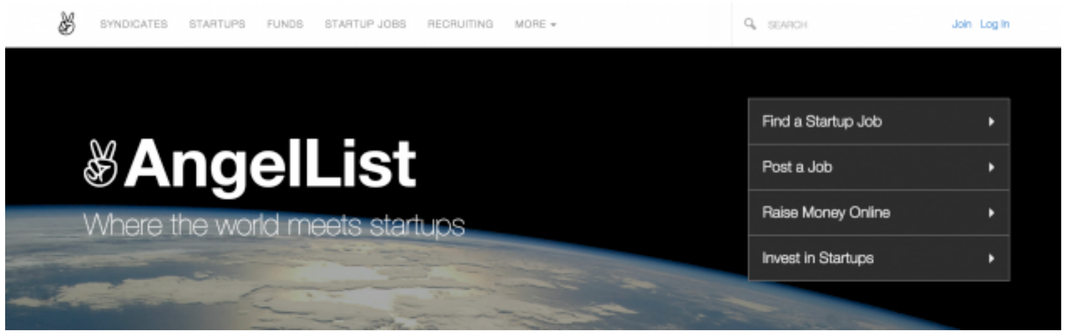
The object tend to be perceived as symmetrical shapes that form around their center.

Gestalt principle: Figure-ground

- Objects grouped into either foreground or background
- Your tool should have a distinct foreground (usually where the content is) versus background

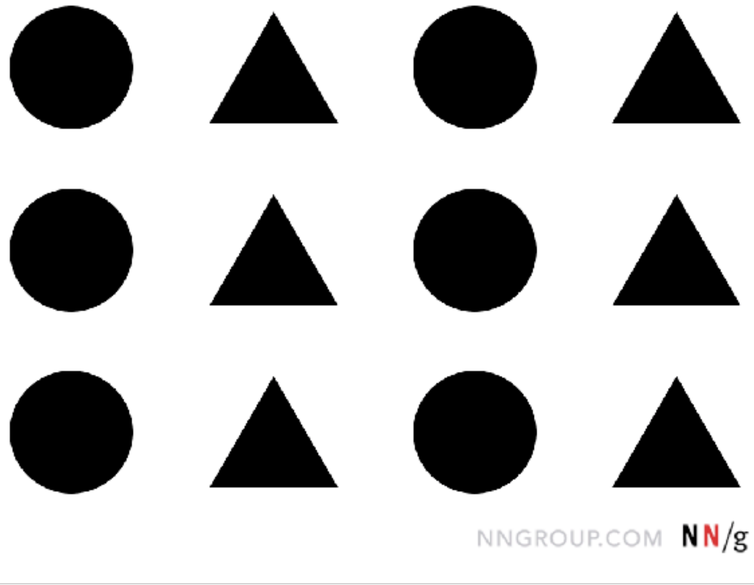


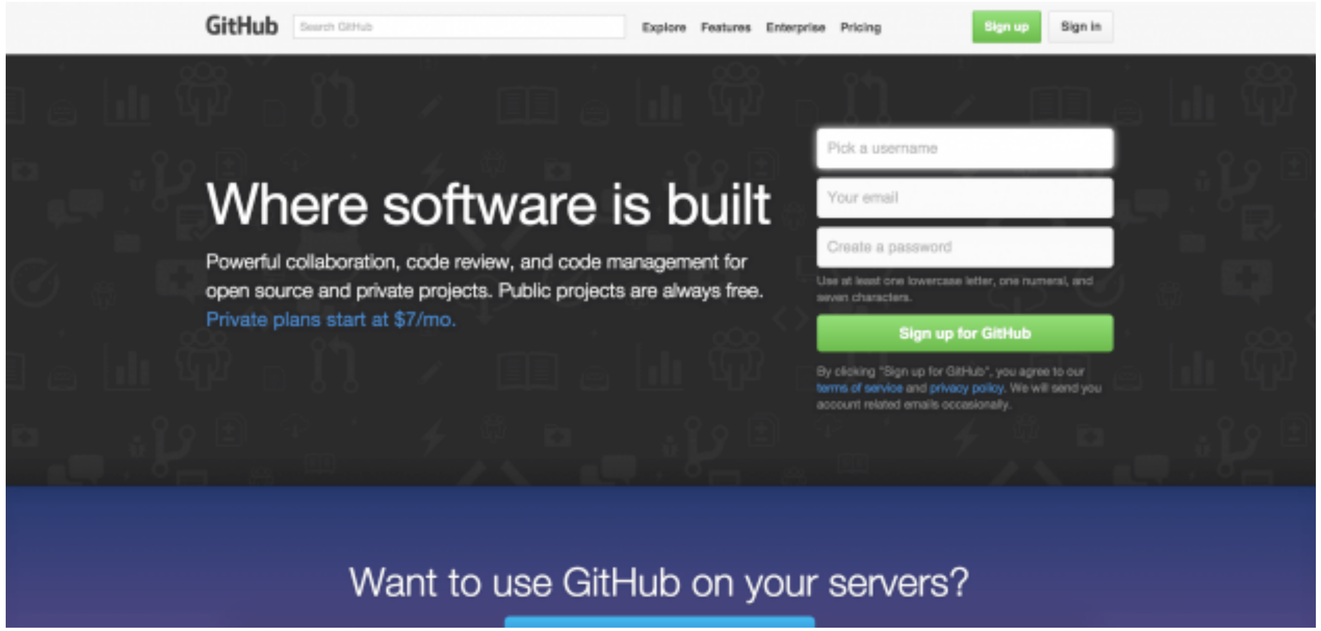




Gestalt principle: Similarity

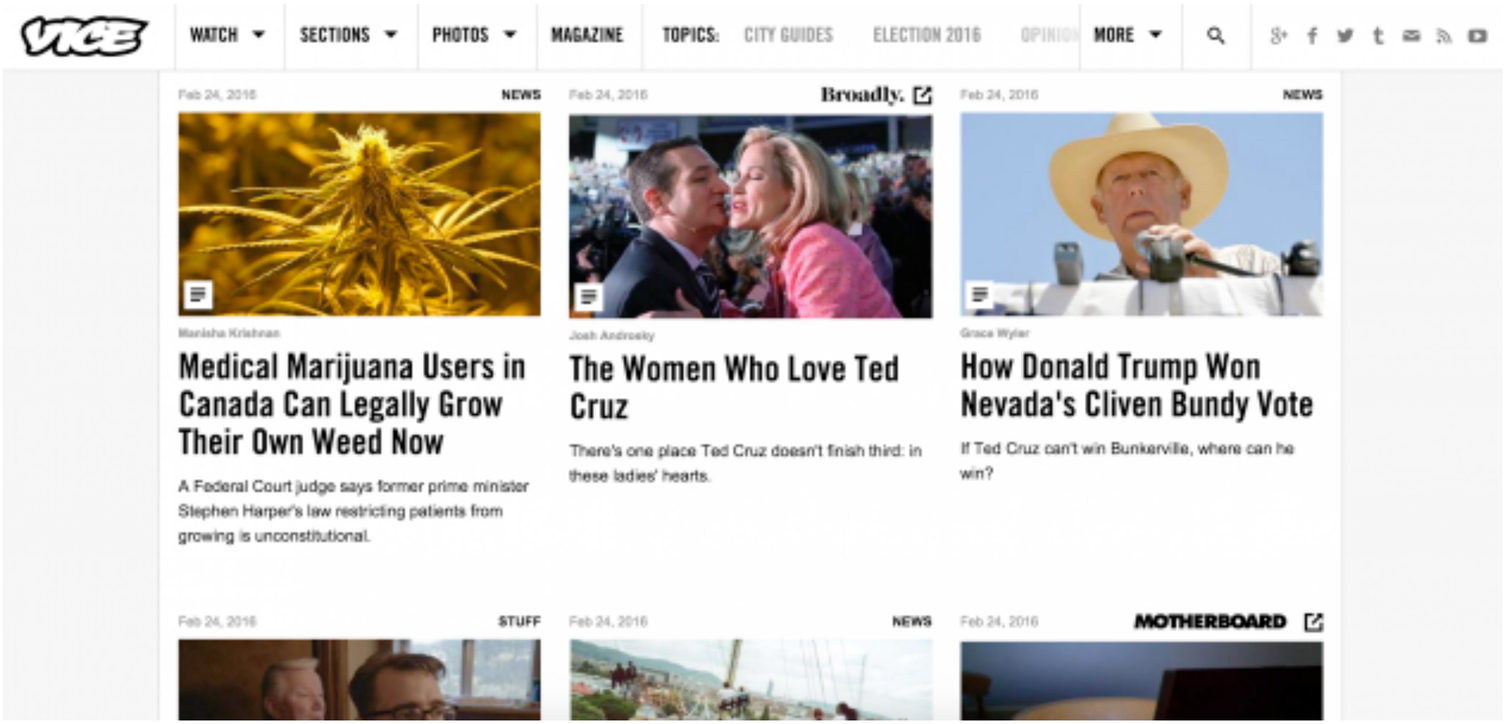
- Visually similar objects are grouped together
- E.g., three text input fields and then an action button that’s differently colored



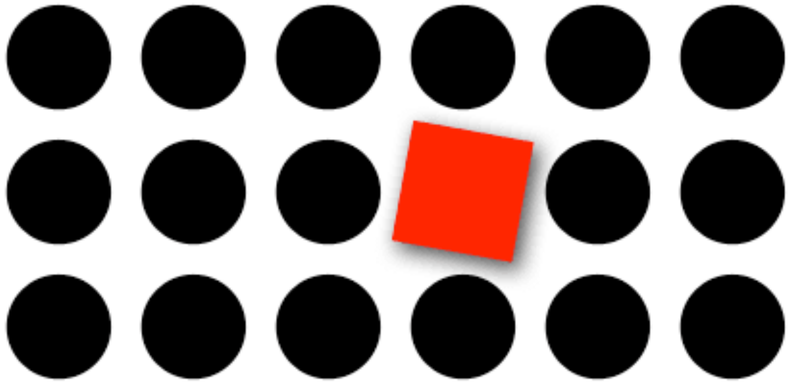


Gestalt principle: Proximity

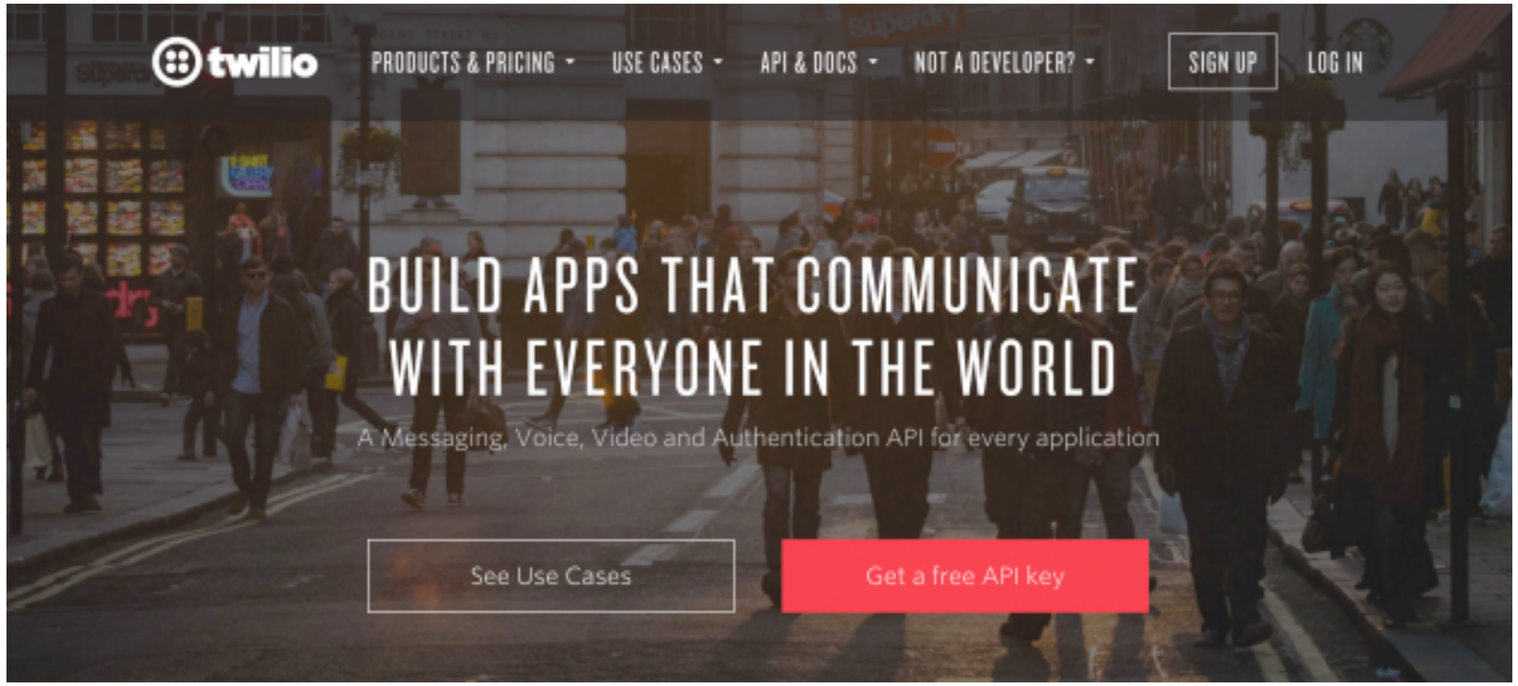
- Visually close objects are grouped together
- E.g., can use proximity to create “cards” that all have the same style of information



Gestalt principle: Focal point

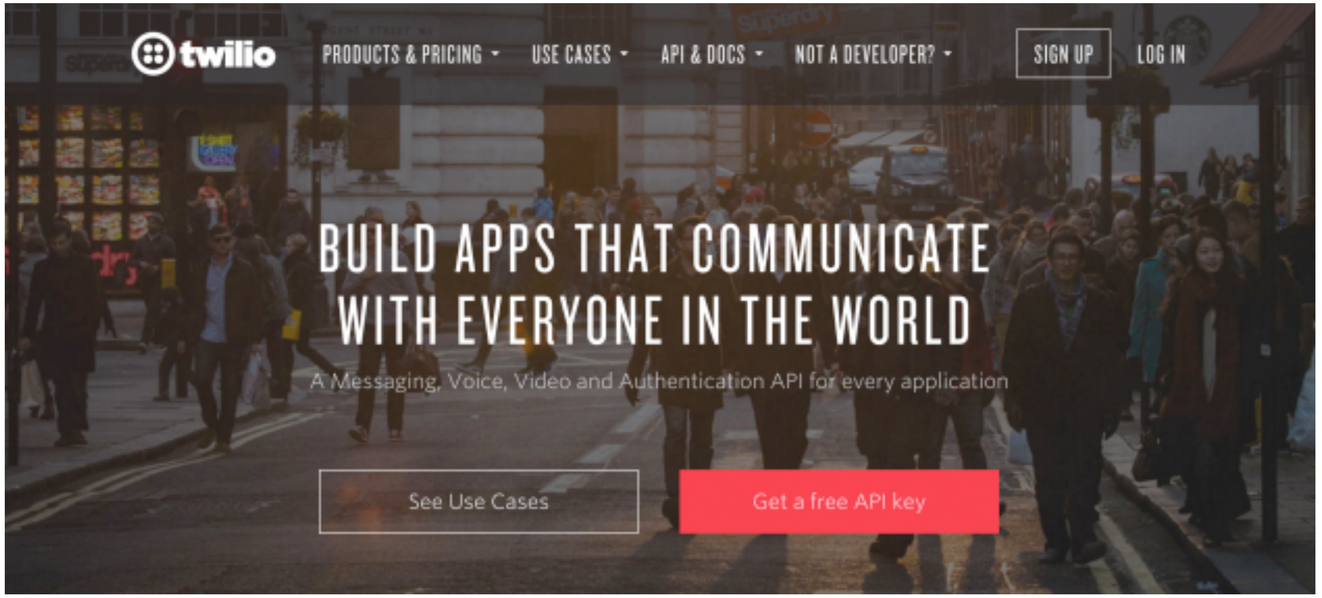


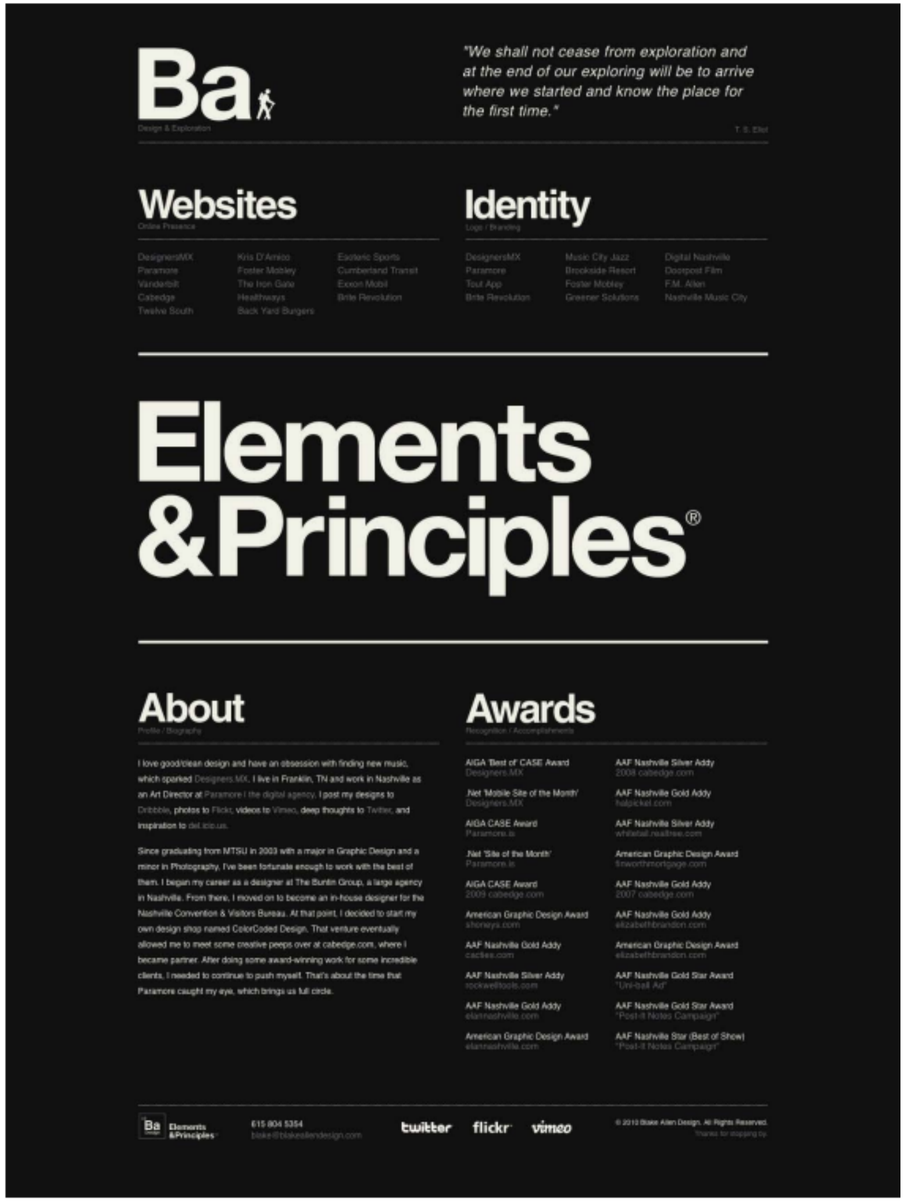
- Contrast drives attention
- Usually used for “call to action” buttons or the most dominant action in a user flow



Hierarchy

- What do you read first?
- A: The biggest content/font size
- Should make hierarchy decisions and keep them consistent between pages (e.g., all headings at 48pt, all body text at 16pt)
  - Look into “em” unit in CSS - 1em is body font size, so here, a heading would be 3em





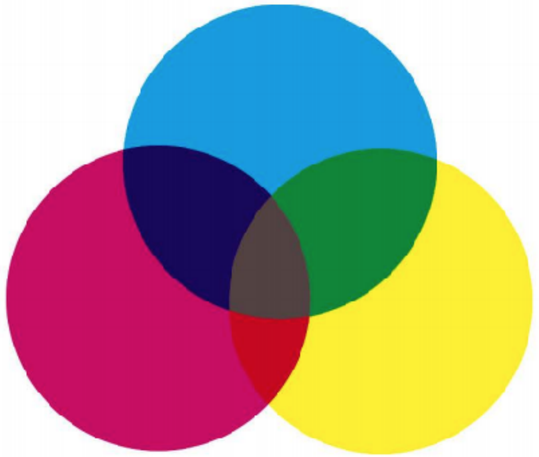
Grids

- An underlying grid-based design can provide structure and alignment
- How-to:
  - Overlay a grid on your design
  - Make sure content aligns with guides
  - Work with constraints and restrictions
  - Pay attention to spacing

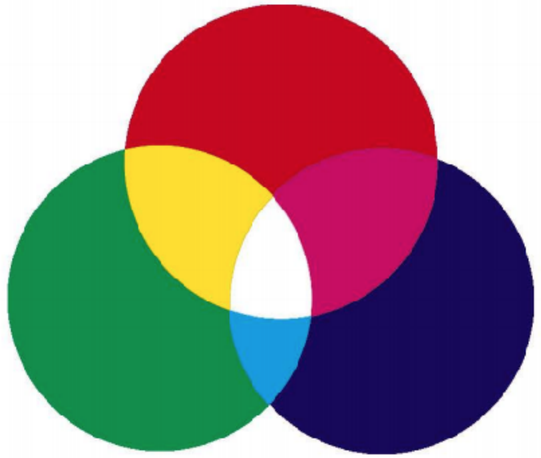
https://webdesign.tutsplus.com/all-about-grid-systems--webdesign-14471a

Color

- In design, what often matters is not a single color in isolation but how they are perceived *together*
- Many tools online to help you pick matching, visually appealing color palettes
- Color is cultural - what do you want to connote with your color scheme?
- **Accessibility** is important with color: make sure the value contrast is high enough, and you can put your design through colorblind filters online



Subtractive color (CMYK)

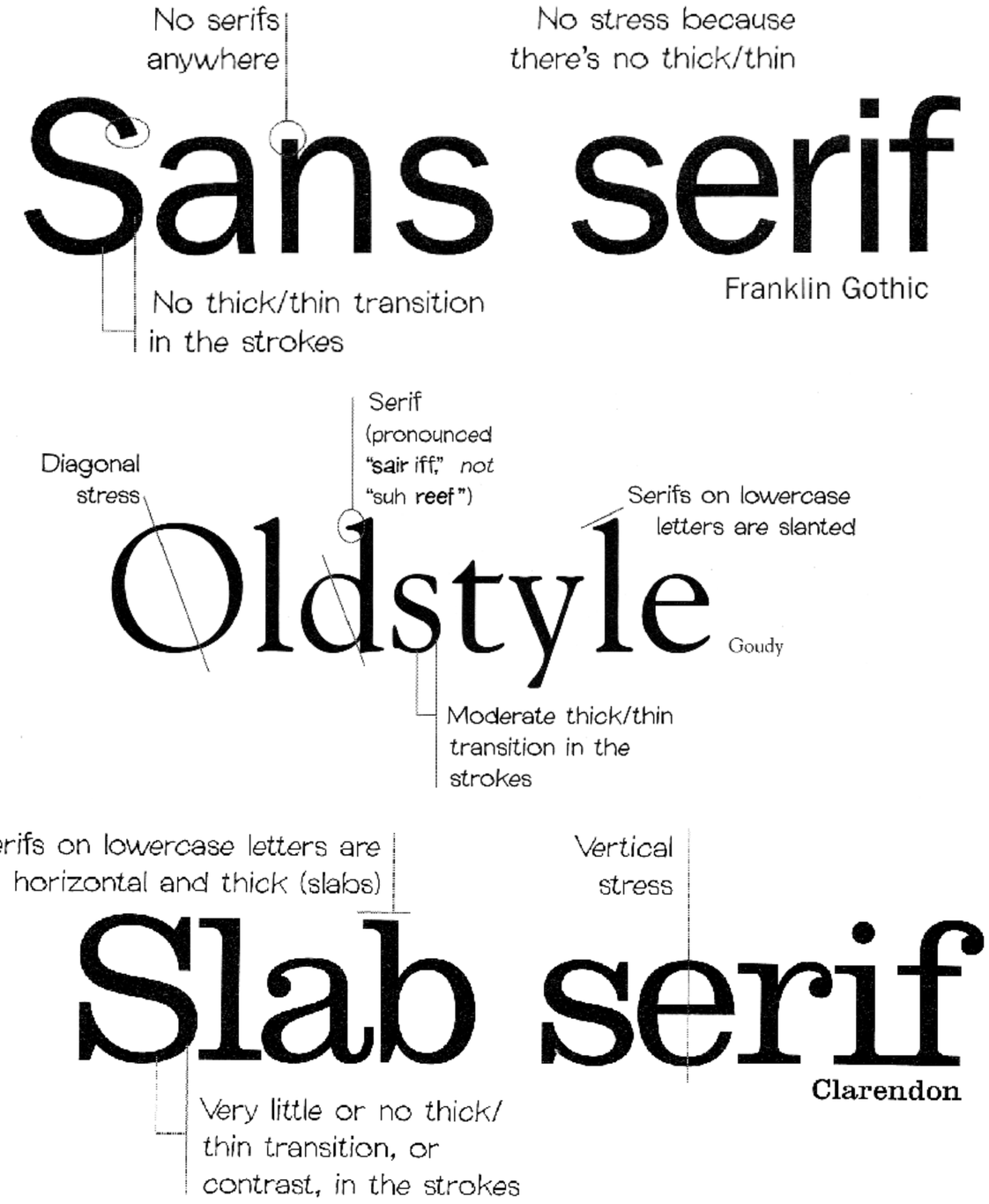


Additive Color (RGB)

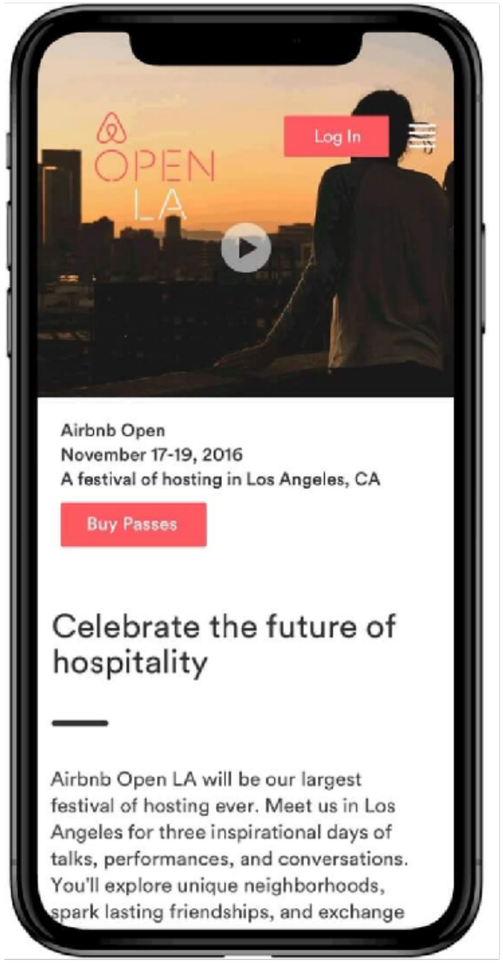
Josef Albers

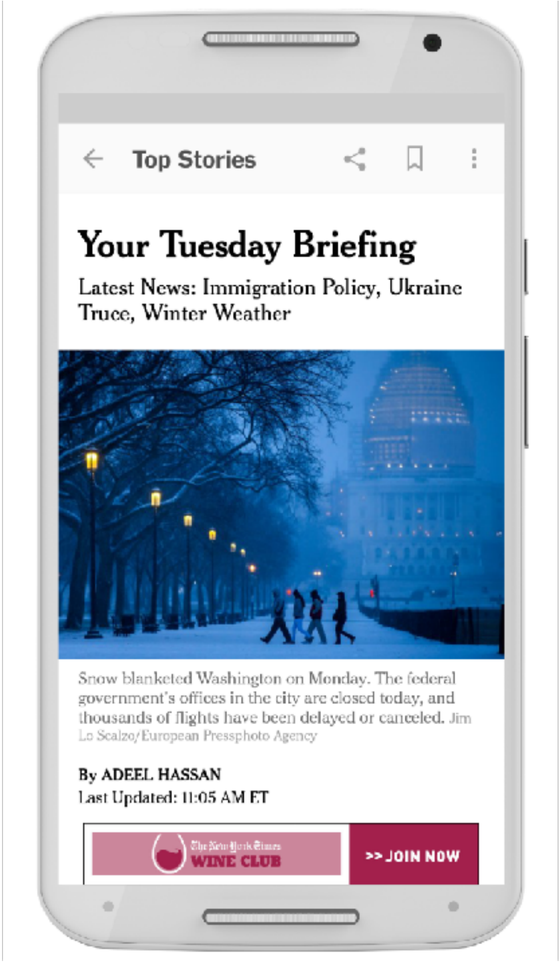
Typography

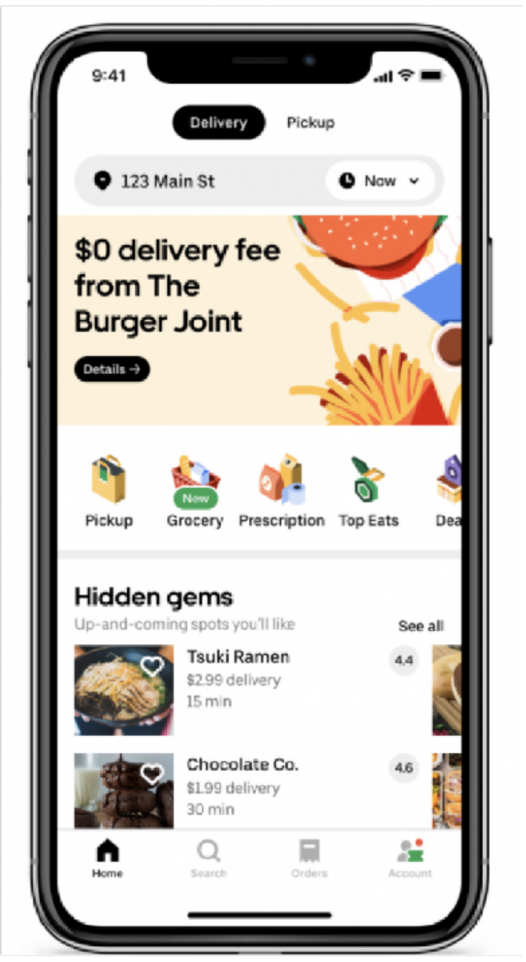
- Principles of typography from R. Bringhurst:
  - Typography exists to honor content
  - Read the text before designing it
  - Choose a typefaces that matches the character of the text



Choosing fonts that fit the “vibe”







Resources: Google Font pairings (recommended, free & easy to embed in web pages), Adobe fonts, Typewolf

Styles & weights

Roboto Thin & *Thin Oblique*

Roboto Light & *Light Oblique*

Roboto Regular & *Oblique*

Roboto Medium & *Medium Oblique*

Roboto Bold & *Bold Oblique*

Roboto Black & *Black Oblique*

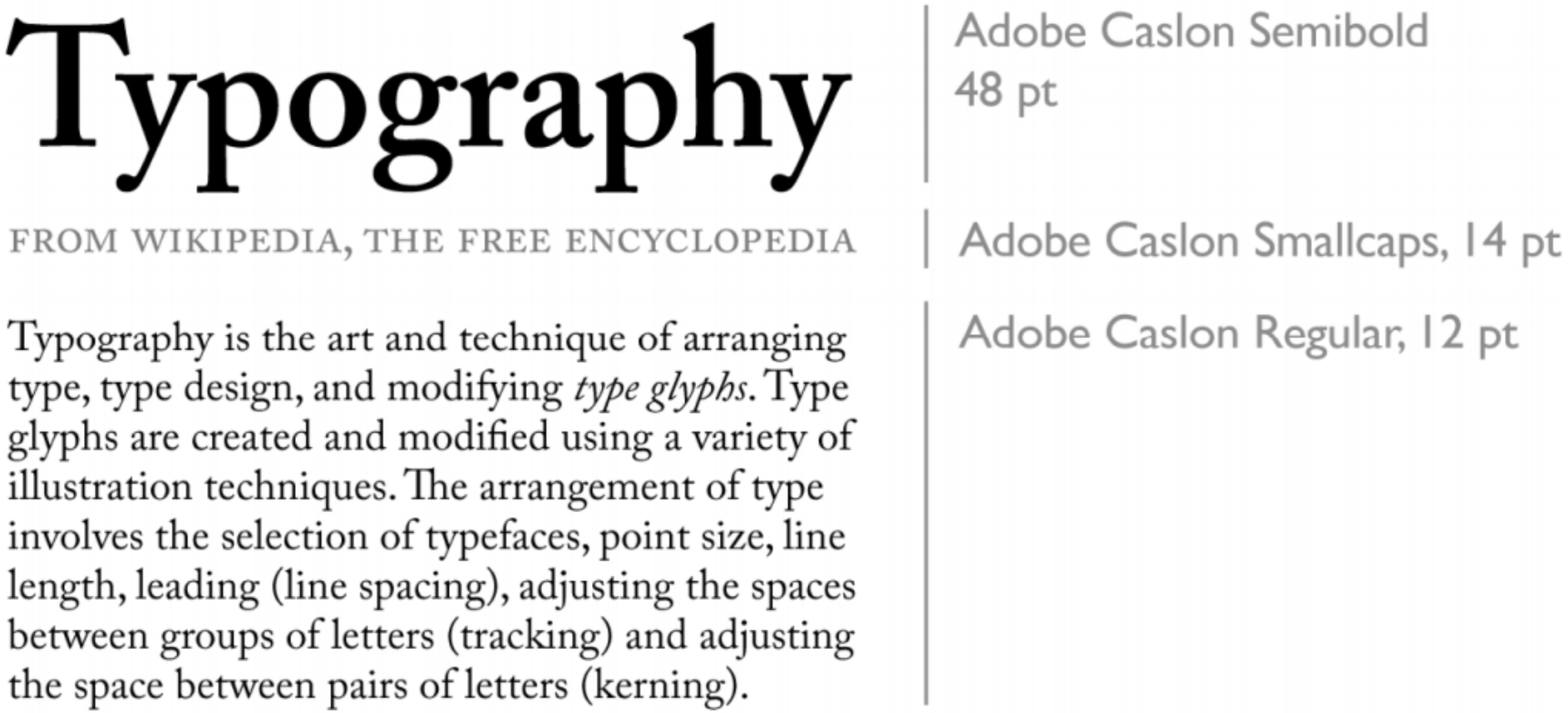
Roboto Condensed Light & *Condensed Light Oblique*

Roboto Condensed & *Condensed Oblique*

Roboto Condensed Bold & *Condensed Bold Oblique*

Combining Type: Concordant

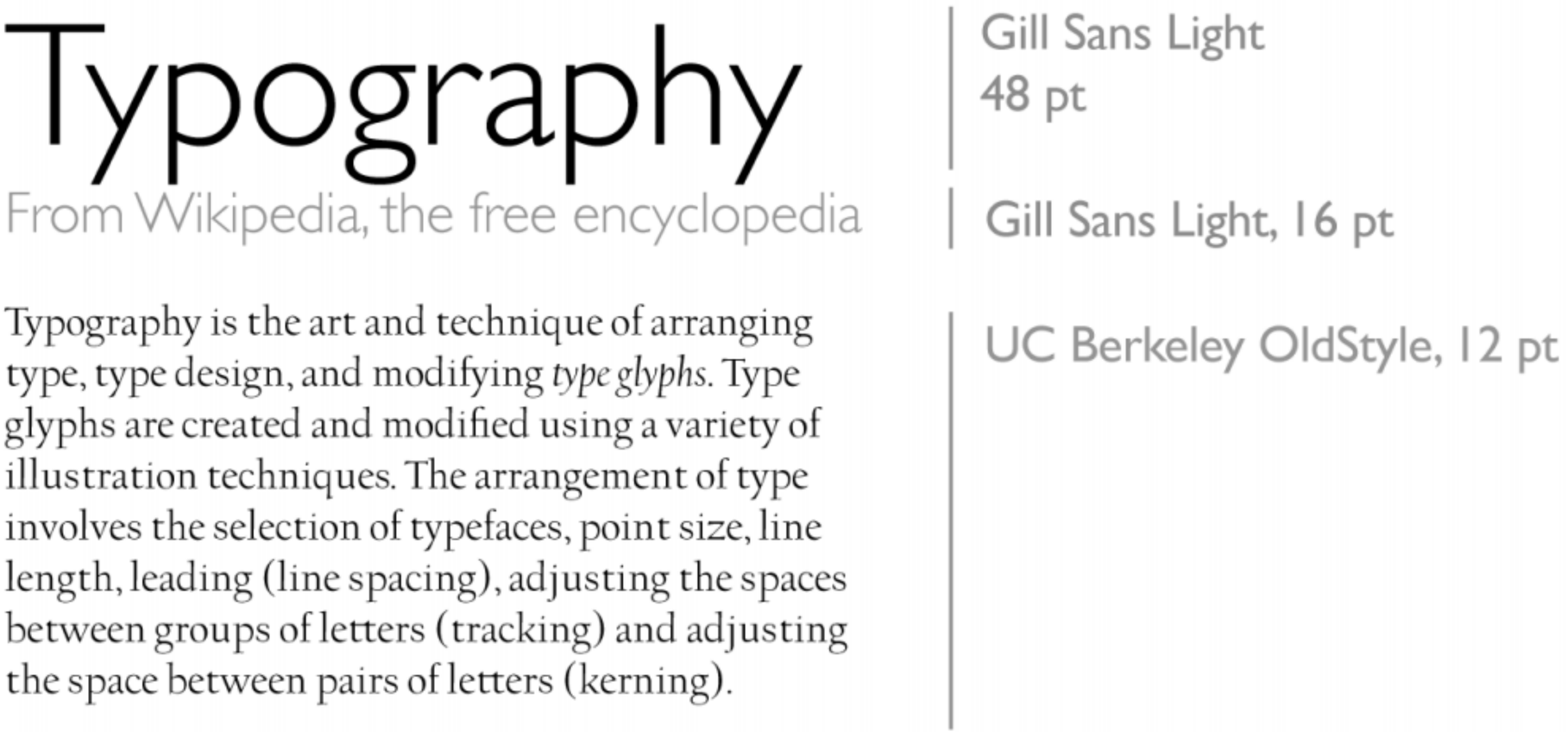
- Concordant = just one typeface with different weights



https://www.sitepoint.com/typography-concordant-contrasting-conflicting/

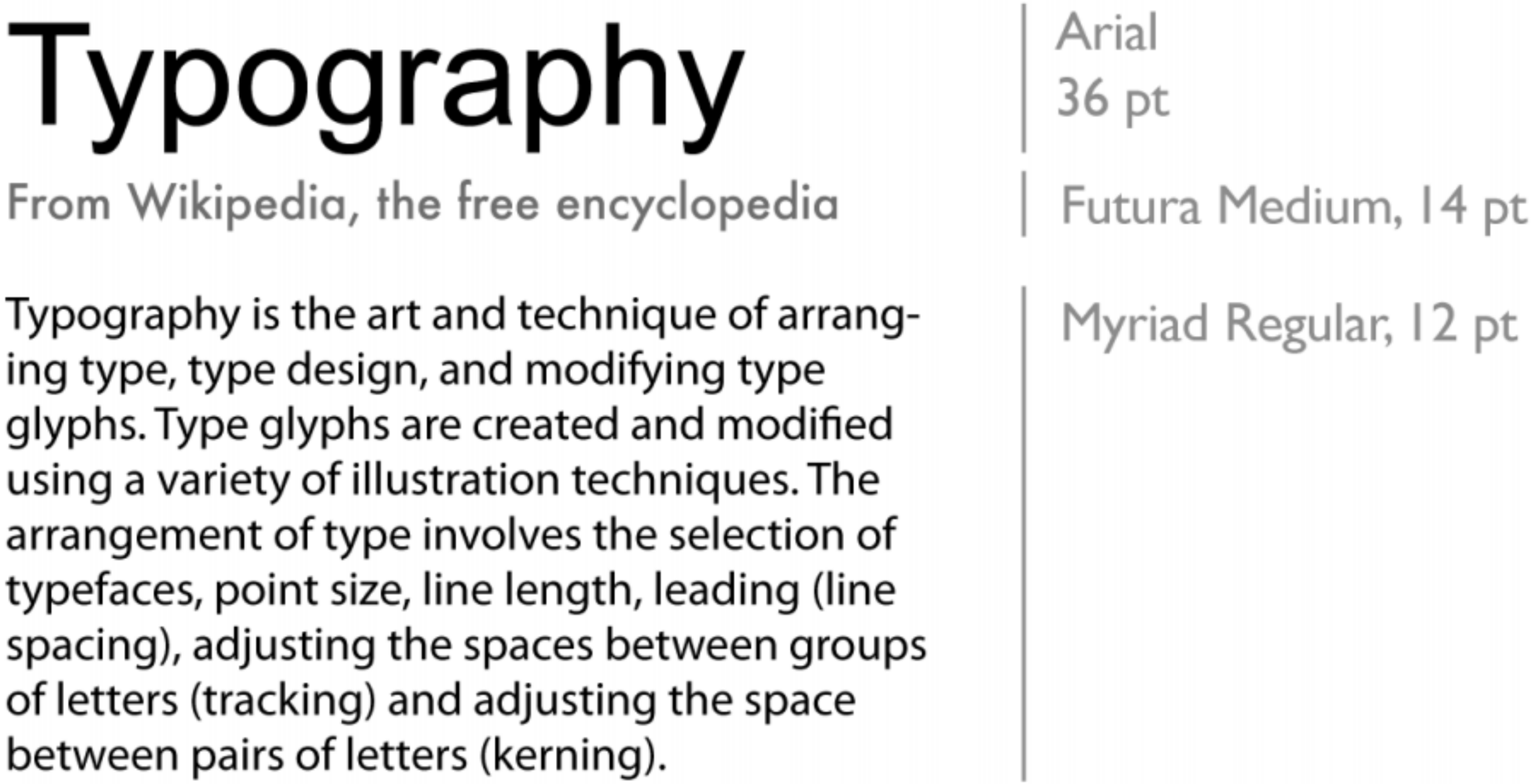
Combining Type: Contrasting

- Contrasting: purposefully picking different typefaces that are substantially different and match



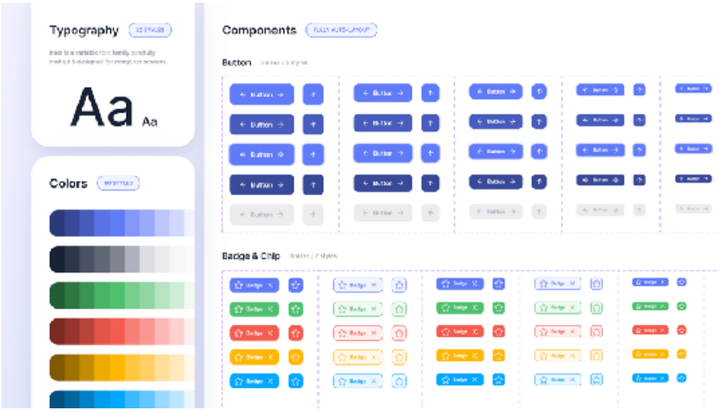
Combining Type: Conflicting

- Conflicting: different typefaces are too visually similar, does not “say anything” in visual communication



Design system

- What are your type faces for headings vs body text? What are your main vs accent colors? Do you have a logo? How do you establish a “brand identity”?
- Think of this for your WoZ prototype!
- E.g., pre-made design system Figma template: https://www.figma.com/community/file/1267195373409722424



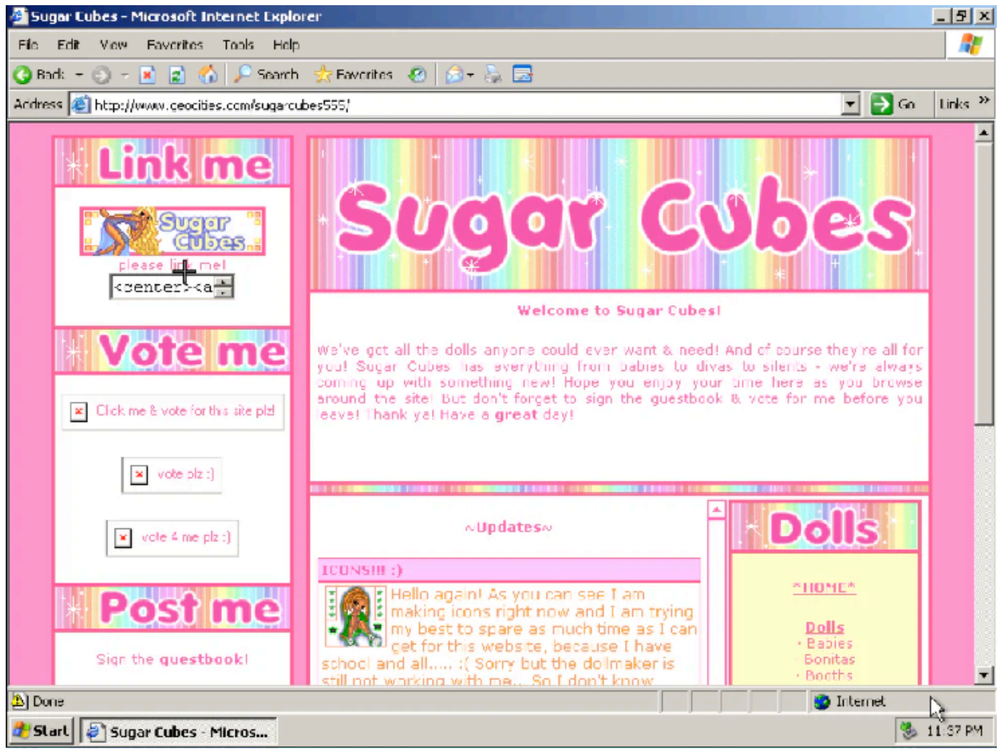
Visual design: Summary

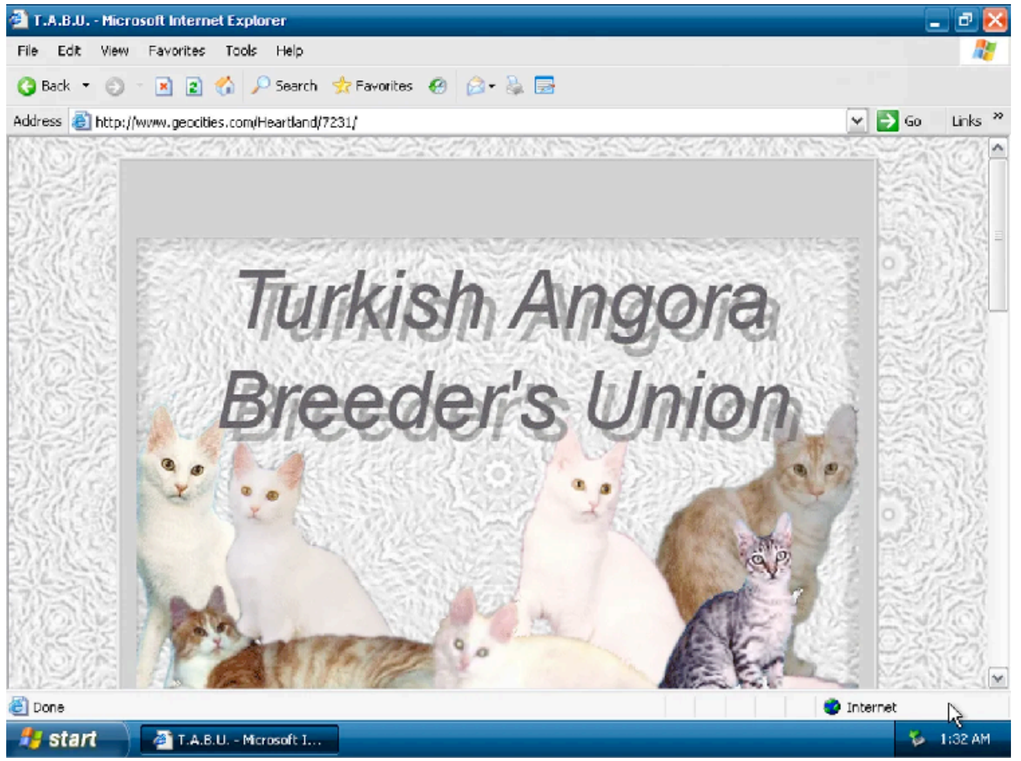
- Visual design is a language. To get better, we practice. Some tips:
  - Express hierarchy through size and weight; keep decisions consistent throughout screens
  - Use grids for underlying structure & alignment; learn the rules before you break them
  - Lean into contrast for visual interest, avoid ambiguous differences
  - Make sure color choices are accessible with enough value difference
  - Commit to a design system and keep it consistent!

Web Design

A brief biased history of web technologies

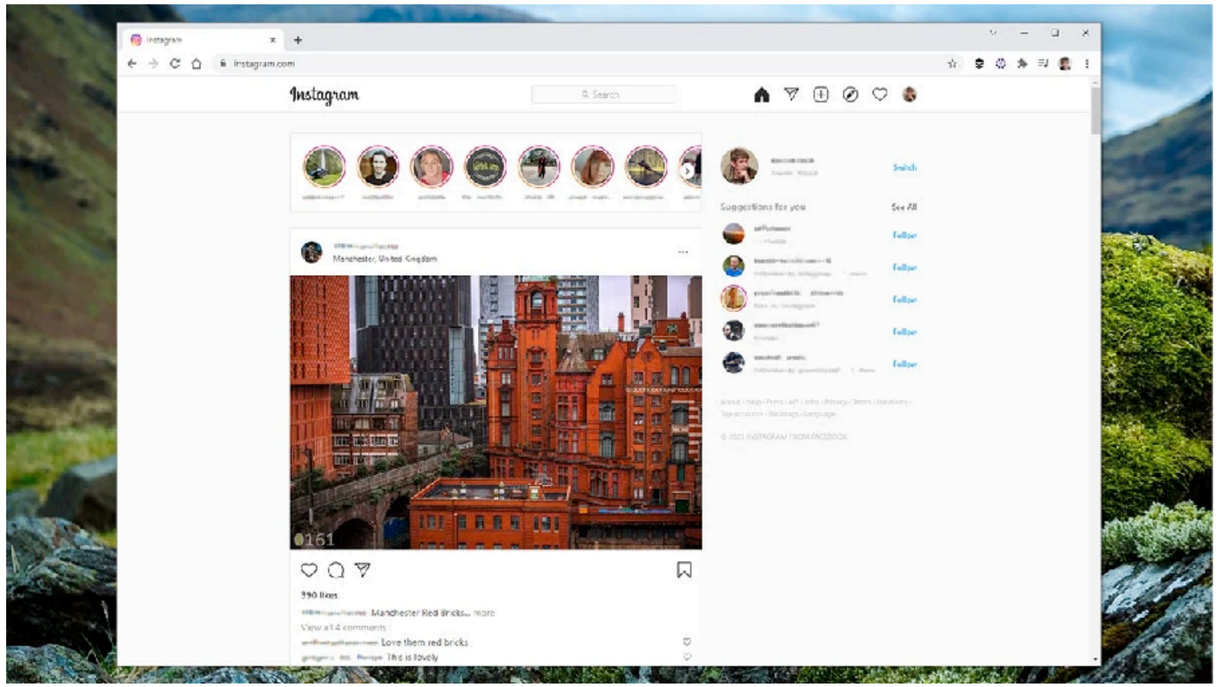
- Early websites were written in raw HTML/CSS/Javascript (sometimes), making source code very readable, easily learnable/sharable
- Early websites' goals were to share personal interests, not monetize or sell user data

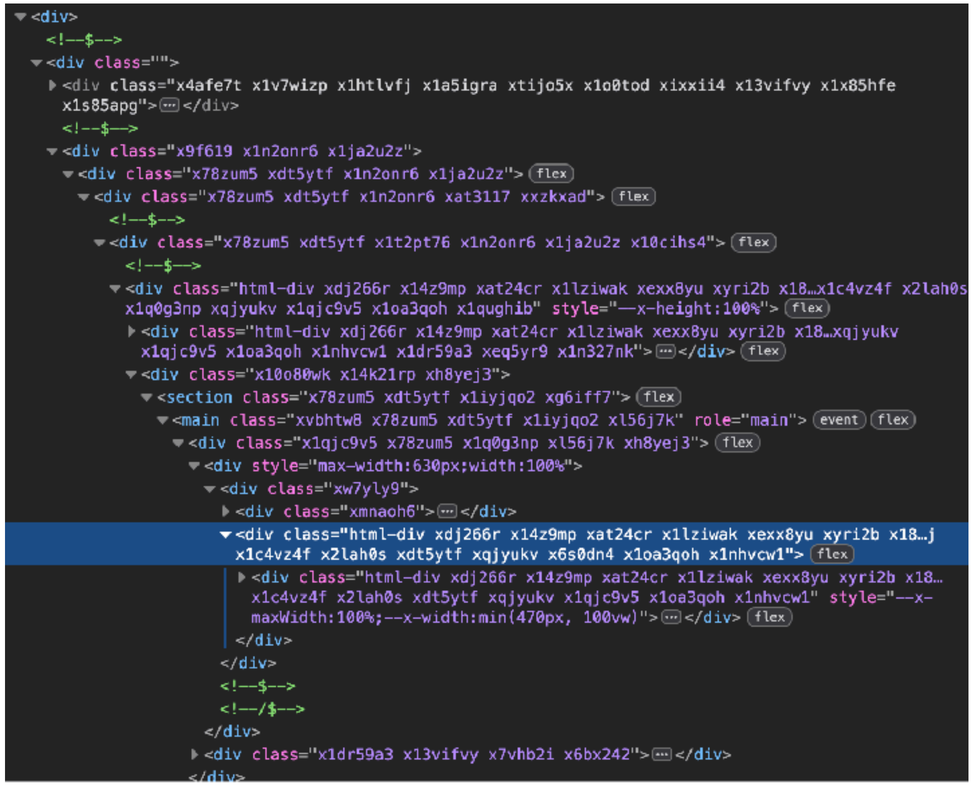




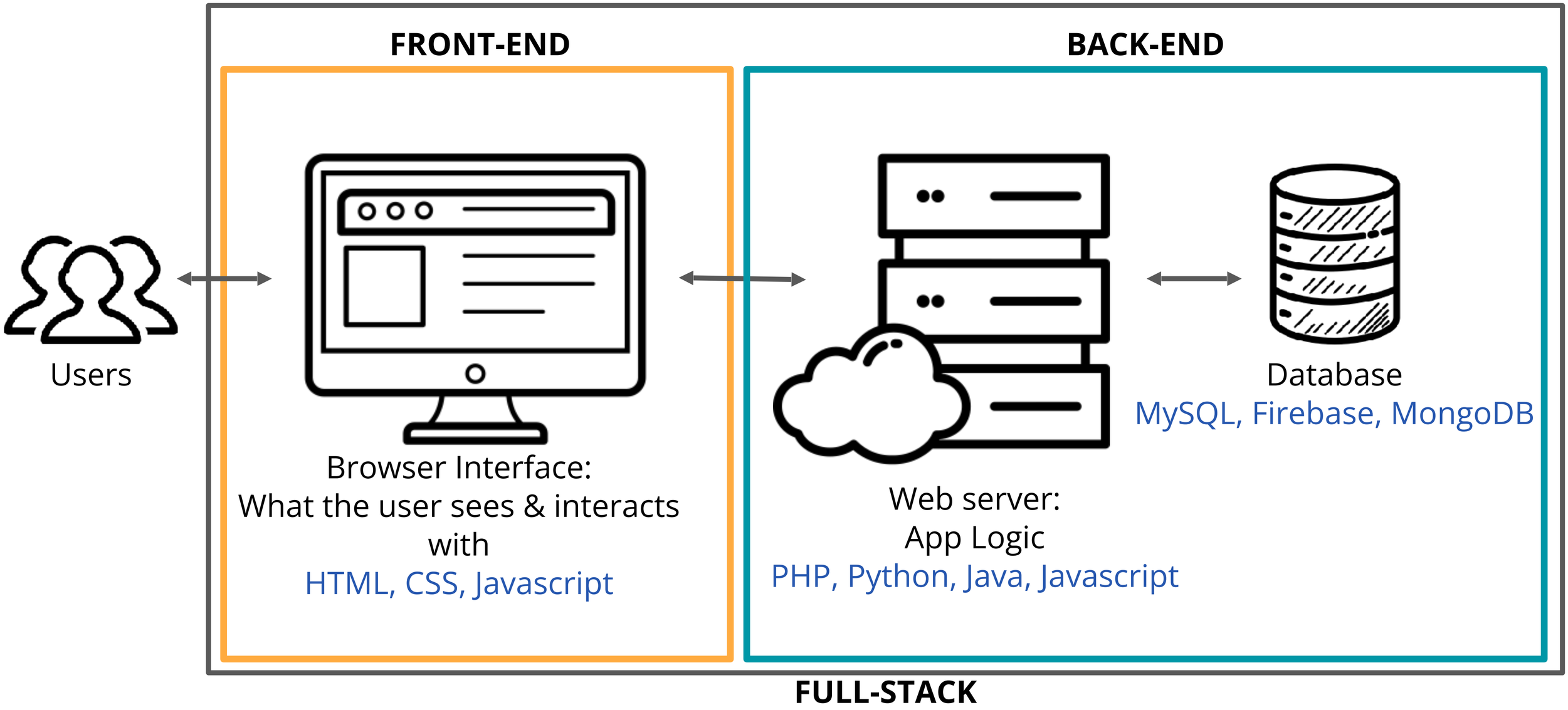
A brief biased history of web technologies

- As websites got more advanced, developers were pushing against the limits of front-end technologies and HTML
- React “compiles down” to JS/HTML, making modern source code unreadable





Nowadays, front-end vs back-end



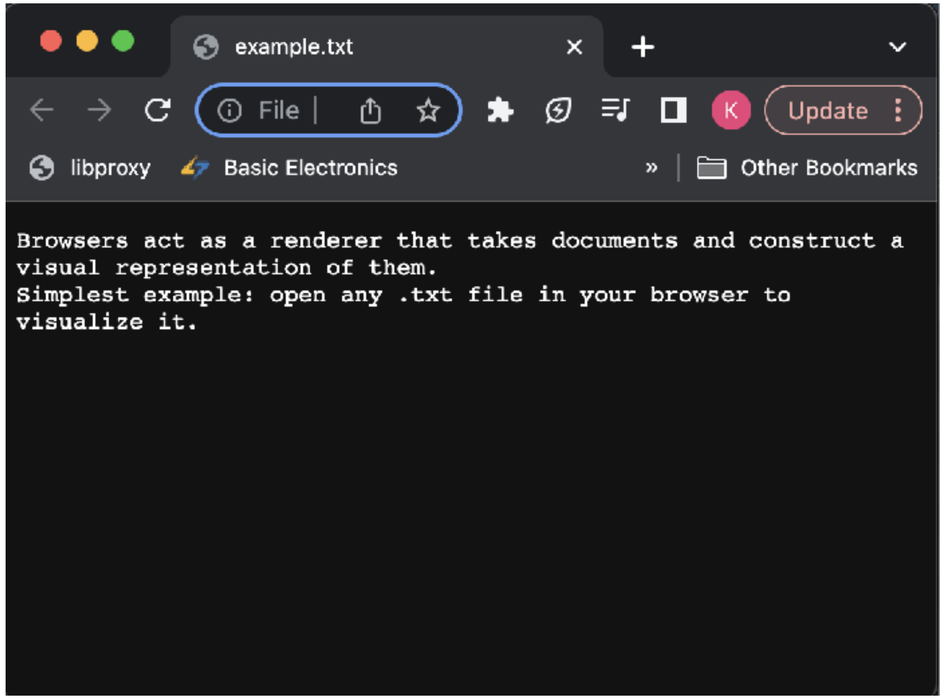
Core technologies

- HTML (Hyper Text Markup Language)
- CSS (Cascading Style Sheets)
- JavaScript (Not related to Java)

HTML

HTML is a markup language

- Browsers act as renderers that takes documents and construct a visual representation of them.
- Simplest example: open any .txt file in your browser to visualize it.
- Problem: text documents without any formatting tend to be hard to read for the user (and quite boring).
- HTML was created to give the text some format.



HTML is a markup language

- HTML is a markup language, NOT a programming language. Its purpose is to give structure to the content of the website, not to define an algorithm.
- It uses a series of nested tags (it is a subset of XML) that contain all the website information (like texts, images and videos). Ex: `<h1>This is a big header text</h1>`
- The HTML defines the page structure. A website can have several HTML files for different pages.

```
<html>
    <head>
    </head>
    <body>
        <div>
            <p>Hi</p>
        </div>
    </body>
</html>
```

HTML: Basic Rules

Some rules about HTML:

- It uses tags with attributes:

```
<tag_name attribute="value"> content </tag_name>
```

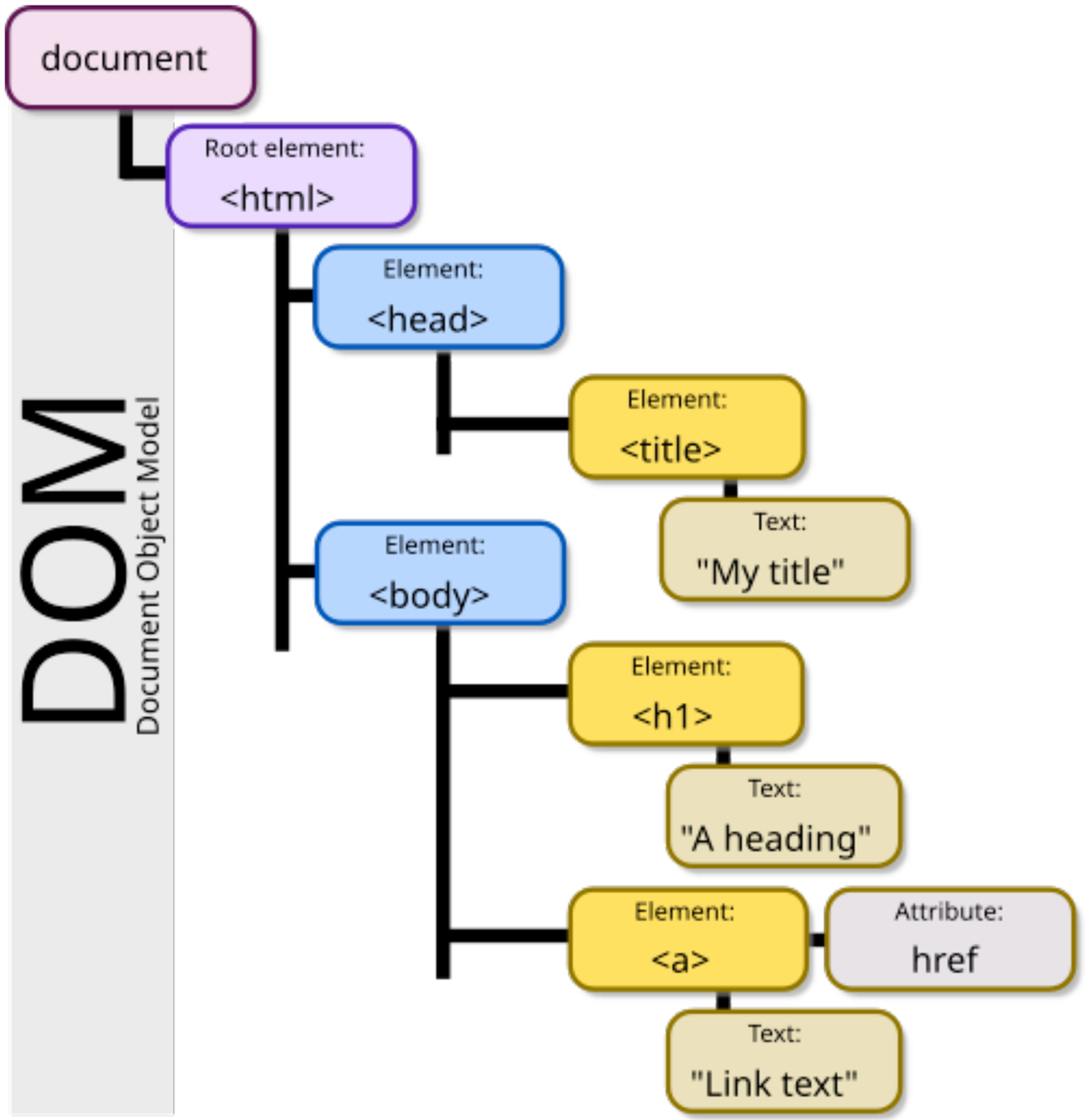
- It stores all the information that must be shown to the user.
- There are different HTML elements for different types of information/behavior.
- The info is stored in a structure called the DOM (Document Object Model).
- It gives the document semantic structure (e.g. this is a title, this is a section, this is a form) for computers and screen readers to understand web content.
- It doesn’t contain most information related to how it should be displayed (that information belongs to CSS), so no color information, font size, position, etc.

HTML: Example

```
<div id="main">
    <!-- this is a comment -->
    This is text without a tag.
    <button class="mini">press me</button>
    <img src="me.png" />
</div>
```

Tags structure a DOM

- The DOM is the tree that contains all information about things on a webpage
- Each node can have one parent, but multiple children



HTML: Common Tags

Although there are lots of tags in the HTML specification, 99% of the webs use a subset of HTML tags with less that 10 tags. Some important ones:

- ** **: a container, usually represents a rectangular area with information inside.
- **<img/>**: an image
- **<a>**: a clickable link to go to another URL
- **<p>**: a text paragraph
- **<h1>**: a title (h2,h3,h4 are titles of less importance)
- **<input>**: a widget to let the user introduce information
- **<style>**: to insert CSS rules
- **<script>**: to execute Javascript
- ****: a null tag (doesn't do anything)

HTML: Interesting Tags

There are some tags that could be useful sometimes:

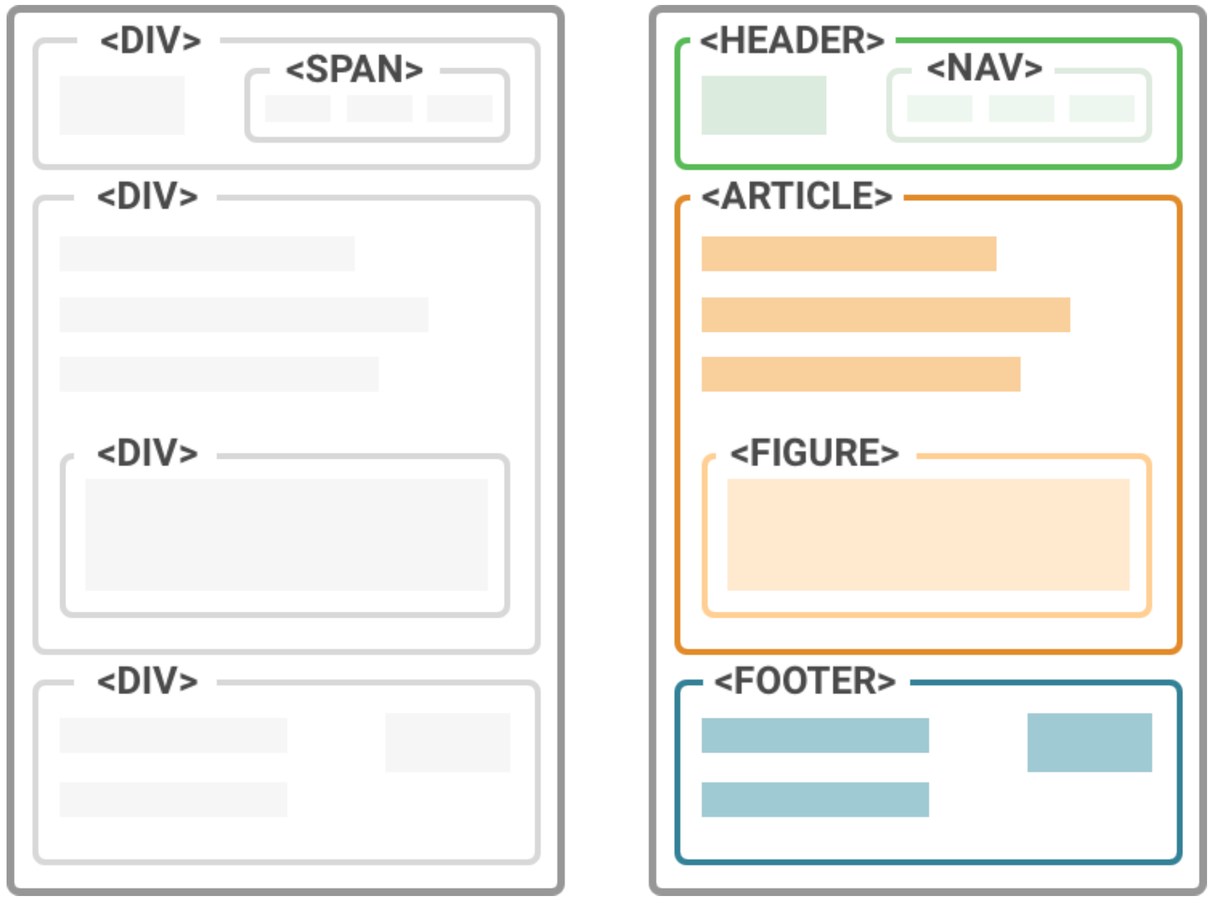
- **<button>**: to create a button
- **<audio>**: for playing audio
- **<video>**: to play video
- **<canvas>**: to draw graphics from Javascript
- **<iframe>**: to put another website inside ours

HTML: Use tags liberally!

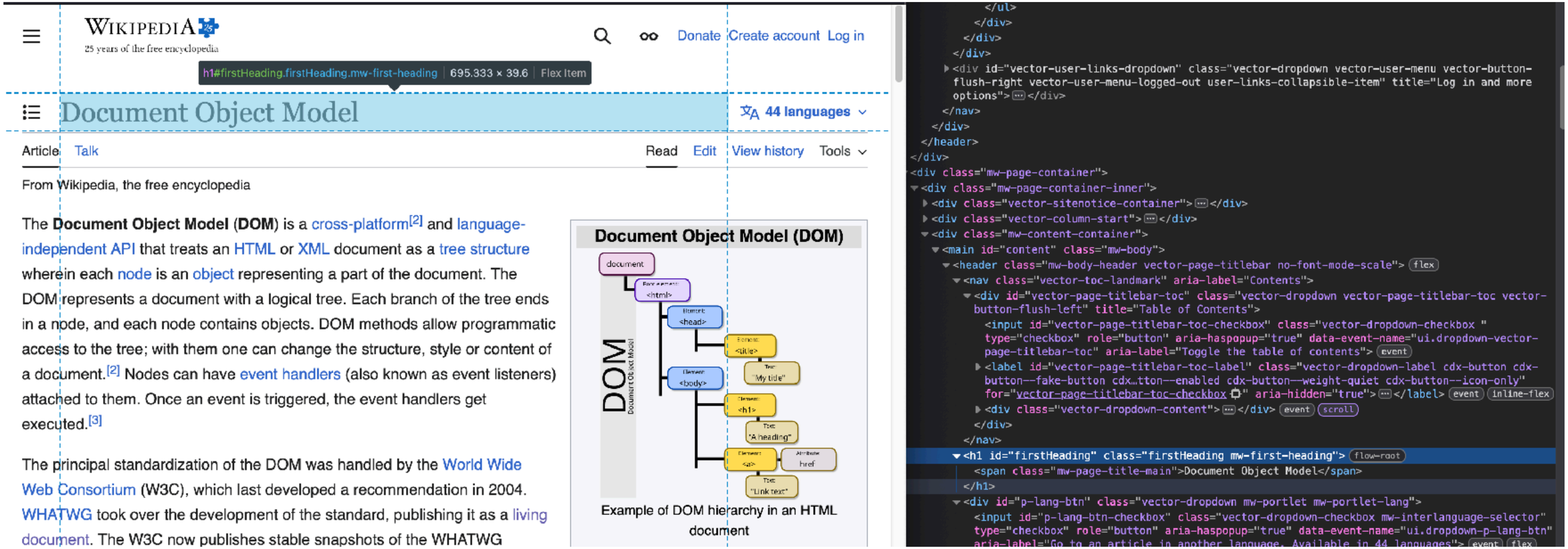
We use HTML tags to wrap different information on our site.

The more structure, the easier it will be to access, present, and modify it.

This info is also needed to make websites accessible (e.g. for screen readers)



Inspect the DOM with browser based developer tools



HTML: Use descriptive tags

Try to avoid doing this:

```
<div>
Title

Here is some content
Here is more content
</div>
```

DON'T DO THIS

Do this instead:

```
<div>
    <h1>Title</h1>
    <p>Here is content.</p>
    <p>Here is more content</
p>
</div>
```

HTML: id and class

We also can extend the code semantics by adding extra attributes to the tags:

- **id**: give a unique identifier for this tag
- **class**: gives a generic identifier for this tag

```
<div id="profile-picture" class="mini-image">...</div>
```

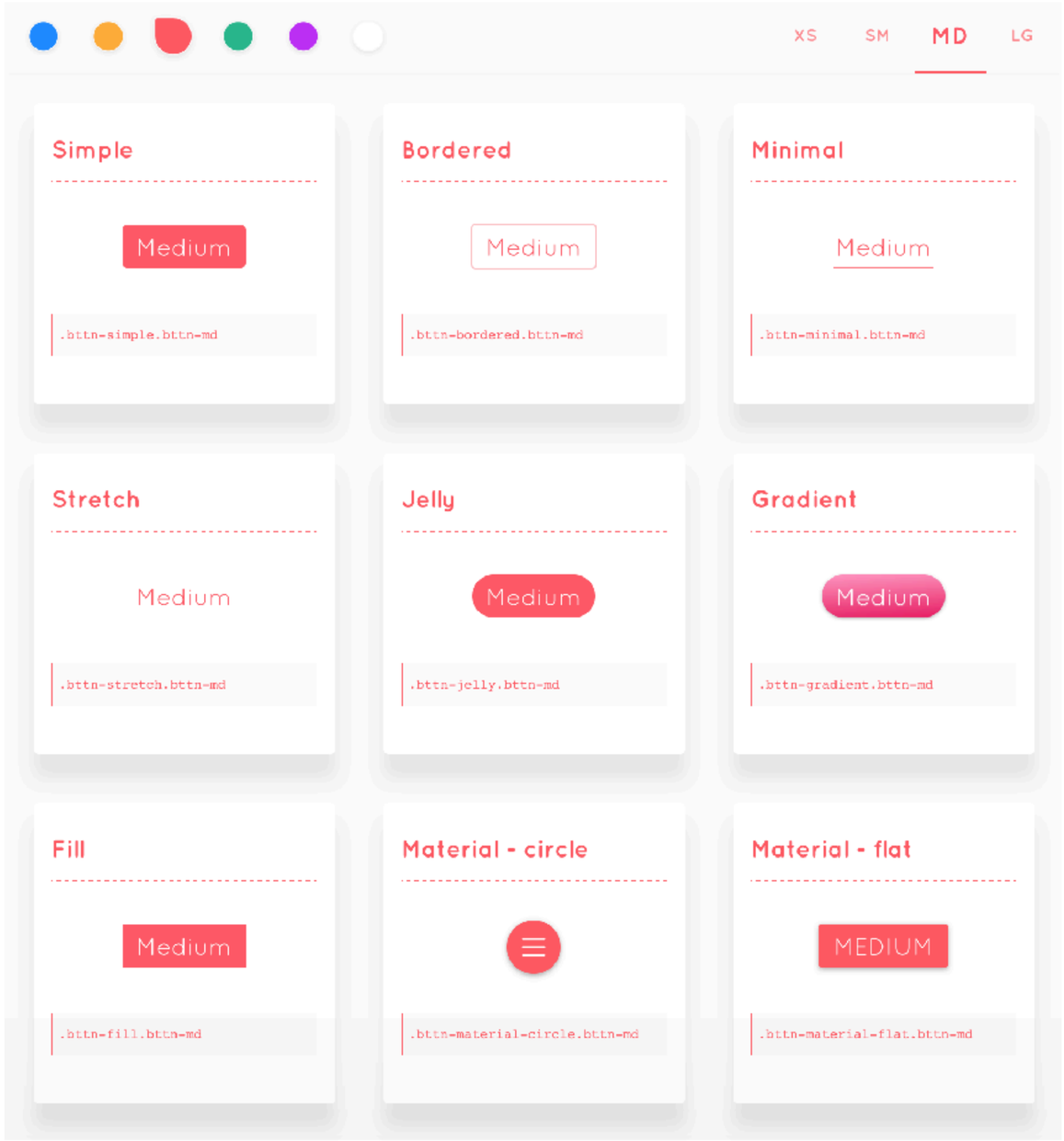
CSS

CSS

CSS is a **selector** markup language that allows us to specify how to present (render) the document info stored in the HTML.

With CSS, we can control all aspects of the visualization and some other features:

- **Colors**: content, background, borders
- **Margins**: interior and exterior margins
- **Position**: where to put it
- **Sizes**: width, height
- **Behavior**: changes on mouse over



CSS: Example

```
* {
        color: blue; /* or #0000FF or rgba(0, 0, 255, 1.0) */
        margin: 10px; // distance from border to outer elements
        font: 14px Tahoma;
}
```

This will change all the tags in my web (‘*’ means *all*) to look blue with font Tahoma with 14px, and leaving a margin of 10px around.

CSS: More Fields

Other common CSS fields:

- `background-color: red;`
- `background-image: url('file.png');`
- `border: 2px solid black;`
- `border-top: 2px solid red;`
- `border-radius: 2px; //to make corners more round`
- `padding: 2px; //distance from the border to the inner elements`
- `width: 100%;  300px;  1.3em;  //many different ways to specify distances`
- `height: 200px;`
- `text-align: center;`
- `box-shadow: 3px 3px 5px black;`
- `cursor: pointer;`
- `display: inline-block;`
- `overflow: hidden;`

CSS: Adding Styles to your Website

There are four ways to add CSS rules to your HTML page:

- Inserting the code inside a style tag

```
<style>
    p { color: blue }
</style>
```

- Referencing an external CSS file

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

- Using the `style` attribute on a tag

```
<p style="color: blue; margin: 10px">
```

- Using Javascript (we will see this one later).

CSS: Affecting Select Tags

Let's start by changing the background color of one tag of our website:

```
div {
        background-color: red;
}
```

Meaning: every HTML `div` tag found in our website will have a red background color. Remember that `div`s are used mostly to represent areas of our website.

We could also change the whole website background by affecting the `body` tag:

```
body {
        background-color: red;
}
```

CSS: Affecting Select Tags

What if we want to change one specific tag & not *all* tags of the same type?

We can specify more precise selectors – using `class` or `id`, for instance. To specify a tag with a given **class** name, we use the **dot**:

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

This will affect only the tags `p` with class name `intro`. i.e.:

```
<p class="intro">
```

CSS Selectors

There are several selectors we can use to narrow our rules to very specific tags of our website. Common ones:

- **tag name**: just the name of the tag
  - `p { ... }  //affects all <p> tags`
- **dot** (.)**:** affects to tags with that class
  - `p.stress { ... } //affects all <p> tags w/ class="stress"`
- **sharp character** (#): specifies tags with that id
  - `p#intro { ... } //affects to the <p> tag with the id="intro"`
- **colon** (:): behavior states (mouse on top)
  - `p:hover { ... } //affects to <p> tags upon mouseover`
- **brackets** ([attr='value']): tags with the attribute attr with the value 'value'
  - `input[type="text"] {...} // affects input tags of type=text`

CSS Selectors

You can also select tags that are inside of other tags. Just separate the selectors by an space:

```
div#main p.intro { ... }
```

This will affect *only* the `p` tags of class `intro` that are inside the tag `div` of id `main`

```
<div id="main">
    <p class="intro">....</p>   ← Affects this one
</div>
```

.... ← **but not this one**

CSS Selectors

And you can combine selectors to narrow it down more.

```
div#main.intro:hover { ... }
```

will apply the CSS to the any tag `div` with id `main` and class `intro` if the mouse is over the tagged element(s).

Conversely, you can use the class or id selectors without tags at all. The below means that the styling will affect *all* nodes of id `main` :

```
#main { ... }
```

CSS Selectors

If you want to select only elements that are direct children of one element (not any one that has an ancestor with that rule), use the `>` character:

```
ul.menu > li { ... }
```

Finally, if you want to use the same CSS actions on several selectors, use the comma (`,`) character:

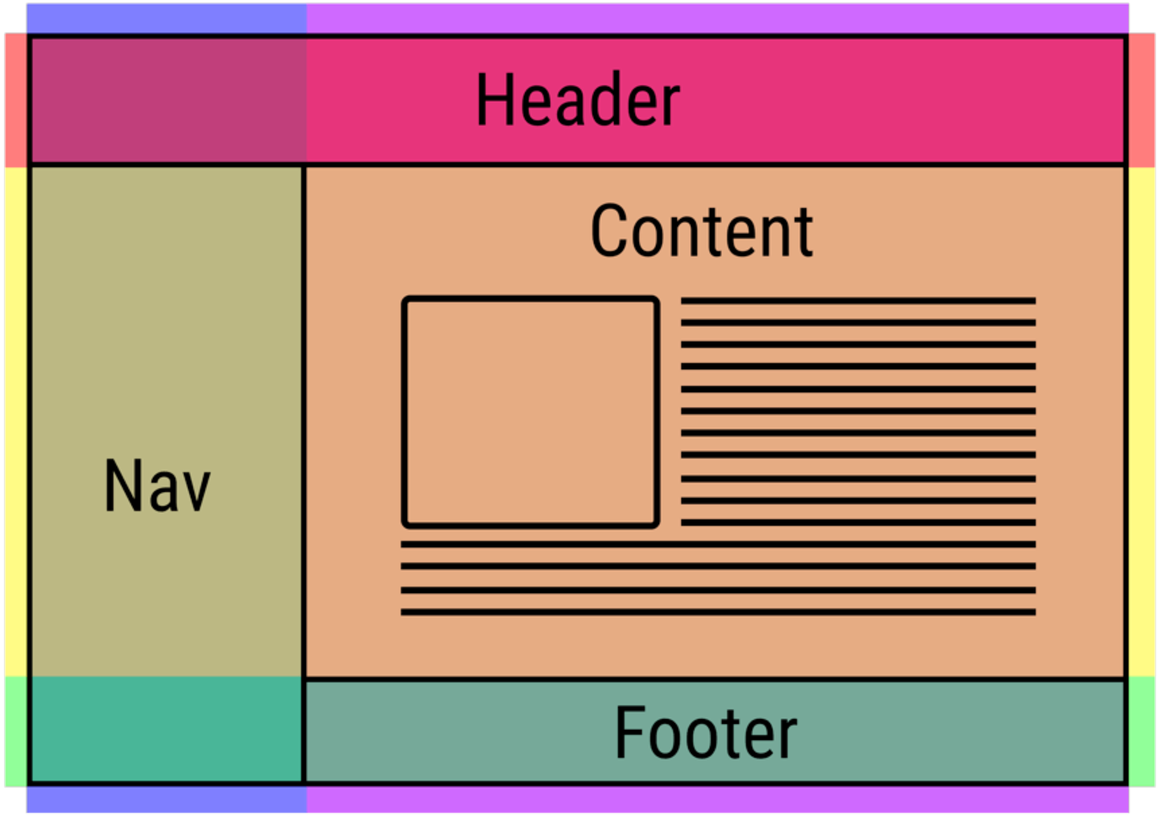
`div, p { … }` ← this will apply to all `div`s and `p` tags

CSS: Layout

One of the hardest and most important parts of CSS is constructing the layout of your website.

By default, HTML tends to put everything in one column, which is not ideal.

There are many ways in CSS to address this issue (tables, fixed divs, flex, grid, ...).



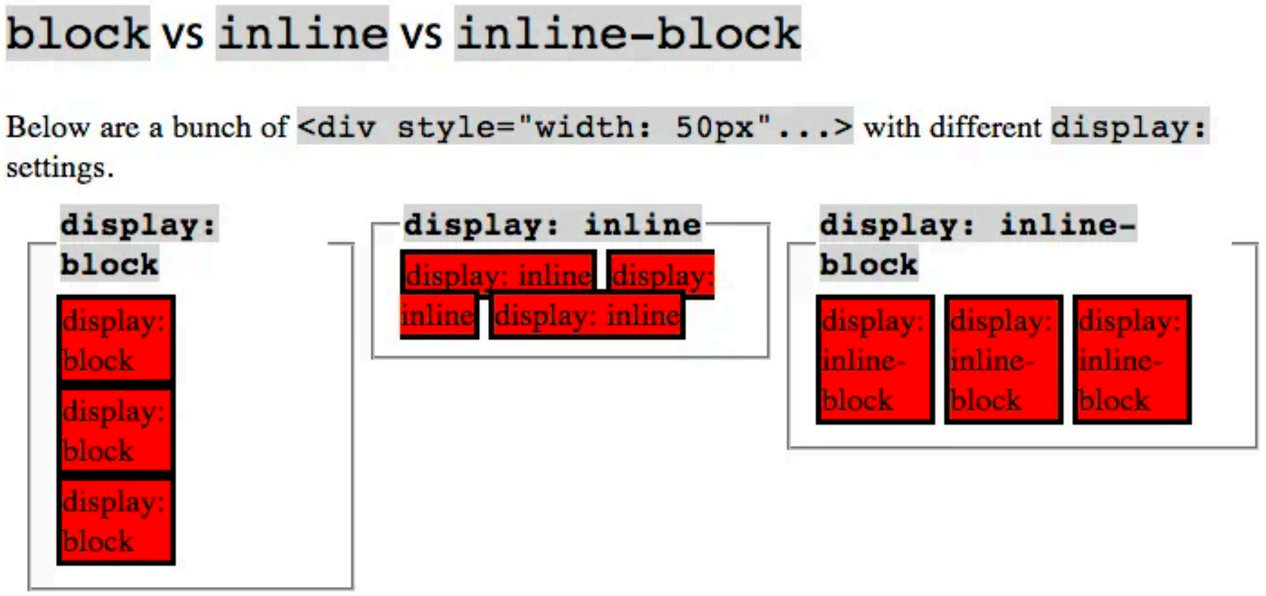
Arranging HTML Elements with CSS

This tutorial explains the different ways elements can be arranged.

You can change the way elements are arranged using the `display` property:

```
div { display: inline-block;
}
```

Also check out the property `float`.



Source: Erica Renée Faulkenberry

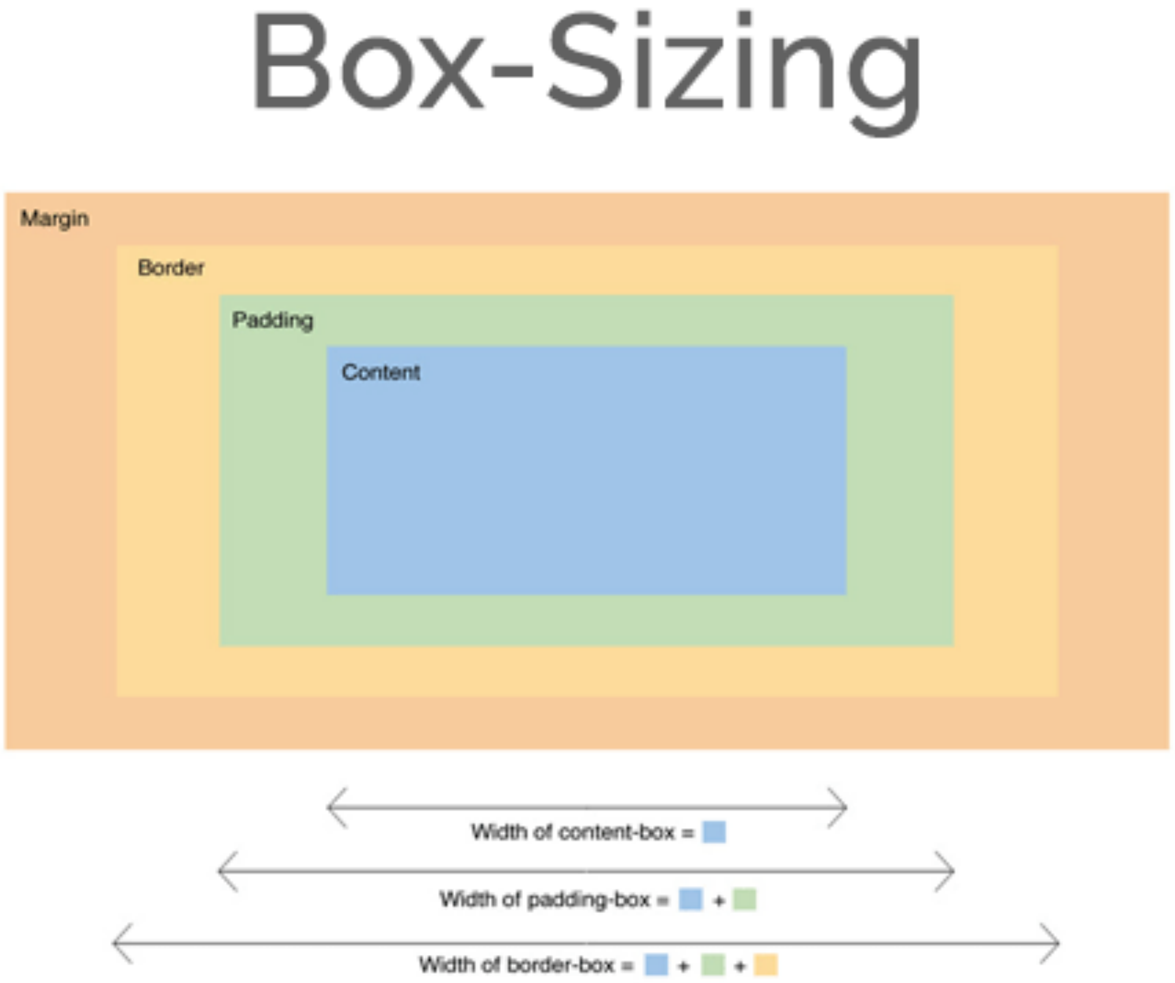
A Note on Sizing Elements

By default, `margins` are added to any `width` and `height` specified for an element. So a `div` with `width` 100px and `margin` 10px will measure 120px on the screen, not 100px.

This could break your layout.

You can change this by changing the box model of the element so the width uses the outermost border:

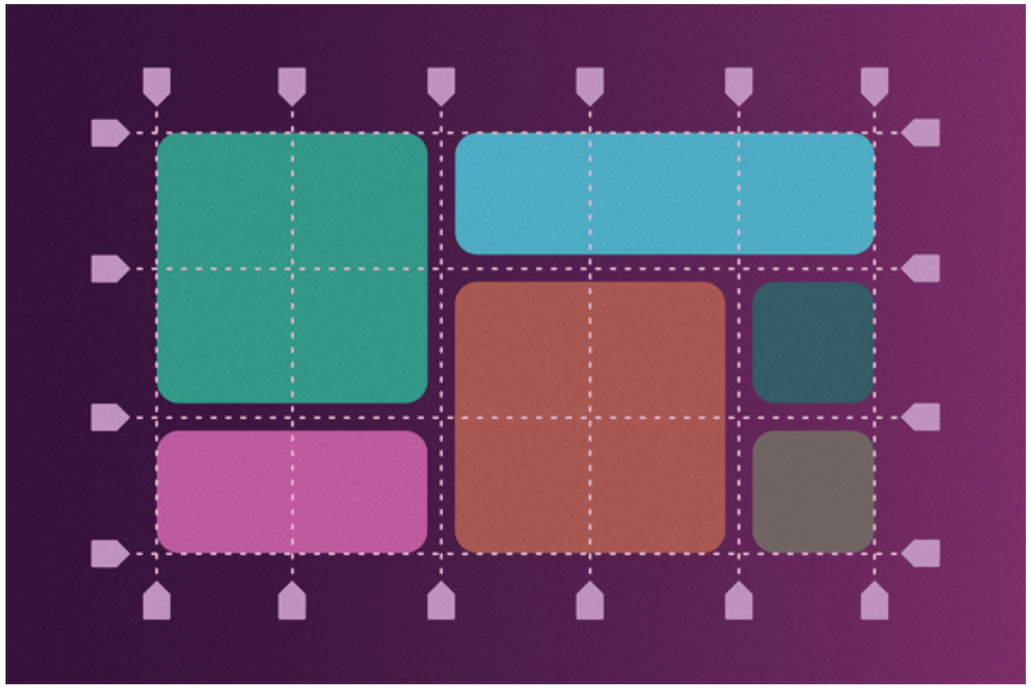
```
div { box-sizing: border; }
```



CSS: Grids

One possible layout method is to use the CSS Grid system.

Tutorial



HTML

```
<div class="grid-container">
  <div class="grid-item1">1</div>
  <div class="grid-item2">2</div>
</div>
```

CSS

```
.grid-container {
  display: grid;
  grid-template-rows: 100px; 100px;
  grid-template-columns: 100px; 100px;
100px;
  grid-gap: 5px;
}

.grid-item1 {
  background: blue;
  border: black 5px solid;
  grid-column-start: 1;
  grid-column-end: 5;
  grid-row-start: 1;
  grid-row-end: 3;
}
```

CSS: Fullscreen divs

We often want to have a `div` that covers the whole screen.

In that case, remember to use percentages to define the size of elements, but keep in mind that percentages are relative to the element's parent size, so you must set the `width` & `height` of the `<body>` element to 100%.

CSS

```
html, body {
        width: 100%;
        height: 100%;
}

div {
        margin: 0;
        padding: 0;
}

#main {
        width: 100%;
        height: 100%;
}
```

CSS: Note about mobile

Desktop layout

Mobile layout

We can use viewport screen size selectors to change the CSS for mobile versus desktop displays.

Include a meta tag in HTML to specify the viewport as device-width

Use @media (min-width: XXXpx) in CSS to add breakpoints at certain screen sizes

HTML

```
<meta name="viewport"
content="width=device-width, initial-
scale=1.0">
```

CSS

```
@media (min-width: 600px) {
  .header {grid-area: 1 / span 6;}
  .menu {grid-area: 2 / span 1;}
  .content {grid-area: 2 / span 4;}
  .facts {grid-area: 3 / span 6;}
  .footer {grid-area: 4 / span 6;}
}
```

https://www.w3schools.com/css/tryit.asp?filename=tryresponsive_breakpoints2

CSS: Note on Centering

Centering divs can be hard sometimes. Try this trick:

```
.horizontal-and-vertical-centering {
  display: flex;
  justify-content: center;
  align-items: center;
}
```

Javascript

JavaScript: Intro

- JS is a regular **programming language**. Easy to start, hard to master!
- Allows us to give some interactivity to elements on the web.
- Syntax is similar to C/C++ or Java.
- You can change the content of the HTML or the CSS applied to an element.
- You can even send or retrieve information to update the content of the web without reloading the page.
- **HTML is the webpage elements, CSS is the stylistic elements, JS is the interactive elements**

JavaScript: Syntax

Very similar to C++ or Java but much simpler.

```
var my_number = 10; //this is a comment
var my_string = "hello";
var my_array = [10,20,"name",true];
var my_object = { name: "javi", city: "Barcelona" };

function say( str )
{
        for(var i = 0; i < 10; ++i)
                console.log(" say: " + str );
}
```

Javascript Syntax

- +, -, *, /, **, % work similar to Python
- **console.log(“str”) is how to print in Javascript**
- ++, - - work similar to C++ and Java
- <, >, ==, !=, <=, >=, also work similar to Python
- Adding in === and !== to check equality and type
- Every statement ends with a semicolon
- Declaration occurs with let, var, and const
- If you use const you can’t change the variable later on
    - Control statements i.e. if, if-else, else-if chain very very similar to C++
- Lots of interesting array characteristics, be careful of “holes” when removing and inserting elements.

Javascript Data Types

1. String - “Hello”
2. Number - 0, 1, 3.14, 2.7182818
3. Boolean - True or False
4. Undefined (a variable that has been declared, but no value has been declared, but no value has been assigned to it yet)
5. Null (a variable with no value - it may have had one at some point, but no longer has a value)
6. Object (the only non-primitive data type)

There are 2 more secret primitives, Symbol and BigInt, but you probably won’t use them.

Javascript Objects

Very much like classes in Java (without explicit types). Can contain strings, numbers, arrays, references to other objects. Example below:

```
var Hotel =
{
    name: ‘Hilton’,

    rooms: 40,

    booked: 25,

    features: [‘microwave’, ‘bed’, ‘nightstand’, ‘couch’]
}
```

JavaScript: Adding It

There are three ways to execute JavaScript code on a website:

- **Embed** the code in the HTML using the `<script>` tag:

```
<script> /* some code */ </script>
```

- **Import** a JS file using the `<script>` tag:

```
<script src="file.js" />
```

- **Inject** the code on an event inside a tag:

```
<button onclick="javascript: /*code*/">press me</button>
```

JavaScript: Example

```
<html>

    <body>

      <h1>This is a title</h1>

      <script>

         var title = document.querySelector("h1");

         title.innerHTML = "This is another title";

      </script>

    </body>

</html>
```

JavaScript API

Javascript comes with a rich API to do many things like:

- Access the DOM (HTML nodes)
- Make HTTP Requests
- Play videos and sounds
- Detect user actions (mouse move, key press)
- Launch threads
- Access the GPU, get a webcam image, etc....

And the API keeps growing with every new update of the standard.

Check the Web API reference to know more

JavaScript: Retrieving Elements

You can get elements from the DOM (HTML tree) using different approaches.

- **Crawling the HTML tree** (starting from the body, and traversing its children)
- **Using a selector** (like in CSS)
- **Attaching events listeners** (calling functions when some actions are performed)

JavaScript: Crawling the DOM

Different variables give access to website information:

- `document`: the DOM information (HTML)
- `window`: the browser window

The `document` variable allows us to crawl the tree:

```
document.body.children[0] // returns the first node inside body tag
```

JavaScript: Using Selectors

You can also retrieve elements using selectors:

```
var nodes = document.querySelectorAll("p.intro");
```

will return an array with all `<p class="intro">` nodes in the document.

Or if we have already a node (e.g., `mynode`) and we want to search inside:

```
var node = mynode.querySelectorAll("p.intro")
```

JavaScript: Modify Nodes

With JS you can change the attributes of an existing node:

```
mynode.id = "intro"; //sets an id
mynode.className = "important"; //set class
mynode.classList.add("good"); //add “good” to the current classes
```

Change the content :

```
mynode.innerHTML = "<p>text to show</p>"; //change content
```

Modify the style (CSS):

```
mynode.style.color = "red"; //change any css properties
```

or add to the behavior of a node:

```
mynode.addEventListener("click", function(e) {
        //do something
});
```

JavaScript: Create Nodes

Create elements:

```
var element = document.createElement("div");
```

And attach them to the DOM:

```
document.querySelector("#main").appendChild( element );
```

Or remove it from its parent:

```
var element = document.querySelector("foo");
element.parentNode.removeChild( element );
```

You can clone an element also easily:

```
var cloned = element.cloneNode(true);
```

JavaScript: Hiding and Showing Elements

Sometimes it may be useful to hide one element or show another.

You can change an element CSS directly by accessing its `style` property.

To avoid being displayed on the web, change `display` to "none"

```
element.style.display = "none"; //hides element from being
rendered
element.style.display = ""; //displays it again
```

JavaScript: Inputs

If you want the user to be able to input some text, use the `<input>` tag:

```
<input type="text"/>
```

There are other inputs. Check out this list.

With JavaScript, we can attach events like "click" or "keydown".

To read or modify the content of the input: `my_input_element.value`

```
my_input_element.value = ""; //clear the text in the input
element
```

Be Careful about Execution Flow

Scripts are executed when the HTML is being parsed.

Be careful accessing the DOM, as the DOM won’t contain all elements until all the HTML is parsed.

It is good practice to start your code with an init function called at **the end of your** HTML.

Sample index.html structure

In a folder, index.html is the default homepage of the directory

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Example Webpage</title>
    <style type="text/css"> /* CSS here */ </style>
</head>
<body>
    <h1>Hello, World!</h1>
    <p>This is an example webpage using HTML, CSS, and
JavaScript.</p>
   <script> /* JS code here */ </script>
</body>
</html>
```

CSS goes at the end of <head>

JS goes at the end of <body>

Putting it Together

HTML in index.html

```
<link href="style.css" rel="stylesheet"/>
<h1>Welcome</h1>
<p>
     <button>Click me</button>
</p>
<script src="code.js"/>
```

CSS in style.css

```
h1 { color: #333; }
button {
     border: 2px solid #AAA;
     background-color: #555;
}
```

Javascript in code.js

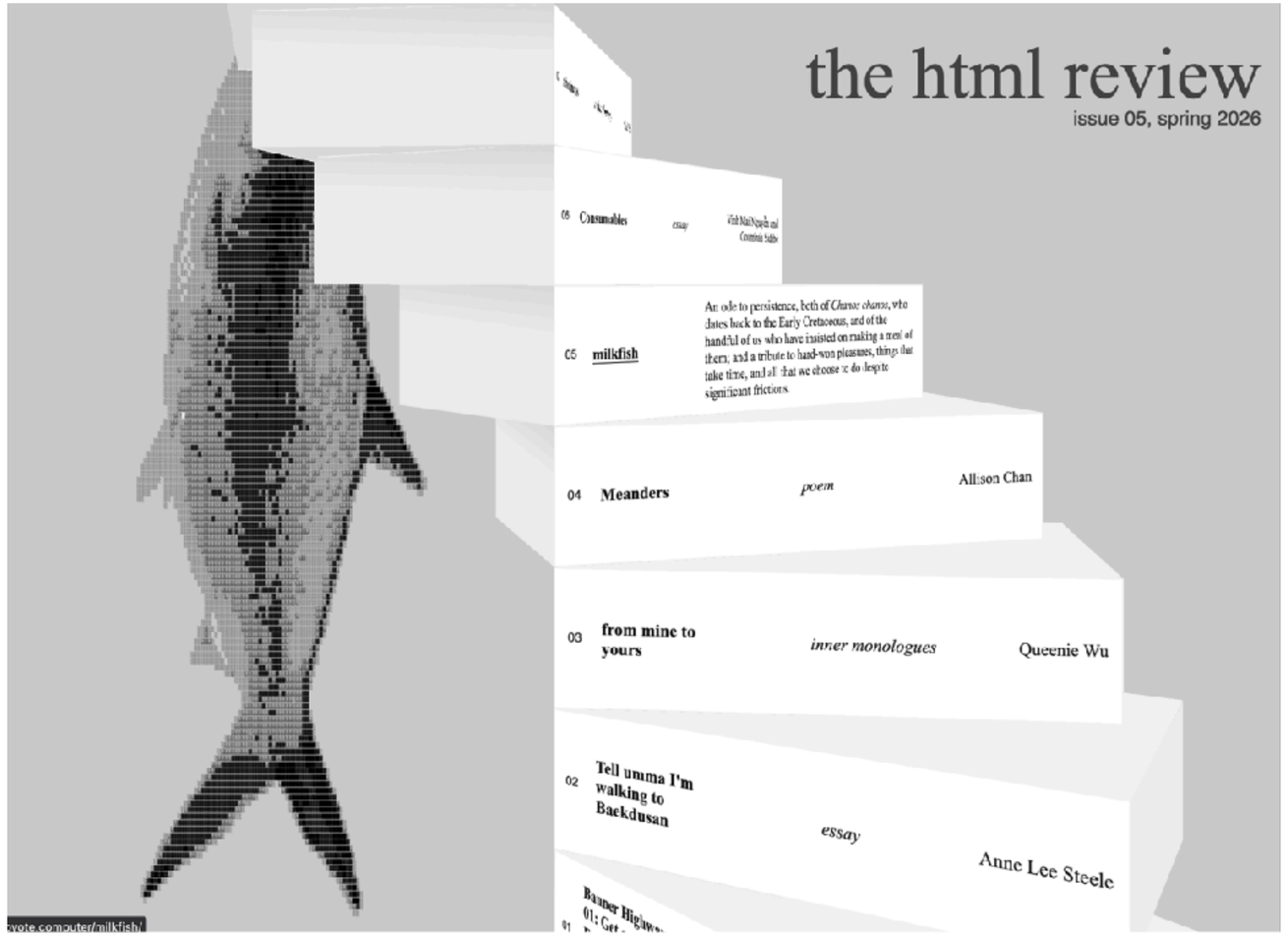
```
//fetch the button from the DOM
var button =
document.querySelector("button");

//attach and event when the user clicks it
button.addEventListener("click", myfunction);

//create the function that will be called
when the button is pressed
function myfunction()
{
          //this shows a popup window
          alert("button clicked!");
}
```

Activity: Browse & share the HTML Review

- The HTML review is a web-based media publication that tries to push the limits of what HTML can do
- Click through some of the publications to get inspired for PM5! Find a cool one to share with a neighbor and why you like it, and how you think they implemented it - inspect source!

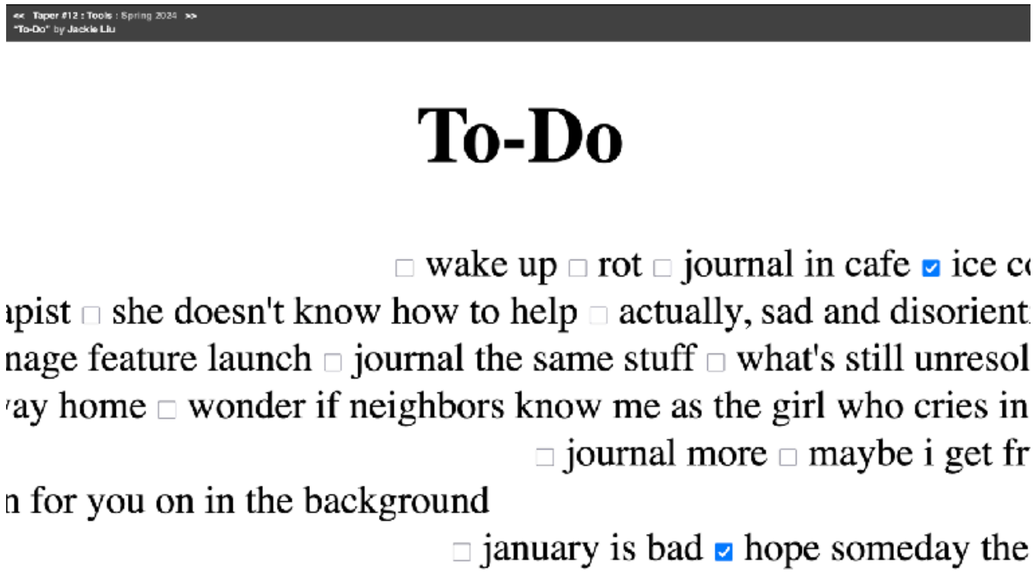


https://thehtml.review/05/

PM5: A Webpage of One’s Own

PM5 logic

- There is so much rich possibility with just HTML, CSS, and JS - no need to use fancy external libraries, or even embed external media like images/audio
- Can you carve out a small (< 64KB) space for yourself on the internet with a single .html file?
- Due in two weeks (Mon Apr 13) - focus on your Figma first!



Inspiration: Taper mag: https://taper.badquar.to/

Class 17 recap

- TODOs
  - **PLEASE** be making progress on your WoZ prototype! In class eval Weds 4/8. It takes longer than you think! (I will give feedback on your video prototypes by tomorrow.)
  - Weds 4/1: RRs, seminar from Claudio & River
  - Mon 4/6: RRs, seminars from Cris & Ivyer; Jack G & Francisco
  - PM5 (A webpage of one's own) due in **2 weeks,** assignment write up will be up by tomorrow
- Resources
  - Write up from last semester: https://cs.pomona.edu/classes/cs181dt/visual-software-design/
  - More external ones after this slide
  - Thanks to Catherine Song for some slides

More resources

- **HTML**
  - HTML Reference: a description of all HTML tags.
  - The 25 Most used tags: a list with information of the more common tags.
  - HTML5 Good practices: some tips for starters.

- **CSS**
  - One line layouts tutorials
  - Understanding the Box Model: a good explanation of how to position the information on your document.
  - All CSS Selectors: the CSS selectors specification page.
  - CSS Transition: how to make animations just using CSS
  - TailwindCSS: a CSS Framework