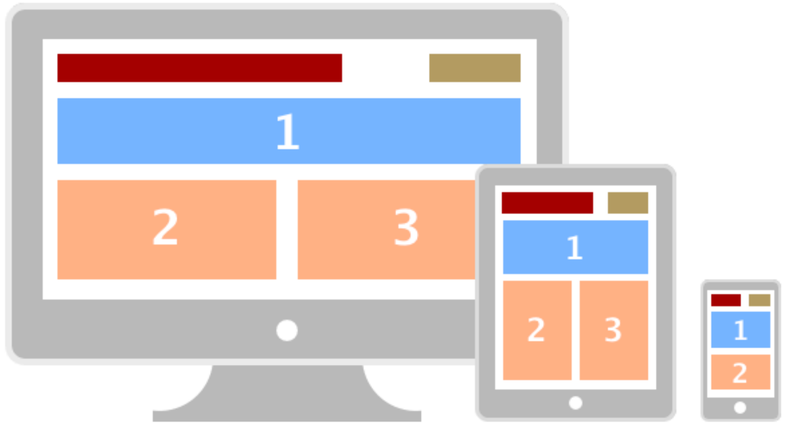
IN4MATX 133: User Interface Software

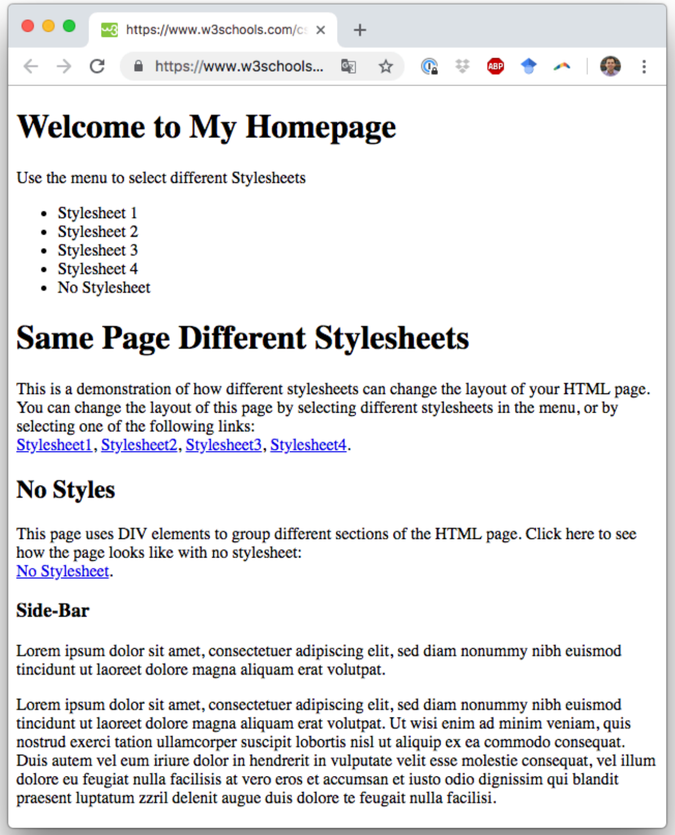
Lecture 3:
CSS & Responsive Design

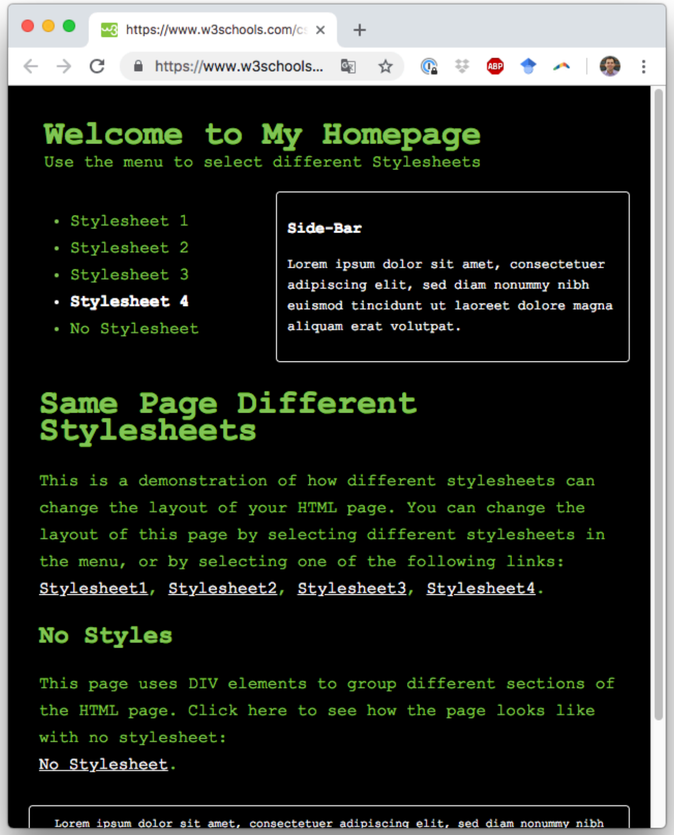
Professor Daniel A. Epstein
TA Lucas de Melo Silva
TA Jong Ho Lee

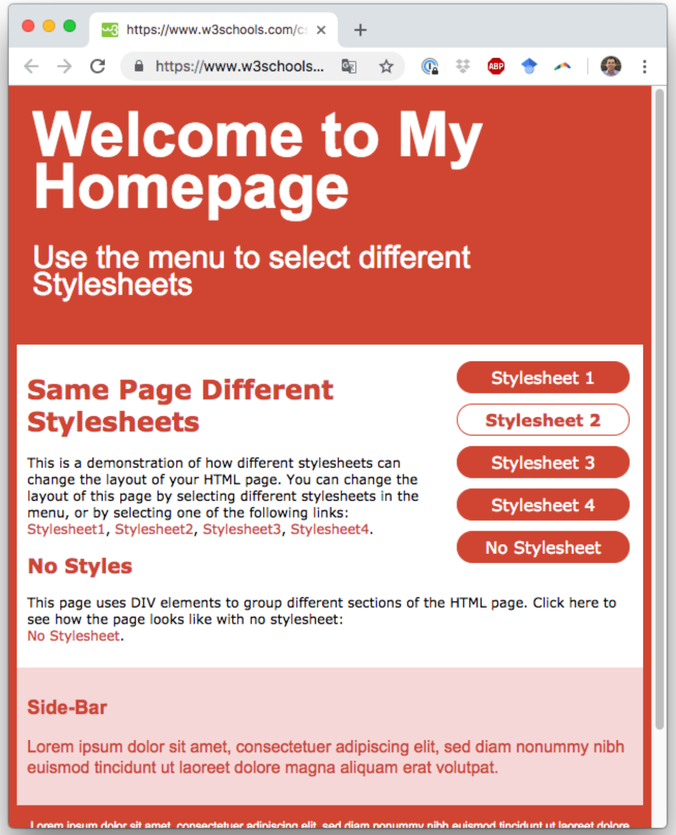
Today: CSS and responsive design



Same page, different stylesheets







https://www.w3schools.com/css/demo_default.htm

Today’s goals

By the end of today, you should be able to…

- Explain the goals of CSS and why it exists as separate from HTML
- Describe the CSS hierarchy and fallback structure
- Utilize the box model and positioning options to arrange content
- Style nested tags with child, adjacent sibling, and general sibling selectors
- Describe how responsive and adaptive design differ
  and when you might prefer one or the other
- Explain the advantages and disadvantages of a mobile-first design

CSS

Cascading Style Sheets

- Defines rules for styling
- Differs from HTML, which provides structure for the document

CSS: but why?

- Reusability
  - Apply the same style to multiple web pages
- Modularity
  - Include multiple stylesheets that apply to a single page
- Sane management
  - Files can be version controlled, separate from HTML structural content
- Maintainability
  - Easier to find a page's style

Ok, so how do I write CSS?

CSS syntax

- **Selectors** specify which elements a **rule** applies to

- **Rules** specify what *values* to assign to different formatting **properties**

```
/* CSS Pseudocode */
selector {
  property: value;
  property: value;
  ...
}
```

One rule, many properties

CSS syntax

```
h1 {
  font-family: 'Arial';
  color: blue;
  background-color: #ff0000; /*red*/
}
```

Apply to all h1 tags

"font"

- Link to stylesheets in HTML's **<head>**

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

relation between
this page and reference

no content,
so no closing tag

Element, ID, and Class selectors

- element: what tag is being styled
- class: a type of element
- id: one specific element

```
p {
  font-family: 'Arial';
  color: red;
}
.emphasize {
  font-family: 'Arial';
  color: red;
}
#redtext {
  font-family: 'Arial';
  color: red;
}
```

HTML Class and ID attributes

```
<div class="widget foo" id="baz"></div>
```

- Variable-value just like any other attribute (`href`, `src`)
- An element can have many classes, only one ID
- Each page can have only one element with a given ID
  - Required to pass validation
- Can use the same class on multiple elements
  - And should; it's useful to apply the same style to many elements

https://css-tricks.com/the-difference-between-id-and-class/

HTML Class and ID attributes

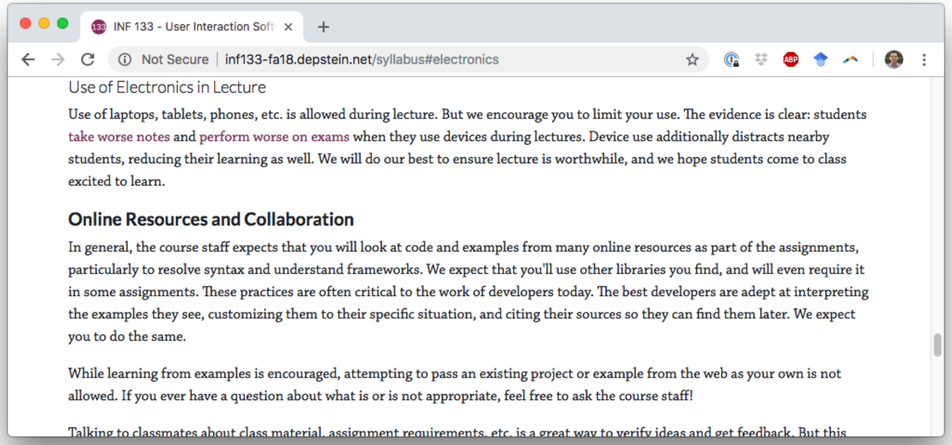
```
<div class="widget foo" id="baz"></div>
```

```
div.widget.foo#baz {
  /*can chain selectors together!*/
}
```

https://css-tricks.com/the-difference-between-id-and-class/

HTML Class and ID attributes

- Fun trick: IDs can be used for navigation
- `http://example.com/#id`



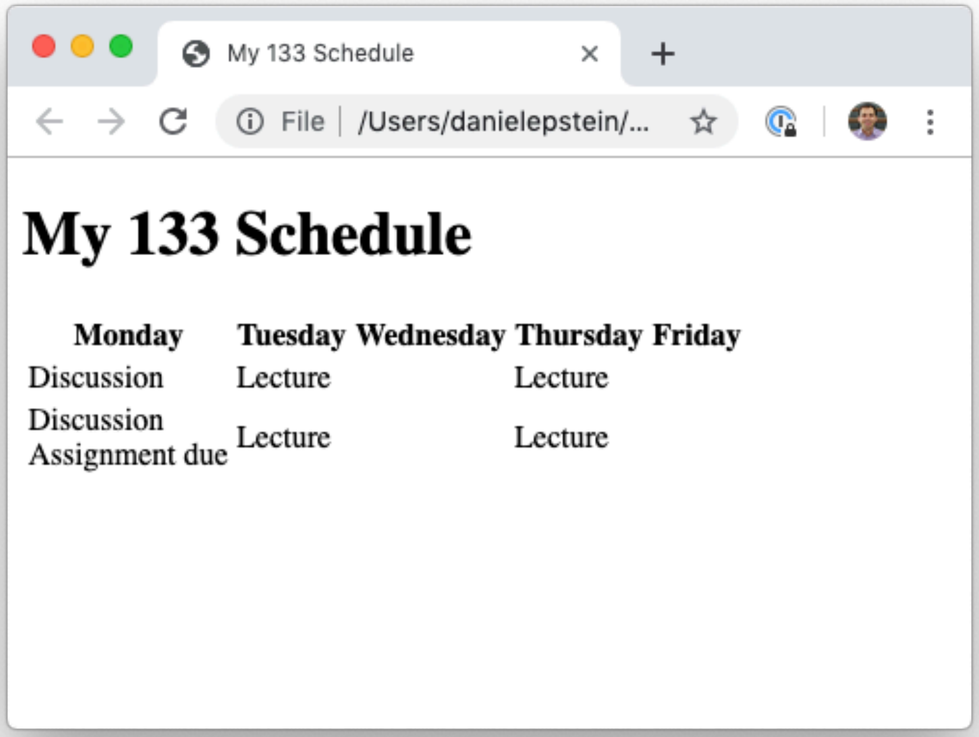
https://css-tricks.com/the-difference-between-id-and-class/

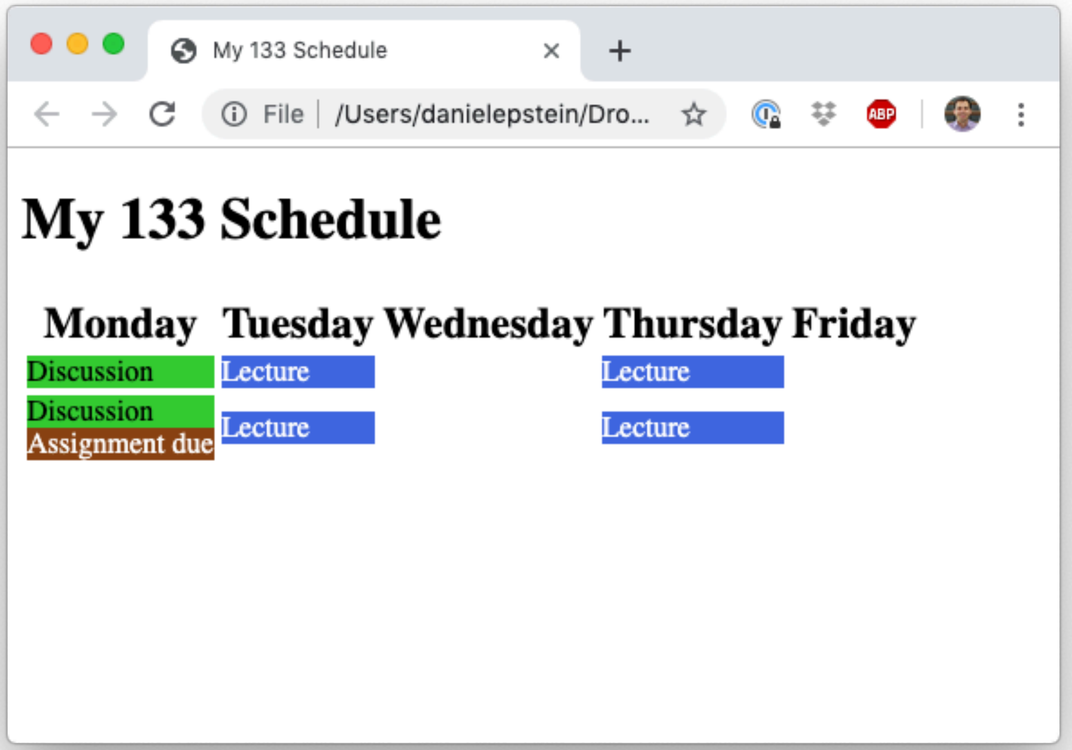
CSS properties

- `font-family`: the “font” (fallback alternatives separated by commas)
- `font-size`: the size of the text
- `font-weight`: boldness
- `color`: text color
- `background-color` (element’s background)
- `opacity` (transparency)
- And much, much more!

http://www.w3schools.com/cssref/default.asp

Let’s style a class schedule





HTML vs. CSS

- HTML specifies the *semantics*
- CSS specifies the *appearance*

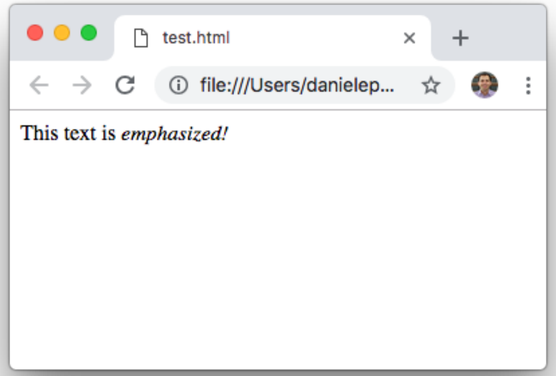
HTML vs. CSS

```
<!--HTML-->
<p> This text is
<i>emphasized!</i></p>
```

Conflates appearance
and semantics

```
<!--HTML-->
<p> This text is
<em>emphasized!</em></p>
```

Says nothing
about appearance

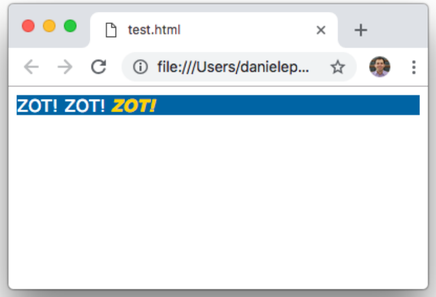


Question

Which CSS renders

```
<p class="para">
ZOT!
ZOT!
<em id="zot">ZOT!</em>
</p>
```

as



?

A

```
p {
    font-family: 'Verdana';
    color: #0064a4; /*Blue*/
    background-color: white;
}
#em {
    color: #ffd200; /*Yellow*/

    font-weight: 700; /*Bold*/
}
```

B

```
.para {
    font-family: 'Verdana';
    background-color: #0064a4; /*Blue*/
    color: white;
}
zot {
    color: #ffd200; /*Yellow*/
    font-style: italic;
    font-weight: 700; /*Bold*/
}
```

C

```
.para {
    font-family: 'Verdana';
    background-color: #0064a4; /*Blue*/
    color: white;
}
em {
    color: #ffd200; /*Yellow*/

    font-weight: 700; /*Bold*/
}
```

D

```
#para {
    font-family: 'Verdana';
    background-color: #0064a4; /*Blue*/
    color: white;
}
.em {
    color: #ffd200; /*Yellow*/
    font-style: italic;
    font-weight: 700; /*Bold*/
}
```

E

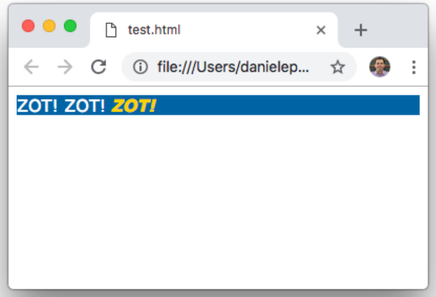
```
.para {
    font-family: 'Verdana';
    background-color: #0064a4; /*Blue*/
    color: #ffd200; /*Yellow*/
}
#zot {
    color: white;
    font-style: italic;
    font-weight: 700; /*Bold*/
}
```

Question

Which CSS renders

```
<p class="para">
ZOT!
ZOT!
<em id="zot">ZOT!</em>
</p>
```

as



?

A

```
p {
    font-family: 'Verdana';
    color: #0064a4; /*Blue*/
    background-color: white;
}
#em {
    color: #ffd200; /*Yellow*/

    font-weight: 700; /*Bold*/
}
```

B

```
.para {
    font-family: 'Verdana';
    background-color: #0064a4; /*Blue*/
    color: white;
}
zot {
    color: #ffd200; /*Yellow*/
    font-style: italic;
    font-weight: 700; /*Bold*/
}
```

C

```
.para {
    font-family: 'Verdana';
    background-color: #0064a4; /*Blue*/
    color: white;
}
em {
    color: #ffd200; /*Yellow*/

    font-weight: 700; /*Bold*/
}
```

D

```
#para {
    font-family: 'Verdana';
    background-color: #0064a4; /*Blue*/
    color: white;
}
.em {
    color: #ffd200; /*Yellow*/
    font-style: italic;
    font-weight: 700; /*Bold*/
}
```

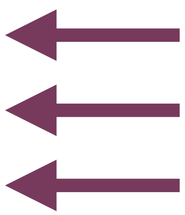
E

```
.para {
    font-family: 'Verdana';
    background-color: #0064a4; /*Blue*/
    color: #ffd200; /*Yellow*/
}
#zot {
    color: white;
    font-style: italic;
    font-weight: 700; /*Bold*/
}
```

Cascading Style Sheets

- Multiple rules can apply to the same element (in a “cascade”)

```
<!--HTML-->
<p class="big blue">
  This tag has two classes: "big" and "blue"
  (classes are separated by spaces)
</p>

/* CSS */
p { font-family: 'Verdana'; }
.big { font-size: larger; }
.blue { color: blue; }
```



Apply to all `<p>` tags

Apply to all with `class=“big”`

Apply to all with `class=“blue”`

Cascading Style Sheets

- CSS rules are also inherited from parent tags

```
<div class="content"> <!-- has own styling -->
  <div class="sub-div"> <!-- has own styling + .content styling -->
    <ol class="my-list"> <!-- own styling + .sub-div + .content -->
                         <!-- own style is ol AND .my-list rules-->

      <!-- li styling + .my-list + .sub-div + .content -->
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
    </ol>
  </div>
</div>
```

Cascading Style Sheets

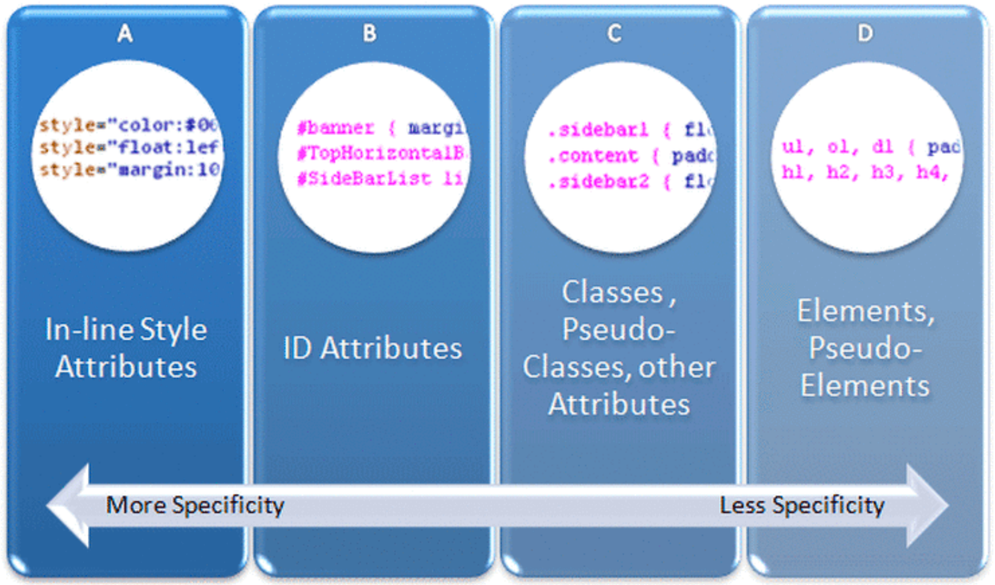
- Rules are applied **in order** (last rule always wins among peer selectors)

```
<!--HTML-->
<p class="red green">
  <em class="blue">Text is blue!</em>
</p>
```

```
/* CSS */
p { font-family: 'Verdana'; }
.red { color: red; }
.green { color: green; }
.blue { color: blue; }
```

Specifying styles

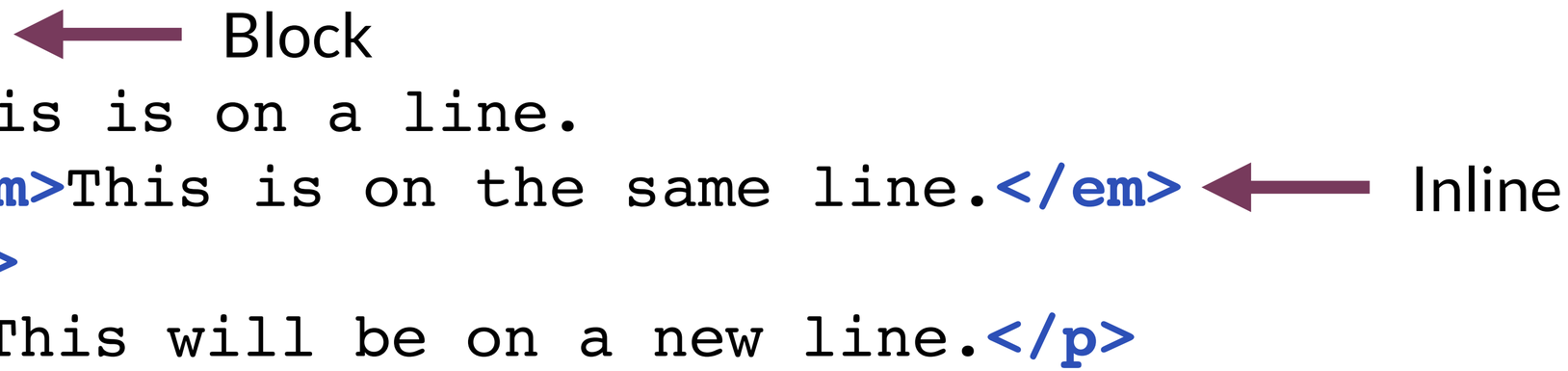
- CSS specificity is *calculated* based on which selector designates it
- General rule: rule that's "closer to the HTML element" applies
- This is difficult stuff, usually trial-and-error resolves most things



Our progress so far...

Positioning

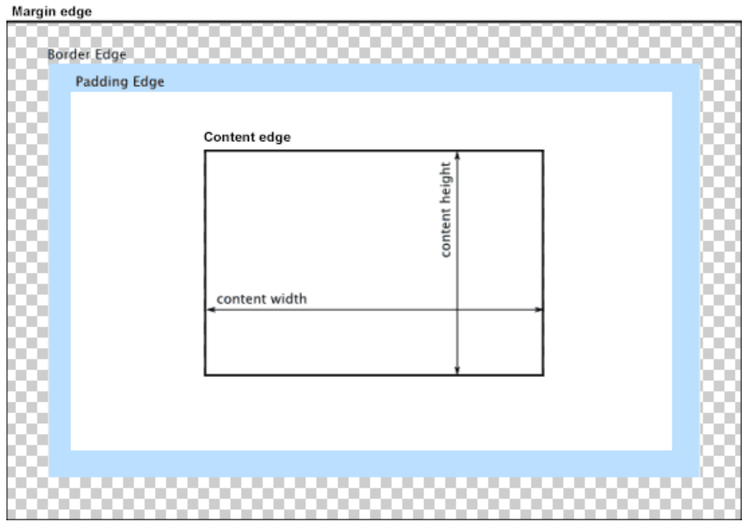
- HTML tags are either*:
  - **Block** elements (line break after them)
  - **Inline** elements (no line break)



- Don’t put block elements inside inline elements!

Positioning: box model

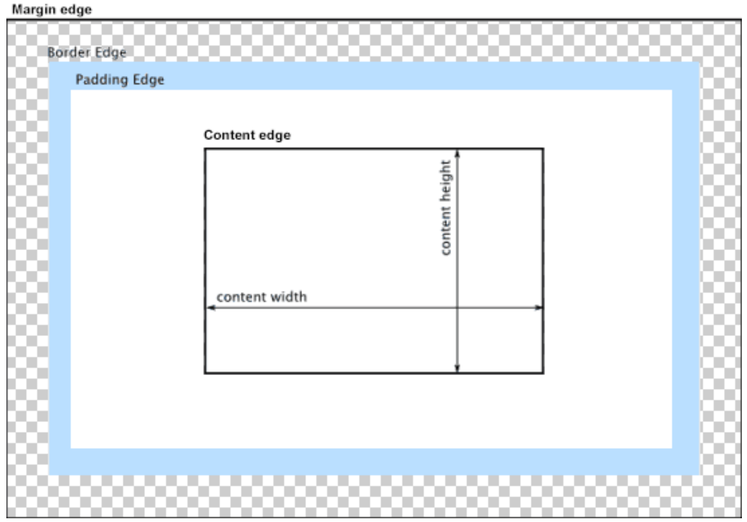
- Content contains “real” content
- Padding extends content area
- Border is similar
- Margin is intended to separate elements from neighbors



https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Box_Model/Introduction_to_the_CSS_box_model

Positioning: box model

- Content dimensions are specified with `width` and `height`
- `padding`, `border`, and `margin` have direction properties (e.g., `padding-top`, `margin-right`, `border-left`)
- `border` can have `border-color`, `border-width`, and `border-style`
- Content color (e.g., `background-color`) extends into padding

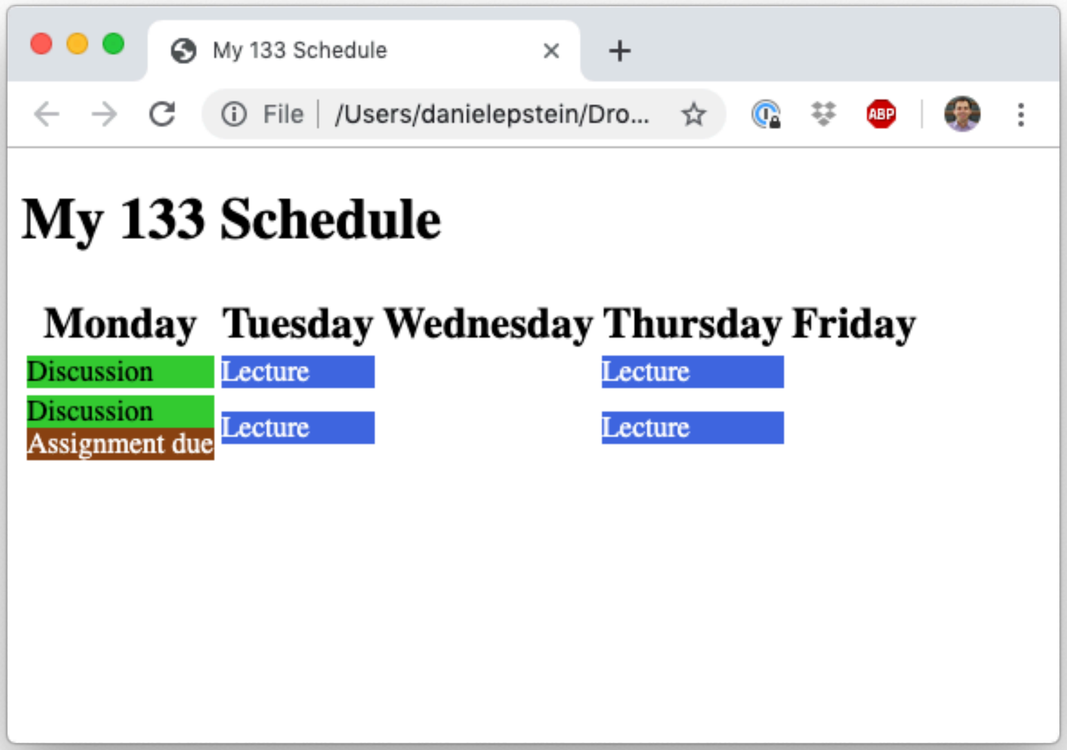


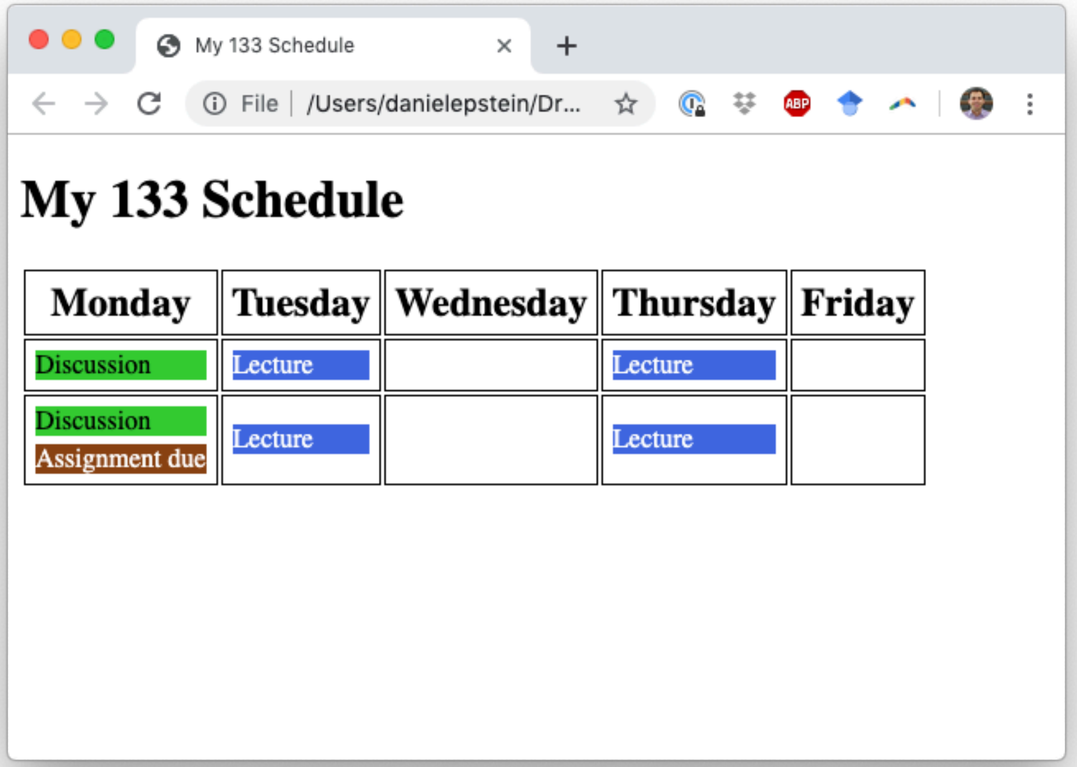
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Box_Model/Introduction_to_the_CSS_box_model

Positioning

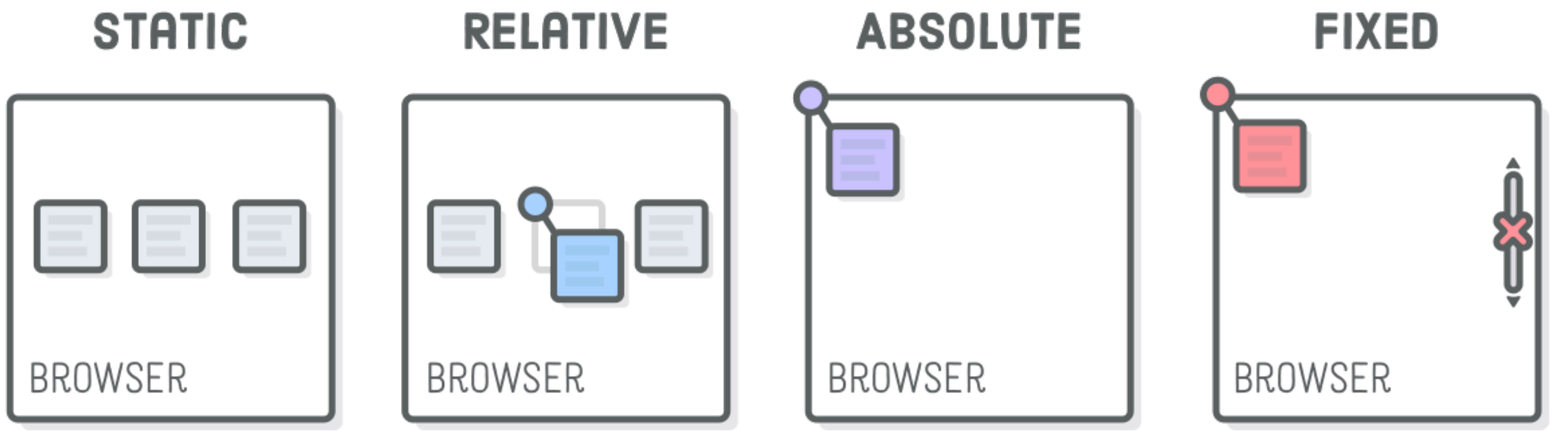
- All positioning is relative to the parent
  - If you nest tags, the child’s margins, etc. are all dependent on parent’s

Let’s style a class schedule





Positioning: types



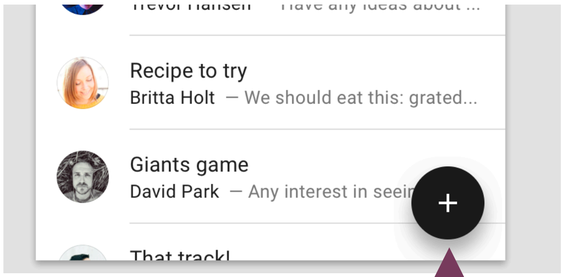
- `static` (default)
- `relative` (offset from default)
- `absolute` (from top-left)
- `fixed` (absolute + floating)

https://internetingishard.com/html-and-css/advanced-positioning/

Positioning: types

- `static` and `relative` follow the overall flow of a page
  - `relative` helps make adjustments to the flow
- `absolute` and `fixed` ignore it entirely
  - But they're helpful in some cases,
    like floating action buttons (FABs)

Relative position

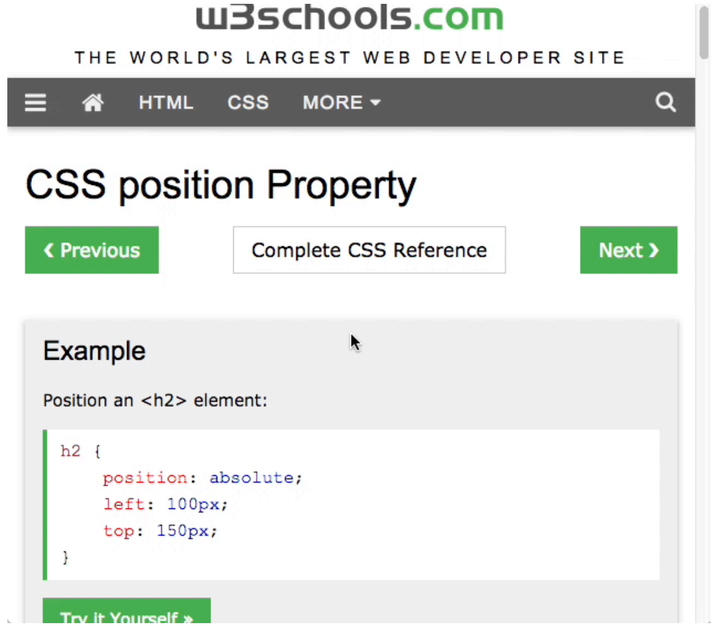


Absolute position

https://css-tricks.com/examples/AbsoluteInsideRelative/

Positioning: types

- `sticky` will stop when a user scrolls past it
  - Useful for menus
  - Not all browsers support it, but getting there



https://css-tricks.com/examples/AbsoluteInsideRelative/

Units

- Pixels (`px`), element units (`em`), percentages (`%`), real-world units (`in`, `cm`)
- Use relative units (`em`, `%`) whenever possible
- Helps accessibility, people with low vision change default size (usually 16px)
  - Em fonts scale from the default, a 30px heading stays 30px
- Also useful to vary based on screen size
  - More on how to do that next lecture

	Recommended	Occasional use	Not recommended
Screen	em, px, %	ex	pt, cm, mm, in, pc
Print	em, cm, mm, in, pt, pc, %	px, ex	

https://engageinteractive.co.uk/blog/em-vs-rem-vs-px

Advanced selectors

- Extremely useful for making clean stylesheets

- Add a top margin for all `h2`s that follow a `p`aragraph

```
p + h2 {
  margin-top: 10px;
}
```

- Or only in a particular `div`

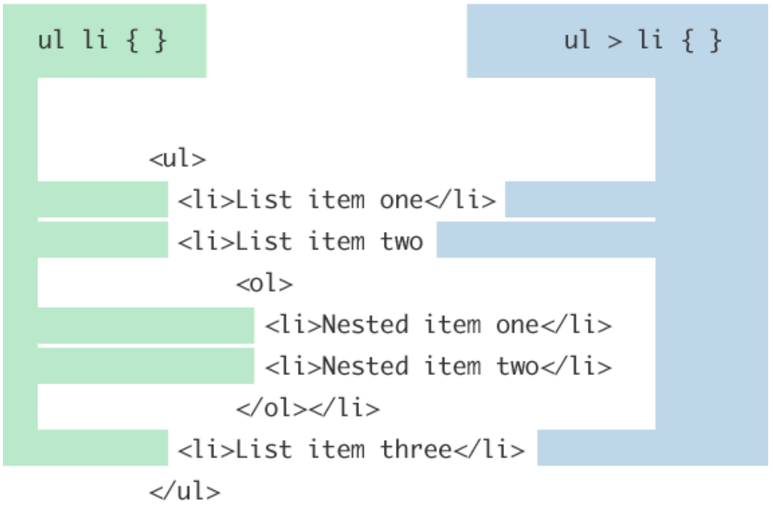
```
div.post p + h2 {
  margin-top: 10px;
}
```

https://www.smashingmagazine.com/2009/08/taming-advanced-css-selectors/

Advanced selectors

Child and Descendant Selectors

- `ui li`
  - Select *all* children and grandchildren
- `ui > li`
  - Select *direct* children (not grandchildren)

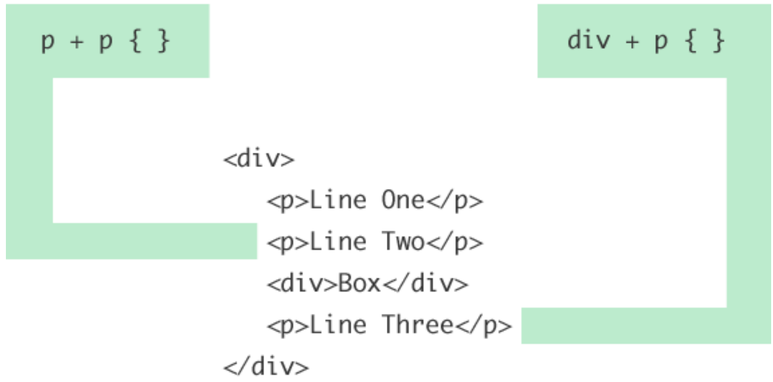


https://css-tricks.com/child-and-sibling-selectors/

Advanced selectors

Adjacent Sibling Selectors

- `p + p`
- `div + p`
  - Target items immediately next to each other

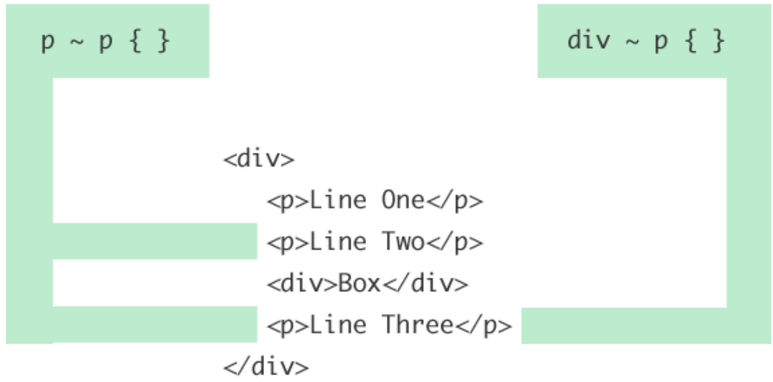


https://css-tricks.com/child-and-sibling-selectors/

Advanced selectors

General Sibling Selectors

- `p ~ p`
- `div ~ p`
  - Target items anywhere after the sibling



https://css-tricks.com/child-and-sibling-selectors/

Fonts & fallbacks

- Browsers will try fonts in order

```
p {
    font-family: "Times New Roman", Times, serif;
}
```

- Google Fonts is a great resource

```
<!--HTML-->
<link href="https://fonts.googleapis.com/css?
family=Roboto" rel=“stylesheet">
/* CSS */
font-family: 'Roboto', sans-serif;
```

https://fonts.google.com/

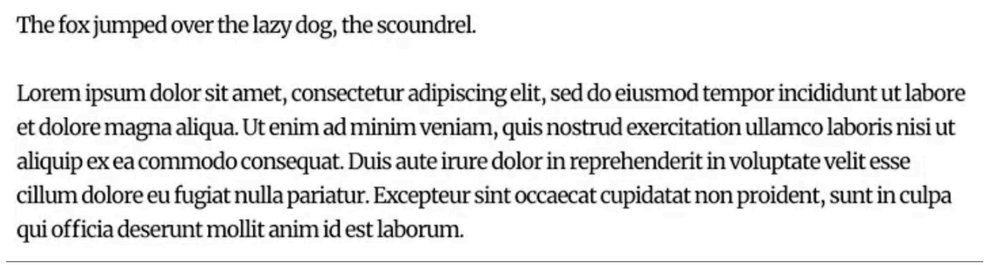
Fallbacks in HTML

- Work similarly to CSS

```
<video autoplay>
 <!--webm not supported in IE or Safari-->
 <source src="lecture3.webm" type="video/webm" />
    <!--mp4 supported in modern browsers, but lower
quality-->
    <source src="lecture3.mp4" type="video/mp4" />
    <!--backup important for some old browsers-->
    <img src="lecture3.jpg" title="Your browser does not
support the <video> tag">
</video>
```

Fallbacks: why?

- Format not supported (webm, ogg, flac)
- Font might not support certain characters
- Might take time to load (“flash of unstyled text”)
  - Pick a similar default font



https://css-tricks.com/css-basics-fallback-font-stacks-robust-web-typography/

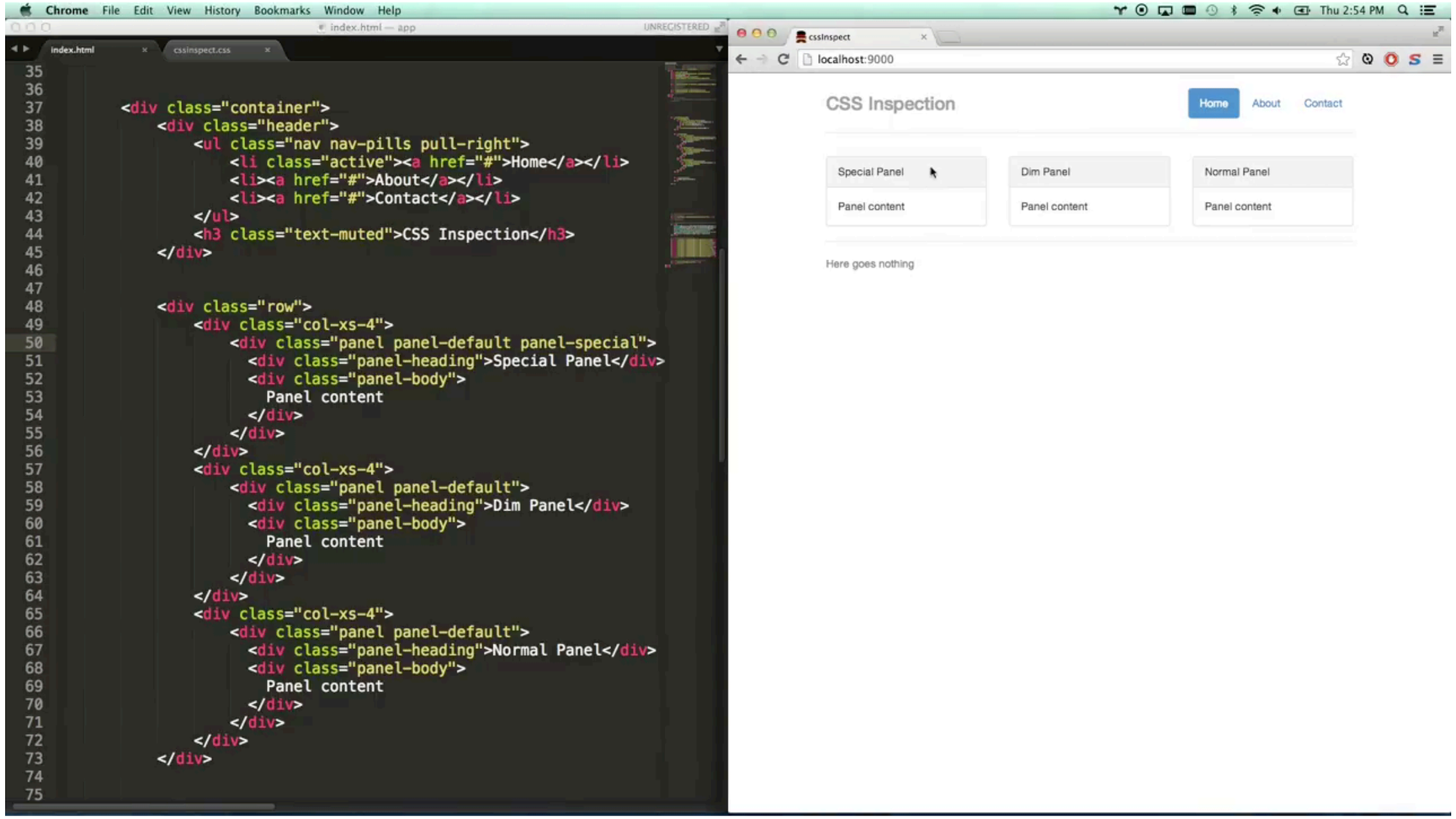
There's a lot to CSS.

I can't create much from memory alone.

References

- https://www.w3schools.com/cssref/
- https://cssreference.io/
- https://developer.mozilla.org/en-US/docs/Web/CSS/Reference
- https://www.codecademy.com/learn/learn-css

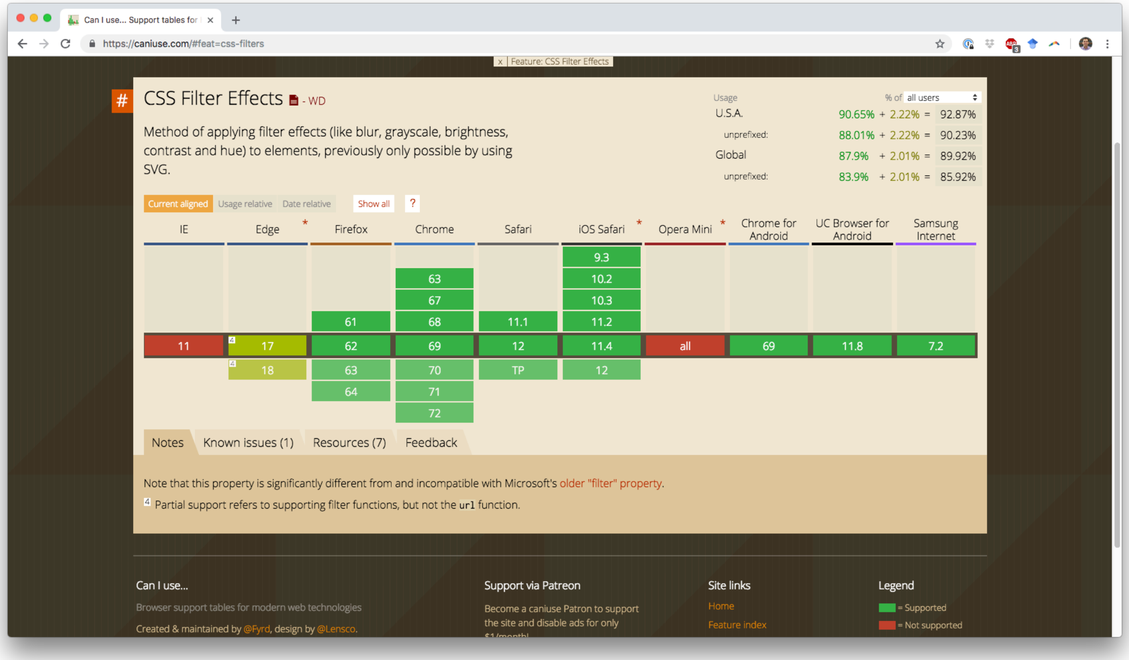
Debugging in browser



https://www.youtube.com/watch?v=Z3HGJsNLQ1E

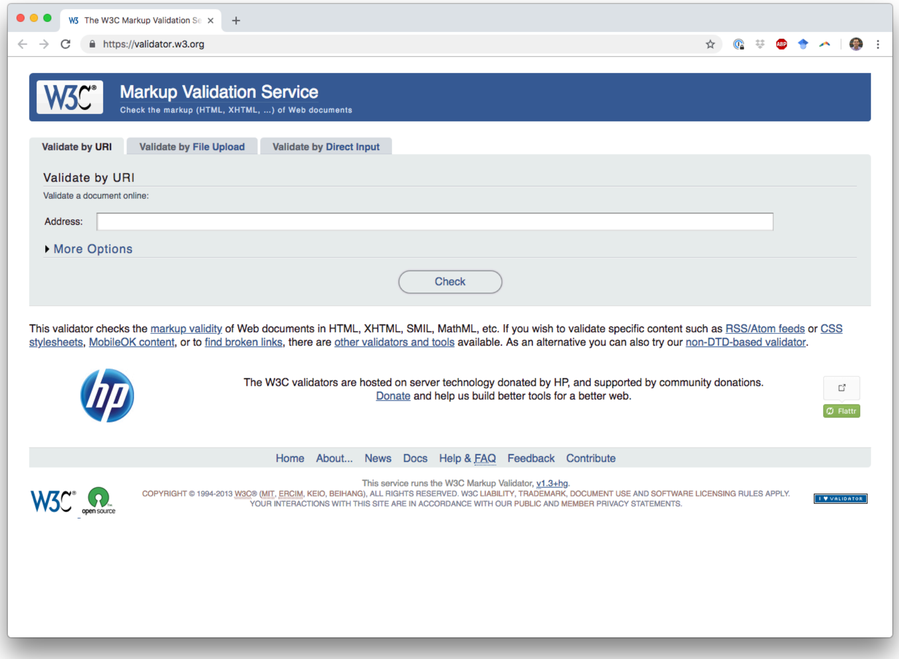
Browser compatibility

Used to be a much bigger issue, but still worth checking



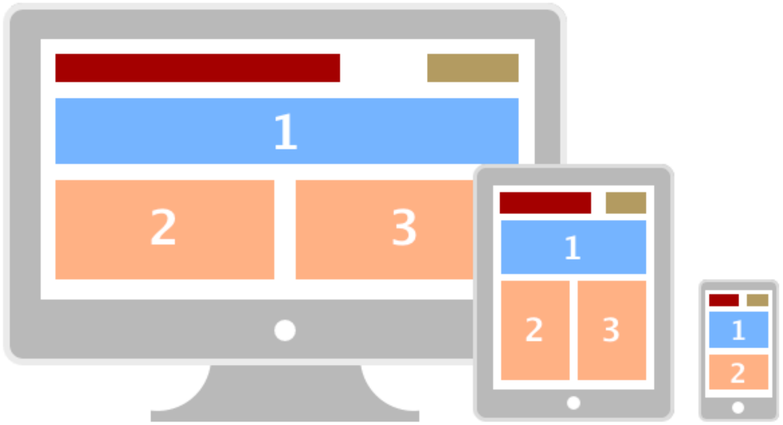
https://caniuse.com/

Validation

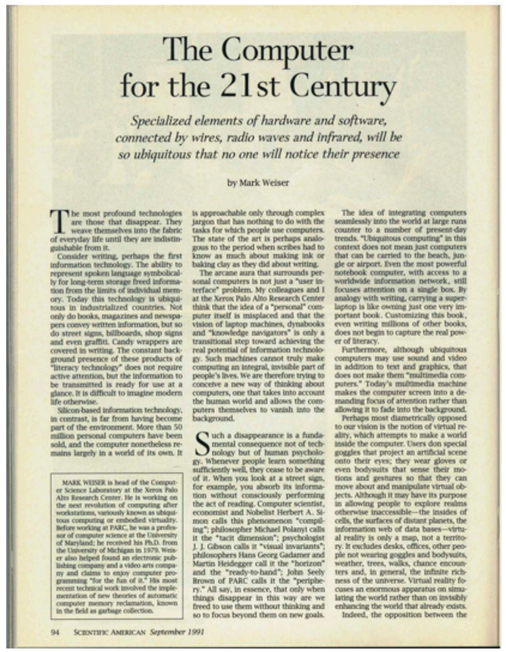


https://validator.w3.org

Switching gears: responsive design



Recall the three waves of computing…



The Computer for the 21st Century

Specialized elements of hardware and software, connected by wires, radio waves and infrared, will be so ubiquitous that no one will notice their presence

by Mark Weiser

The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.

Consider writing, perhaps the first information technology. The ability to represent spoken language symbolically for long-term storage freed information from the limits of individual memory. Today this technology is ubiquitous in industrialized countries. Not only do books, magazines and newspapers convey written information, but so do street signs, billboards, shop signs and even graffiti. Candy wrappers are covered in writing. The constant background presence of these products of "literacy technology" does not require active attention, but the information to be transmitted is ready for use at a glance. It is difficult to imagine modern life otherwise.

Silicon-based information technology, in contrast, is far from having become part of the environment. More than 50 million personal computers have been sold, and the computer nonetheless remains largely in a world of its own. It is approachable only through complex jargon that has nothing to do with the tasks for which people use computers. The state of the art is perhaps analogous to the period when scribes had to know as much about making ink or baking clay as they did about writing.

The arcane aura that surrounds personal computers is not just a "user interface" problem. My colleagues and I at the Xerox Palo Alto Research Center think that the idea of a "personal" computer itself is misplaced and that the vision of laptop machines, dynabooks and "knowledge navigators" is only a transitional step toward achieving the real potential of information technology. Such machines cannot truly make computing an integral, invisible part of people's lives. We are therefore trying to conceive a new way of thinking about computers, one that takes into account the human world and allows the computers themselves to vanish into the background.

Such a disappearance is a fundamental consequence not of technology but of human psychology. Whenever people learn something sufficiently well, they cease to be aware of it. When you look at a street sign, for example, you absorb its information without consciously performing the act of reading. Computer scientist, economist and Nobelist Herbert A. Simon calls this phenomenon "compiling"; philosopher Michael Polanyi calls it the "tacit dimension"; psychologist J. J. Gibson calls it "visual invariants"; philosophers Hans Georg Gadamer and Martin Heidegger call it the "horizon" and the "ready-to-hand"; John Seely Brown of PARC calls it the "periphery." All say, in essence, that only when things disappear in this way are we freed to use them without thinking and so to focus beyond them on new goals.

The idea of integrating computers seamlessly into the world at large runs counter to a number of present-day trends. "Ubiquitous computing" in this context does not mean just computers that can be carried to the beach, jungle or airport. Even the most powerful notebook computer, with access to a worldwide information network, still focuses attention on a single box. By analogy with writing, carrying a superlaptop is like owning just one very important book. Customizing this book, even writing millions of other books, does not begin to capture the real power of literacy.

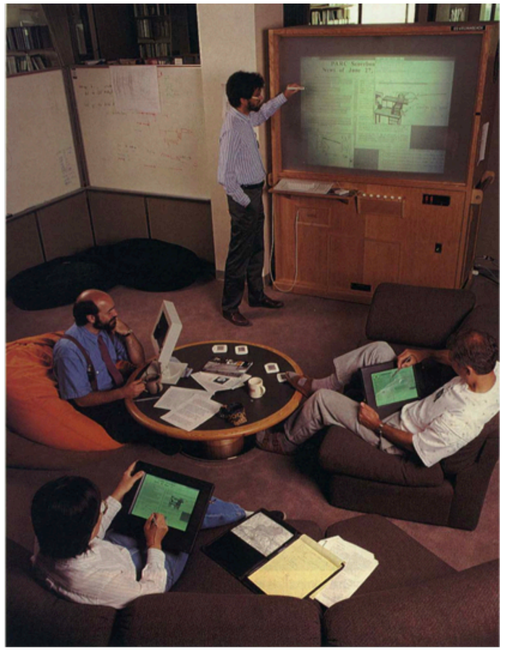
Furthermore, although ubiquitous computers may use sound and video in addition to text and graphics, that does not make them "multimedia computers." Today's multimedia machine makes the computer screen into a demanding focus of attention rather than allowing it to fade into the background.

Perhaps most diametrically opposed to our vision is the notion of virtual reality, which attempts to make a world inside the computer. Users don special goggles that project an artificial scene onto their eyes; they wear gloves or even bodysuits that sense their motions and gestures so that they can move about and manipulate virtual objects. Although it may have its purpose in allowing people to explore realms otherwise inaccessible—the insides of cells, the surfaces of distant planets, the information web of data bases—virtual reality is only a map, not a territory. It excludes desks, offices, other people not wearing goggles and bodysuits, weather, trees, walks, chance encounters and, in general, the infinite richness of the universe. Virtual reality focuses an enormous apparatus on simulating the world rather than on invisibly enhancing the world that already exists.

Indeed, the opposition between the

MARK WEISER is head of the Computer Science Laboratory at the Xerox Palo Alto Research Center. He is working on the next revolution of computing after workstations, variously known as ubiquitous computing or embodied virtuality. Before working at PARC, he was a professor of computer science at the University of Maryland; he received his Ph.D. from the University of Michigan in 1979. Weiser also helped found an electronic publishing company and a video arts company and claims to enjoy computer programming "for the fun of it." His most recent technical work involved the implementation of new theories of automatic computer memory reclamation, known in the field as garbage collection.

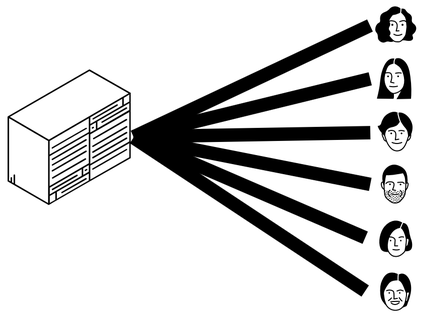
94 SCIENTIFIC AMERICAN *September 1991*



Three waves of computing

1

Mainframe computing



“Many to one”

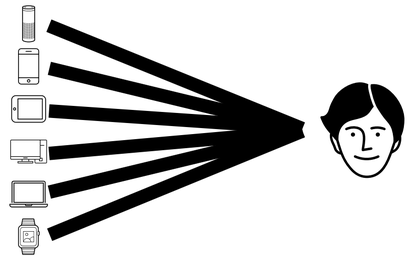
2

Personal computing

“One to one”

3

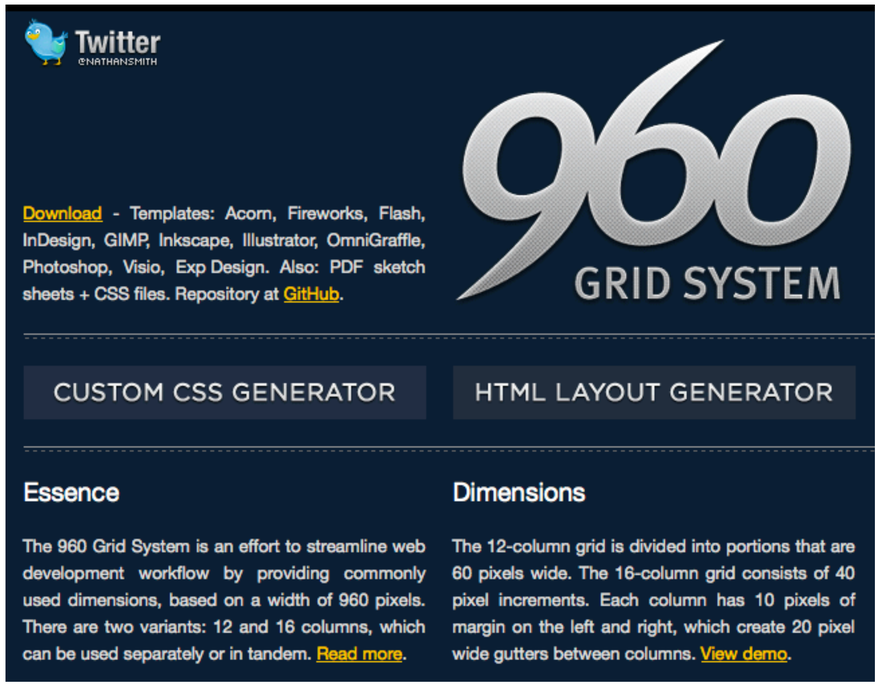
Ubiquitous computing



“One to many”

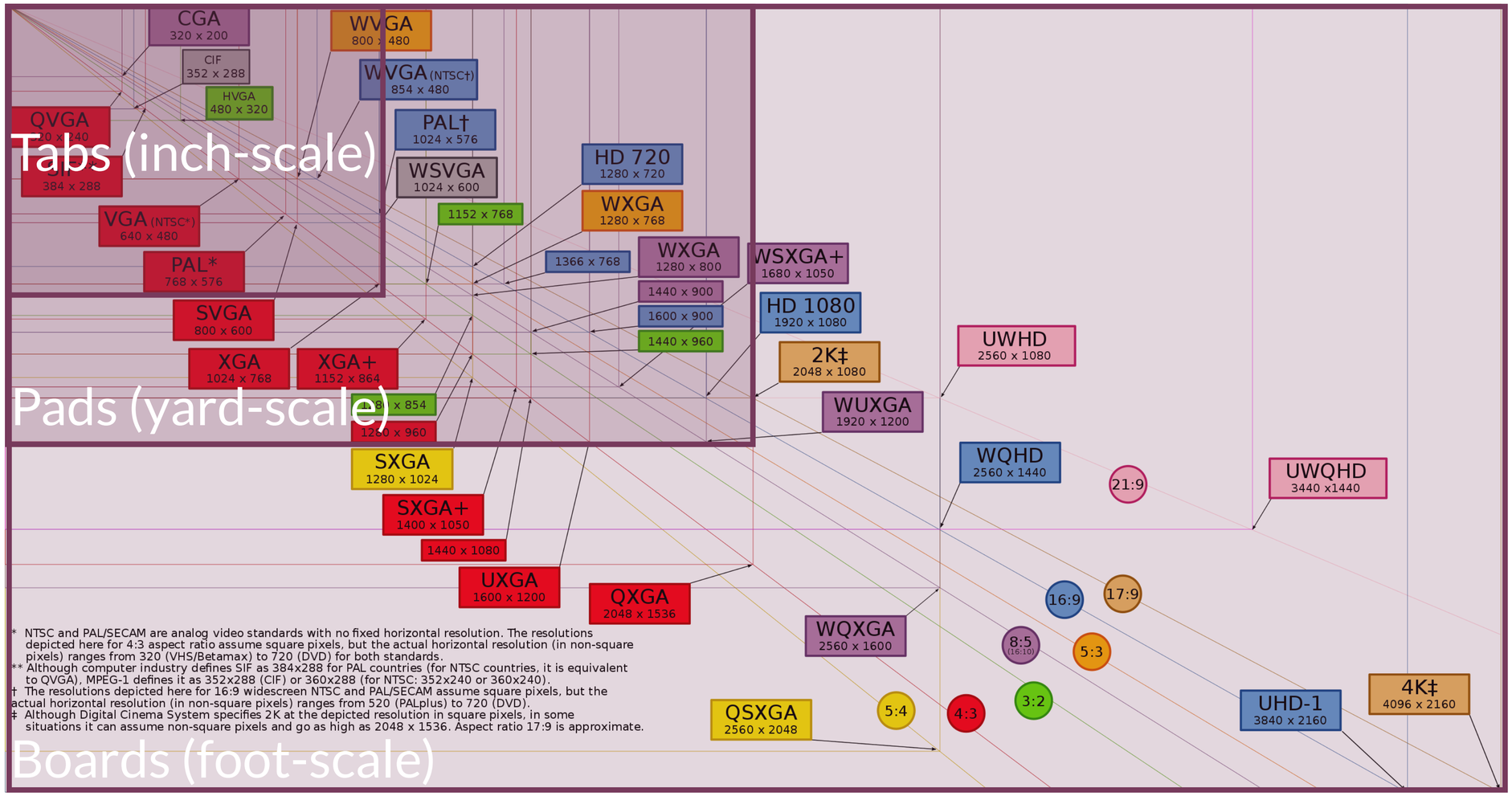
Websites in the personal computing era

- 960 px wide was pretty common
  - Most screens were 1024x978, leave some room for vertical scrollbar
  - Nicely divisible, can create even columns



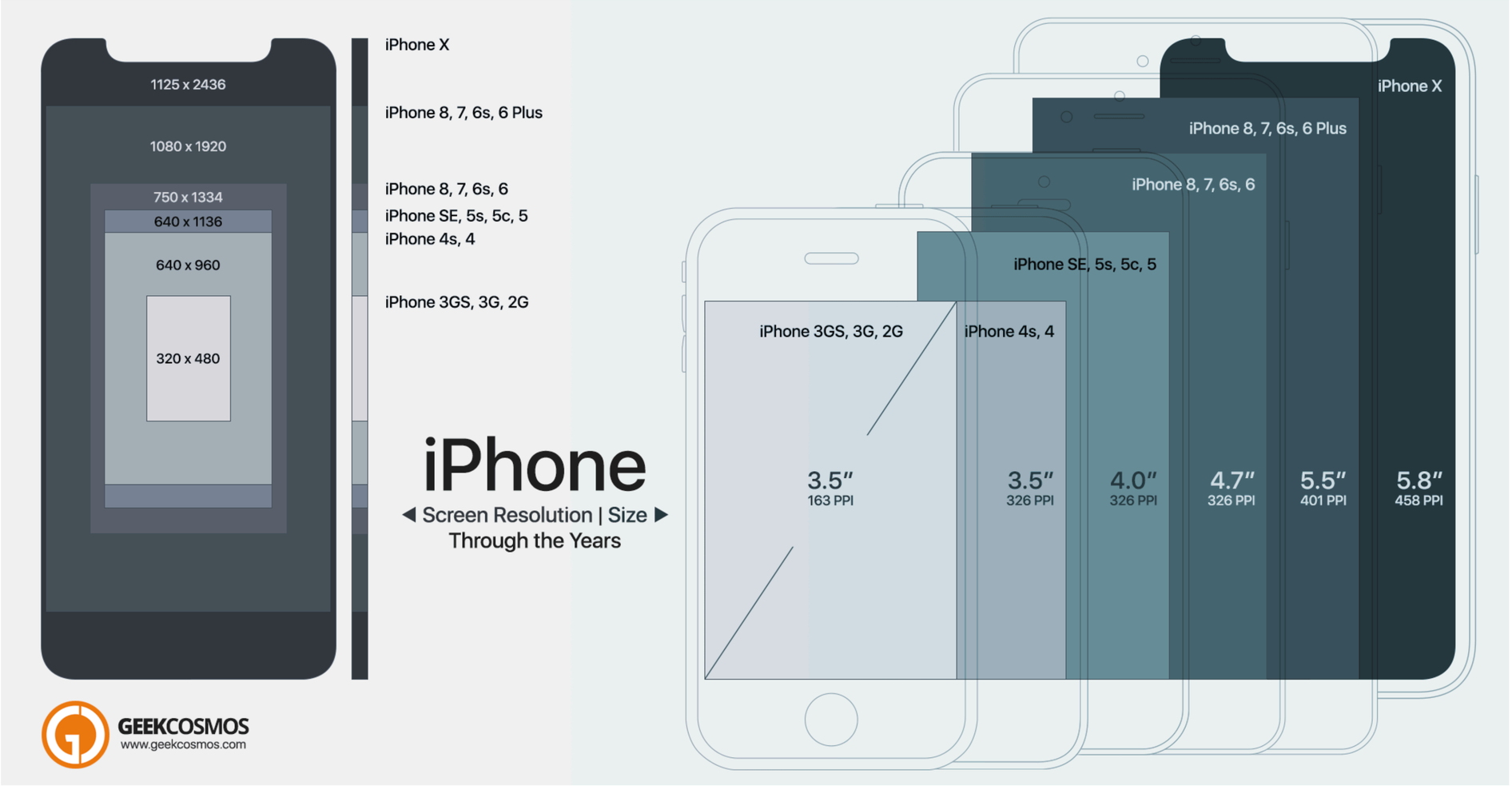
https://960.gs/

Websites today: ubiquitous computing



https://en.wikipedia.org/wiki/Display_resolution

Websites today: just the iPhone!



So… how do we account for this?

Responsive design or Adaptive design

Responsive design

- Develop one set of HTML and CSS which changes layout depending on screen sizes

Adaptive design

- Develop and maintain multiple sets of code, change layout depending on device type and screen size

Question

Responsive or Adaptive?

A. Top is responsive, bottom is adaptive

B. Top is adaptive, bottom is responsive

C. Both are responsive

D. Both are adaptive

E. These are neither responsive nor adaptive

https://css-tricks.com/the-difference-between-responsive-and-adaptive-design/

Question

Responsive or Adaptive?

A. Top is responsive, bottom is adaptive

B. Top is adaptive, bottom is responsive

C. Both are responsive

D. Both are adaptive

E. These are neither responsive nor adaptive

https://css-tricks.com/the-difference-between-responsive-and-adaptive-design/

Responsive design

+ Easier to maintain one code base, future-proof

- Worse performance; requires downloading entire stylesheet

- Emphasis on making it “look right” rather than creating an experience

Adaptive design

+ Can cater experience to a device’s capabilities and performance

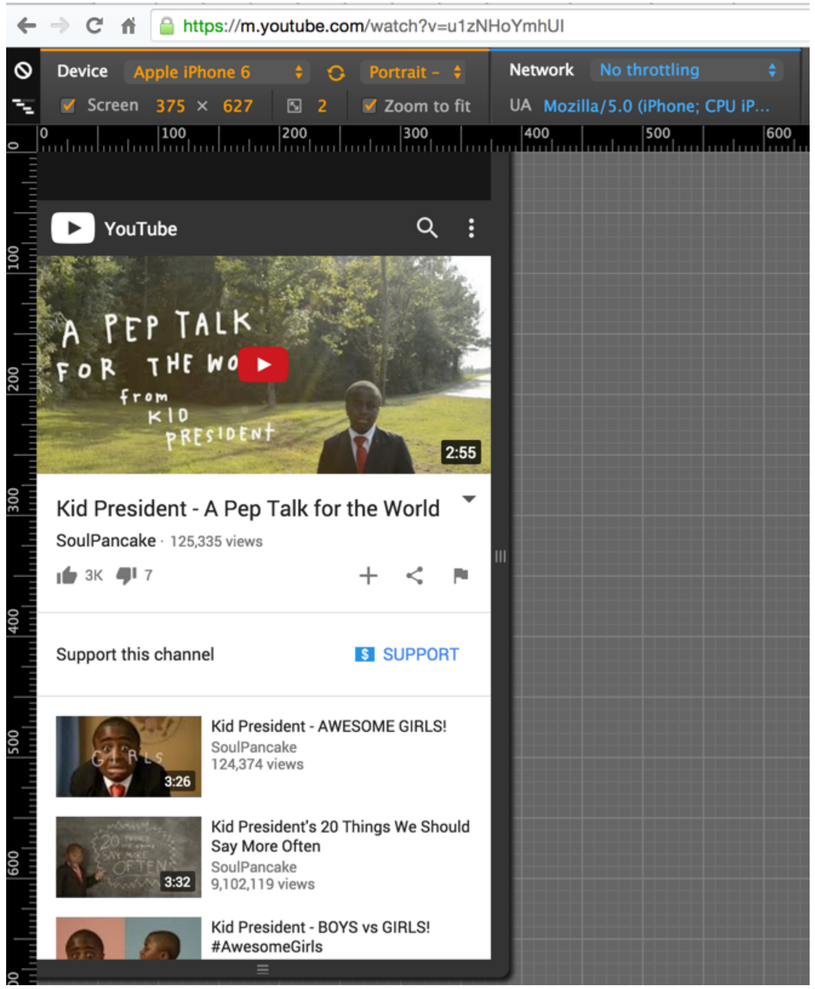
- Much more difficult to maintain separate codebases

- Limits development to a few key capabilities because you have to implement for everything

**Most pages are responsive,
but sometimes it’s crucial
to create the best experience**

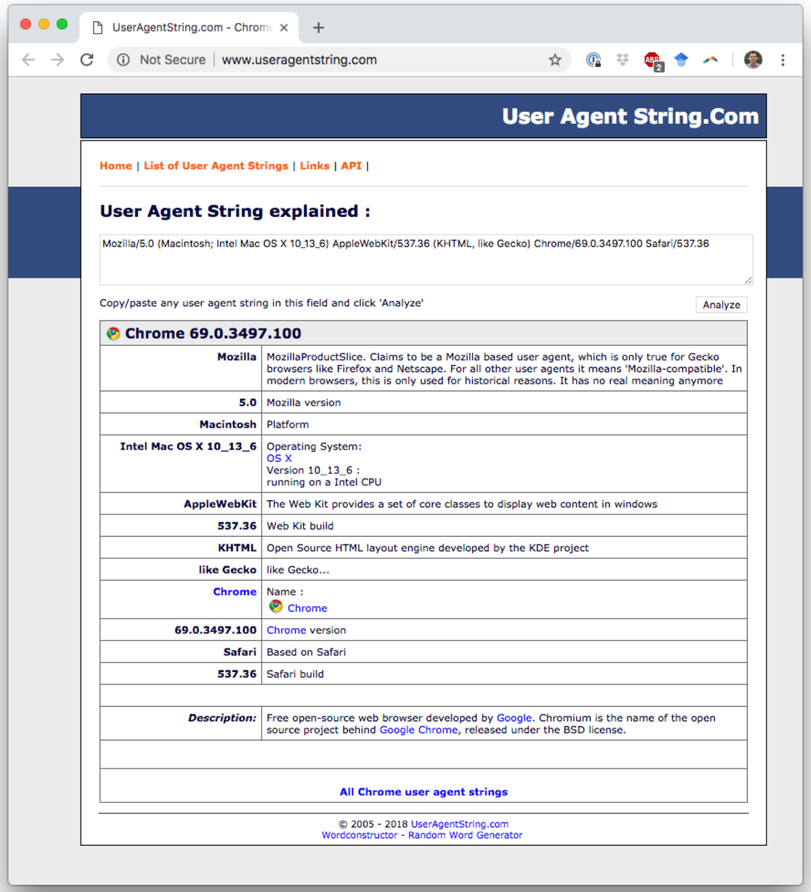
Adaptive design

- Video = a lot to load
  - Why send a higher resolution than the screen can render?
  - Why use up your own bandwidth?
  - Laggy videos mean unhappy users
- Google can afford the development burden



Adaptive design

- User agent string accessible via JavaScript
  - `navigator.userAgent`
- There's usually a better way
  - Do you care about the browser or operating system? Or is resolution sufficient?
  - Can be spoofed or incorrect



https://developer.mozilla.org/en-US/docs/Web/HTTP/Browser_detection_using_the_user_agent

Adaptive design

- Media queries in CSS

```
/* CSS */
@media screen and (device-width: 375px) and (device-height: 667px)
and (-webkit-device-pixel-ratio: 2) {
  /* iPhone 8-specific CSS */
}
```

- Load appropriate external stylesheet

```
<!--HTML-->
<head>
  <link rel="stylesheet" media="screen and (device-width: 375px)
  and (device-height: 667px) and (-webkit-device-pixel-ratio: 2)" href="iPhone8.css">
</head>
```

Media query syntax

- `@media`
- `screen`, `print`, `speech`, `all`
- `min-width`, `max-width`
- `orientation`, `-webkit-min-device-pixel-ratio`
- Many, many more

https://www.w3schools.com/cssref/css3_pr_mediaquery.asp

Moving away from adaptive design, transitioning to responsive design

Breakpoints

- The point at which your design “breaks”
  and is no longer visually appealing or usable

- Designs vary, but most have 3-5 breakpoints

  - extra small (old mobile), small (mobile), medium (tablet), large (laptop or desktop), extra large (wide desktop or wall display)

  - Again, somewhat similar to Weiser’s three types of computers

Breakpoints

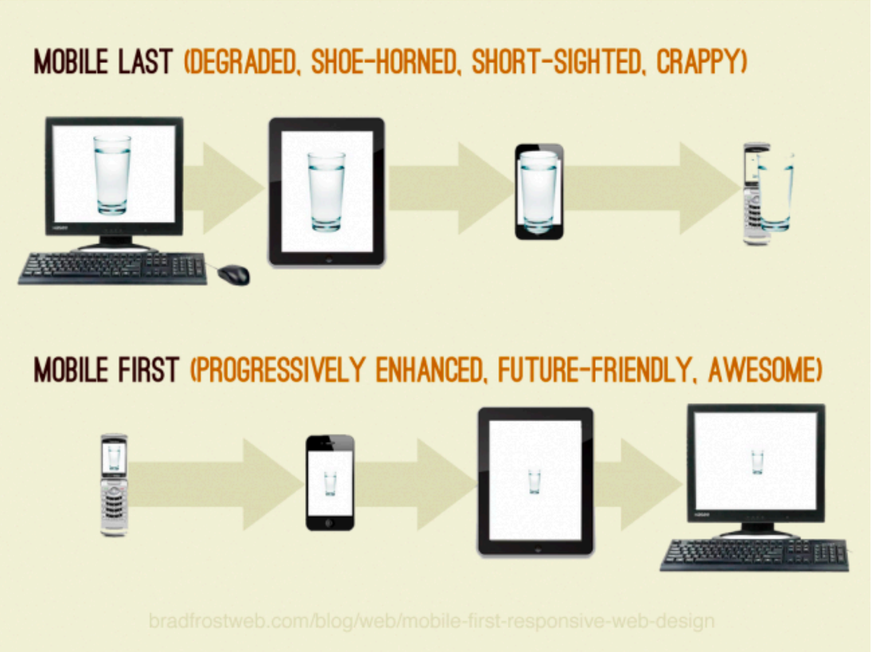
```
@media screen and (max-width: 640px) {
  /* small screens */
}

@media screen and (min-width: 640px and max-width:
1024px) {
  /* medium screens */
}

@media screen and (min-width: 1024px) {
  /* large screens */
}
```

Responsive design

- Fluid grids
  - Lay out content in columns whose widths can vary
  - Bootstrap helps with this; more on that in a bit
- Flexible images
  - Let image size change based on screen layout
  - Put images in containers which will scale appropriately
  - Set `width: 100%`, `max-width: 100%`, `height: auto`

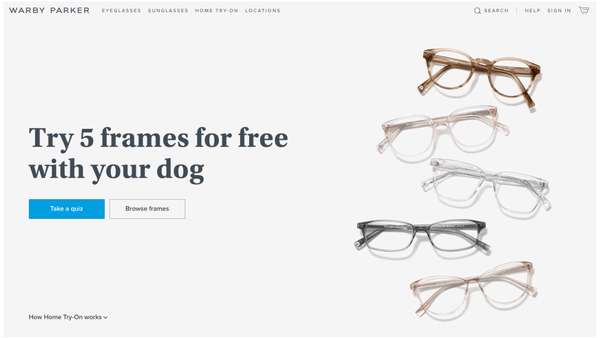
Mobile-first design

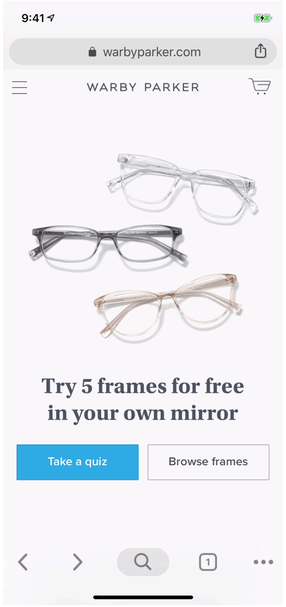
- “Graceful degradation” vs. “progressive enhancement”
- Plan your design for mobile
- Then make your app *better* with more real estate
  - Add more features
  - Make existing features easier to navigate



A few tips for mobile design

- Show the same content, organize it appropriately
- Stack content vertically
- Show navigation on demand
- Larger touch targets

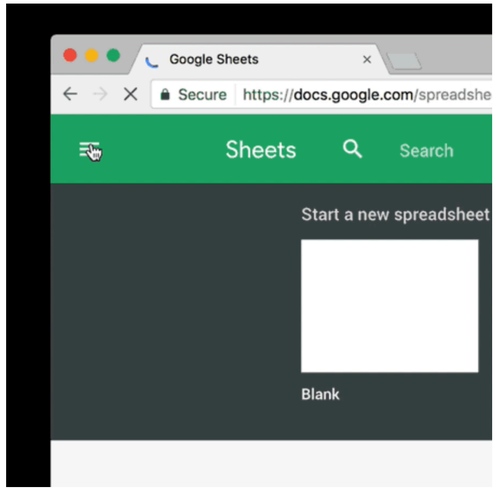


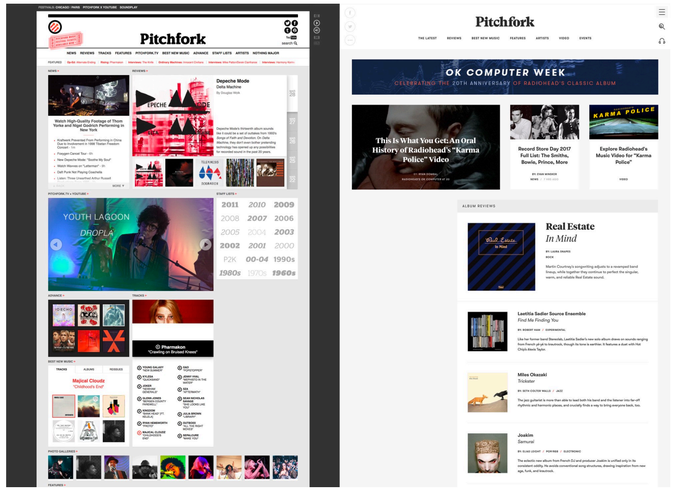


https://www.bluefountainmedia.com/blog/desktop-vs-mobile-three-key-website-design-differences

Mobile-first, not mobile-only

- Copying mobile UI to desktop creates inefficiencies
  - Extra clicks to navigate
  - Underutilized real estate





https://blog.prototypr.io/mobile-first-desktop-worst-f900909ae9e2

Mobile-first, not mobile-only

- Plan your design for mobile
- But consider how the experience should change on desktop, etc.
- Go beyond making everything bigger
  - *Enhance* your design

Today’s goals

By the end of today, you should be able to…

- Explain the goals of CSS and why it exists as separate from HTML
- Describe the CSS hierarchy and fallback structure
- Utilize the box model and positioning options to arrange content
- Style nested tags with child, adjacent sibling, and general sibling selectors
- Describe how responsive and adaptive design differ
  and when you might prefer one or the other
- Explain the advantages and disadvantages of a mobile-first design

IN4MATX 133: User Interface Software

Lecture 3:
CSS & Responsive Design

Professor Daniel A. Epstein
TA Lucas de Melo Silva
TA Jong Ho Lee

Additional slides

Specifying styles

Inline Styling

```
<p style="font-family='Arial'; color=red;">
  Red text
</p>
<p style="font-family='Arial'; color=red;">
  More red text
</p>
```

- Supported, but usually bad practice
  - Goes against DRY principles of programming (Don’t Repeat Yourself)

Specifying styles

Internal Styling

```
<head>
  <style type="text/css">
    p {font-family:'Arial'; color:red;}
  </style>
</head>
<body>
  ...
</body>
```

- Just putting CSS into the `<head>` of your HTML

Specifying styles

External Styling

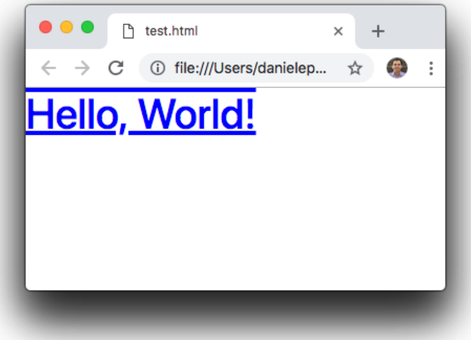
```
<head>
  <link rel="stylesheet" href="./css/style.css">
</head>
<body>
  ...
</body>
```

- Generally a best practice
  - Aligns with the idea of separating structure from style

Specifying styles

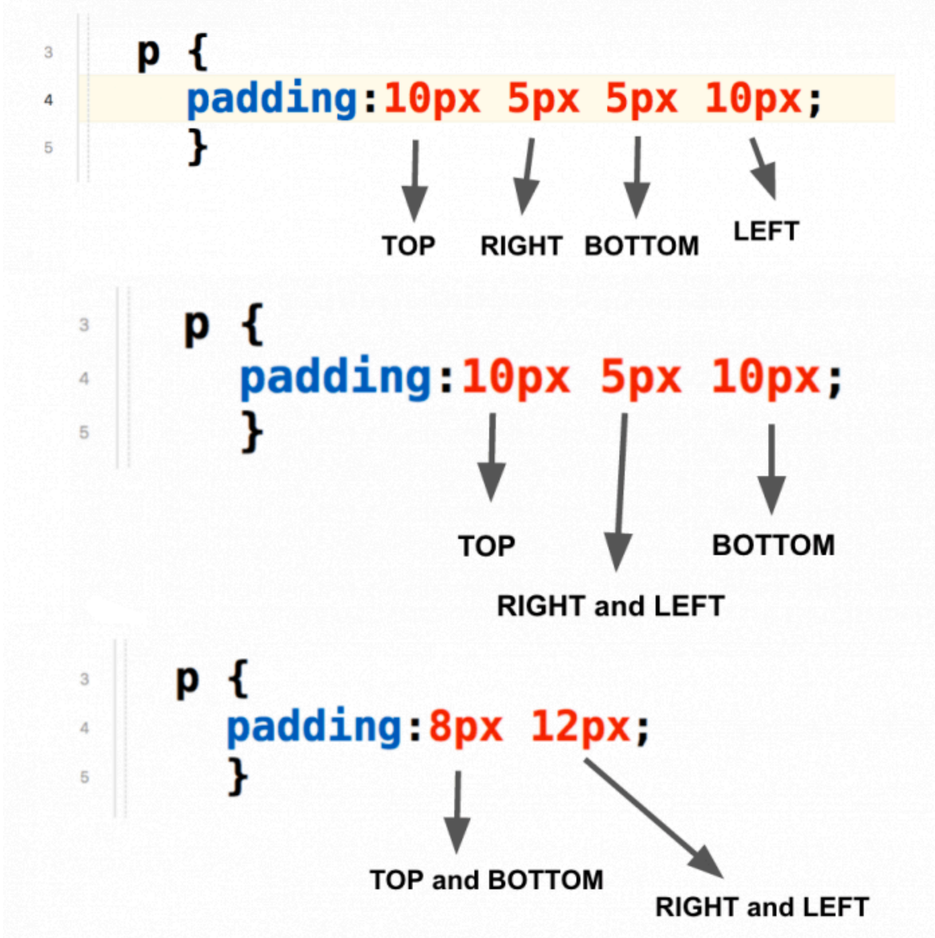
- External styles apply in order, too!

```
<head>
  <link rel="stylesheet"
  href="./css/bootstrap.css">
  <link rel="stylesheet"
  href="./css/style.css">
</head>
<body>
  <h1>Hello, World!</h1>
</body>
```



Positioning: shorthand

- Multiple values can be specified in one line
- Difficult to remember
- Being explicit improves readability at the expense of brevity



https://css-tricks.com/remember-the-order-of-marginpadding-shorthand-with-trouble/

Borders: shorthand

- Multiple values can be specified in one line
- Slightly easier to remember
- Maybe even more readable

```
div {
  border-bottom-width: 3px;
  border-bottom-style: solid;
  border-bottom-color: red;
}

div.equivalent {
  border-bottom: 3px solid red;
}
```

https://www.w3schools.com/css/css_border.asp

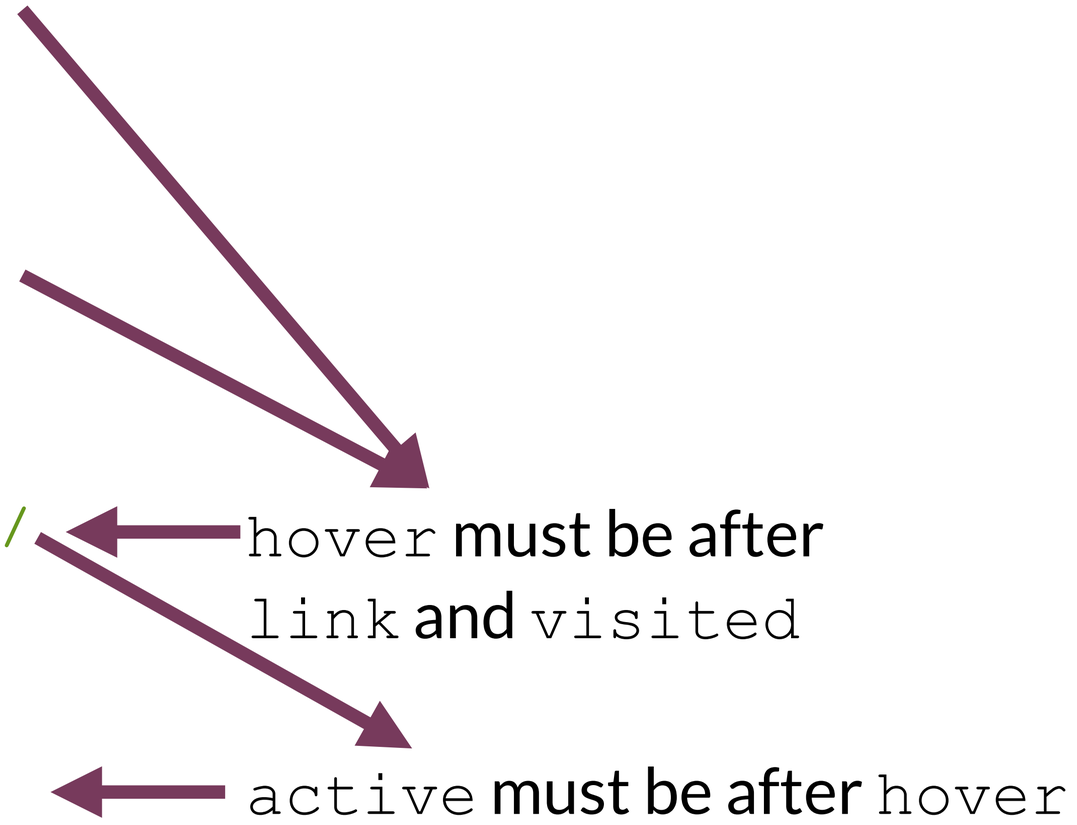
Pseudo-classes

- Define a special state of an element

```
/* CSS Pseudocode */
Selector:pseudo-class {
  property: value;
  property: value;
  ...
}
```

Pseudo-classes

```
a:link { /* unvisited link */
    color: #FF0000;
}

a:visited { /* visited link */
    color: #00FF00;
}

a:hover { /* mouse over link */
    color: #FF00FF;
}

a:active { /* selected link */
    color: #0000FF;
}
```



https://www.w3schools.com/Css/css_pseudo_classes.asp