IN4MATX 133: User Interface Software

CSS & Responsive Design

Today's goals

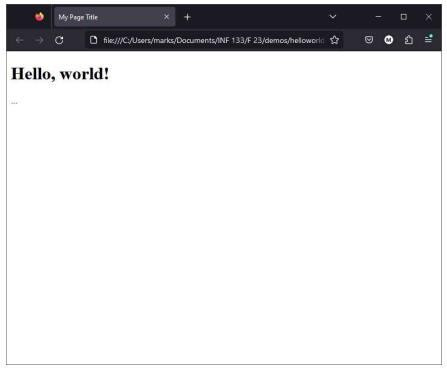
By the end of today, you should be able to…

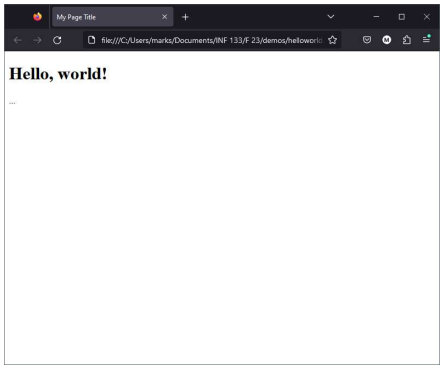
- Explain the goals of CSS and why it exists as separate from HTML
- Describe the CSS hierarchy and fallback structure
- Explain the importance of accessible and semantically meaningful markup
- Generate markup which meets accessibility standards
- Describe how responsive and adaptive design differ
  and when you might prefer one or the other
- Explain the advantages and disadvantages of a grid-based layouts

HTML structure

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="author" content="your name">
    <title>My Page Title</title>
</head>
<body>
    <h1>Hello, world!</h1>
    ...
</body>
</html>
```

```
<html>
<head>
    <title>My Page Title</title>
</head>
<body>
    <h1>Hello, world!</h1>
    <p>...
```





Why does HTML structure matter?

Taking a step back:

Web access is important

How do you use the web?

“The power of the Web is in its universality.
Access by everyone regardless of disability is an essential aspect.”

–Tim Berners-Lee, inventor of the World Wide Web and 2016 Turing award winner
https://www.w3.org/WAI/fundamentals/accessibility-intro/

All sorts of people
will use the webpage you create

Common Disabilities that Affect Technology Use

- Vision
  - Blind, low vision, colorblind
- Auditory impairments
  - Deaf, hard of hearing
- Motor impairments
  - Arthritis, cerebral palsy, tremors, paralysis
- Cognitive impairments
  - Autism, dyslexia, language barriers
- Much more

The technology exists…

…but does the software support it?

How do we support easy navigation with a screen reader?

How do we support easy navigation with a screen reader?

Add semantic meaning to tags

Semantic (landmark) elements

ARIA roles—the “old” way

- Give non-semantic elements (like `<div>`s) a `role` attribute to provide semantic meaning

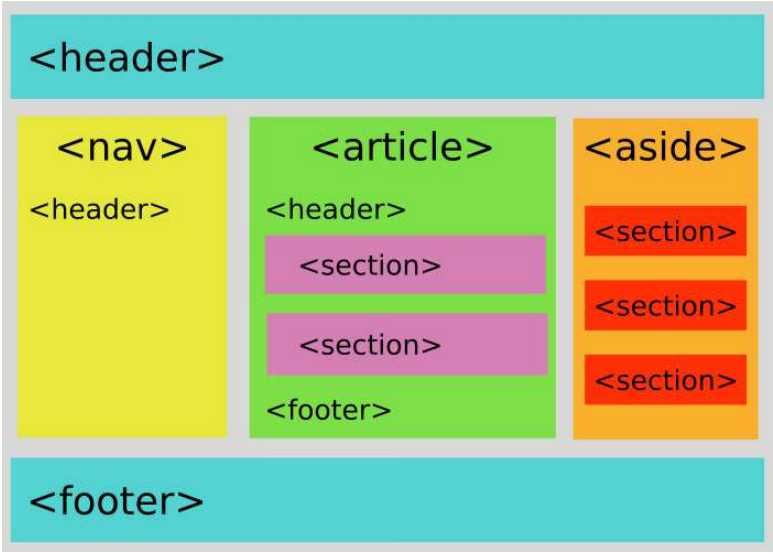
```
<div role="main">

<div role="navigation">

<div role="form">
```

- https://www.w3.org/TR/wai-aria-practices/examples/landmarks/HTML5.html

Semantic (landmark) elements

HTML5 tags—the “new” way

- Dedicated semantic tags
- https://www.w3schools.com/html/html5_semantic_elements.asp



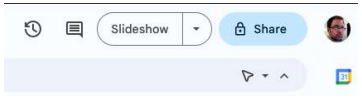
A few other accessibility examples

- “`alt`” attributes in images
- “`aria-label`” attributes to describe non-visual elements (like buttons)

```
<button aria-label="Close">X</button>
```

Alt text guidelines

1. Always include an `alt` attribute, even if it's empty
2. Describe the information, not the picture
3. "Active" images and images which contain information require descriptive alt text
4. Decorative images should have empty alt text
5. Be succinct, avoid being redundant with text

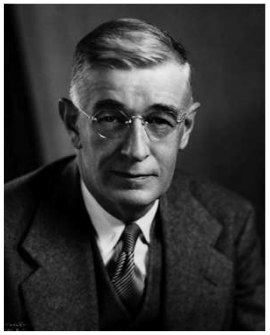


Icons in Google Docs

Cover photos on Twitter/Facebook

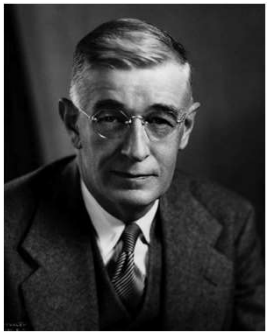
- https://webaim.org/techniques/alttext/
- https://www.abilitynet.org.uk/blog/five-golden-rules-compliant-alt-text

Which `alt` text would best describe this image?



```
<img src="vannevar.jpg" alt="Vannevar Bush">

<img src="vannevar.jpg" alt="A picture of Vannevar Bush">

<img src="vannevar.jpg" alt="A black-and-white photo">

<img src="vannevar.jpg" alt="">

<img src="vannevar.jpg" alt="Vanneaver Bush, writer of 'As
We May Think', posing for a photograph">
```

Which `alt` text would best describe this image?



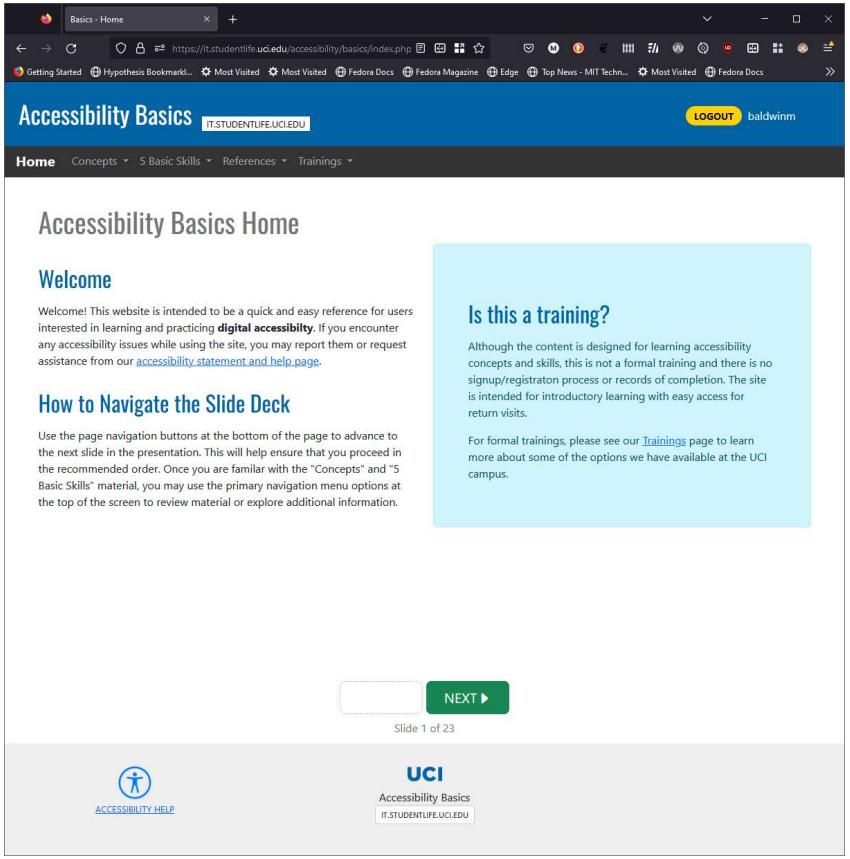
```
<img src="vannevar.jpg" alt="Vannevar Bush">

<img src="vannevar.jpg" alt="A picture of Vannevar Bush">

<img src="vannevar.jpg" alt="A black-and-white photo">

<img src="vannevar.jpg" alt="">

<img src="vannevar.jpg" alt="A picture of Vanneaver Bush,
writer of 'As We May Think'">
```

Accessibility validators

- WAVE http://wave.webaim.org/
- Achecker https://achecks.org/achecker/
- Both over-report problems, requires you to think through whether something is actually an accessibility issue
- Can try on your own with a screen reader
  - VoiceOver (Mac, under Settings -> Accessibility)
  - NVDA (Windows, requires download from https://www.nvaccess.org/)

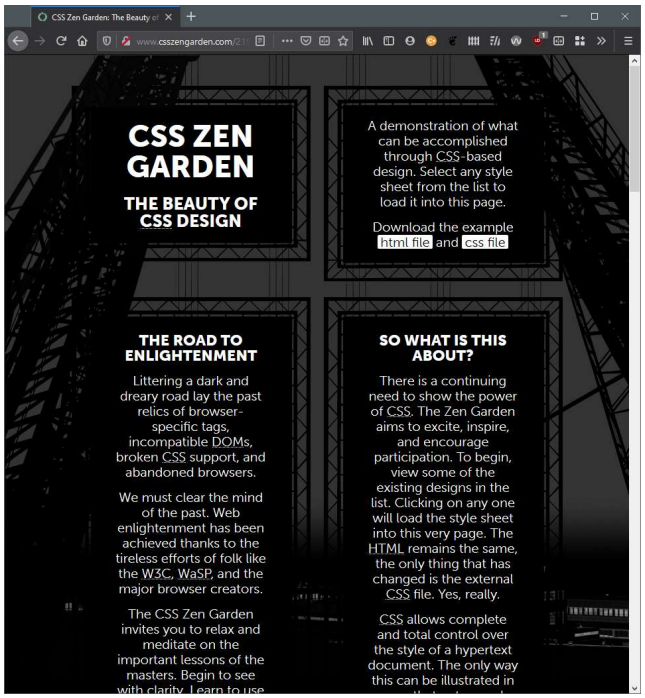
Accessibility Resource

https://it.studentlife.uci.edu/accessibility/basics/



From Semantics to Appearance

Same page, different stylesheets



http://www.csszengarden.com/

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
  - Styles can be contained in a single type of location (style sheets)

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

Apply to all h1 tags

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

"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

```
p {
  font-family: 'Arial';
  color: red;
}
```

- class: a type of element

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

- id: one specific element

```
#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/

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

HTML vs. CSS

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

HTML vs. CSS

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

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

Conflates appearance and semantics

Says nothing about appearance

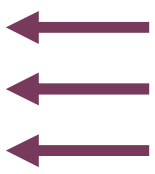
This text is *emphasized*

This text is also *emphasized*

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

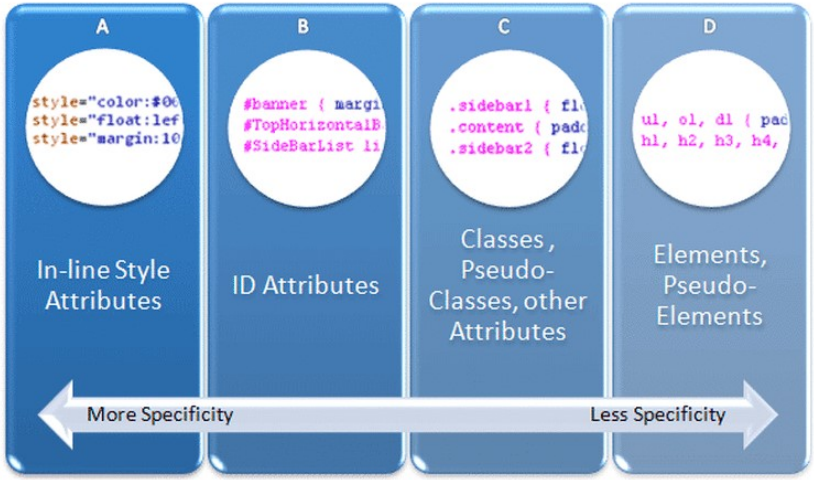
- Rules are applied **in order** (last rule always wins among peer selectors)

```
<!--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



https://www.techrepublic.com/blog/web-designer/css-specificity-hierarchy-what-you-need-to-know/

Positioning

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

```
<p>
  This is on a line.
  <em>This is on the same line.</em>
</p>
<p>This will be on a new line.</p>
```

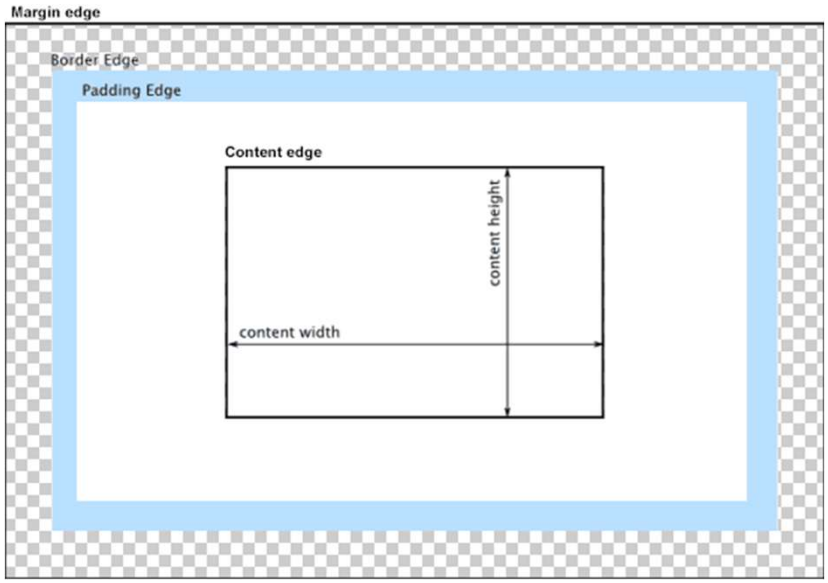
Block

Inline

- Don’t put block elements inside inline elements!

Positioning: box model

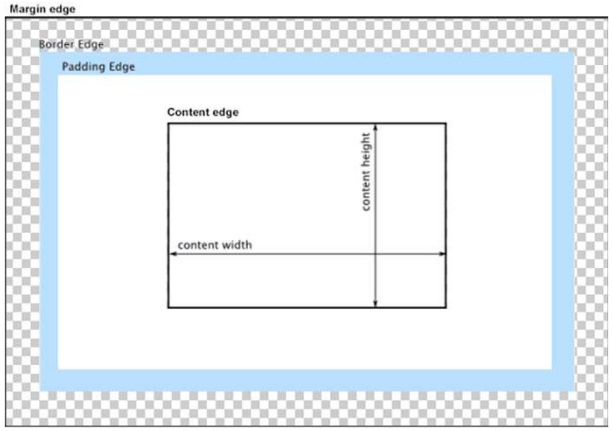
- **Content** contains “real” content
- **Padding** extends content area
- **Border** is similar
- **Margin** is intended to separate elements from neighbors



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

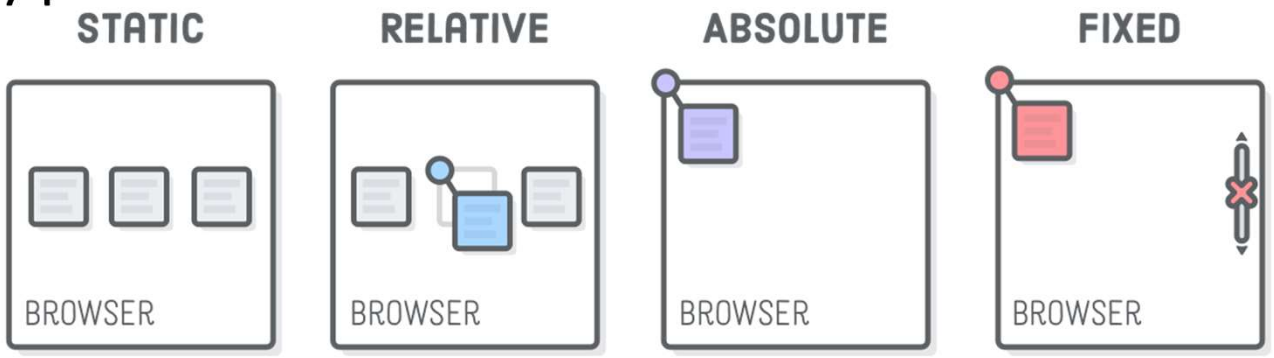


https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Box_Model/Introduction_to_the_CSS_box_model

Positioning

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

Positioning: types



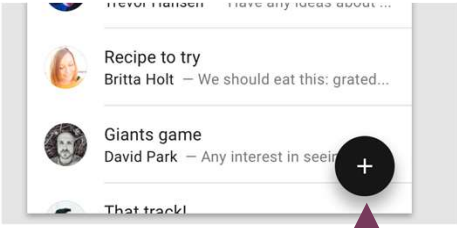
- `static` (default)
- `relative` (offset from default)
- `absolute` (from top-left)
- `fixed` (absolute + floating, fixed to the viewport)

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

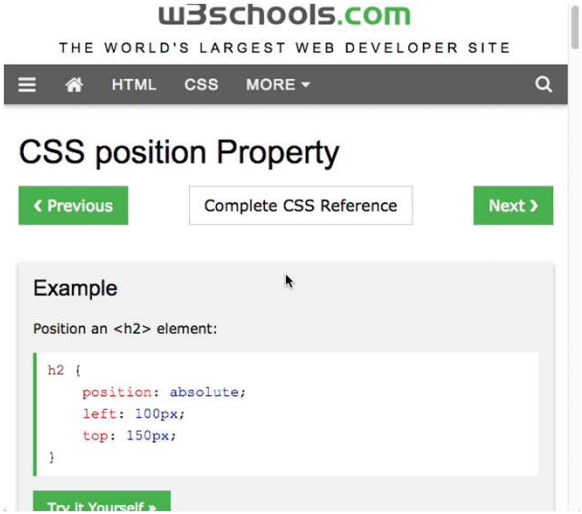


Absolute position

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

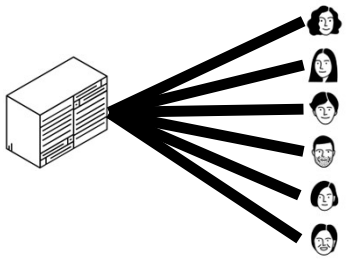
	Recommended	Occasional use	Not recommended
Screen	em, px, %	ex	pt, cm, mm, in, pc
Print	em, cm, mm, in, pt, pc, %	px, ex	

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

Three waves of computing

1

Mainframe computing



"Many to one"

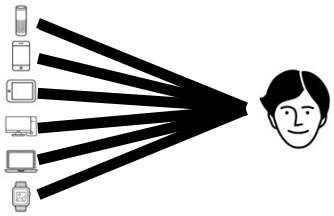
2

Personal computing

"One to one"

3

Ubiquitous computing



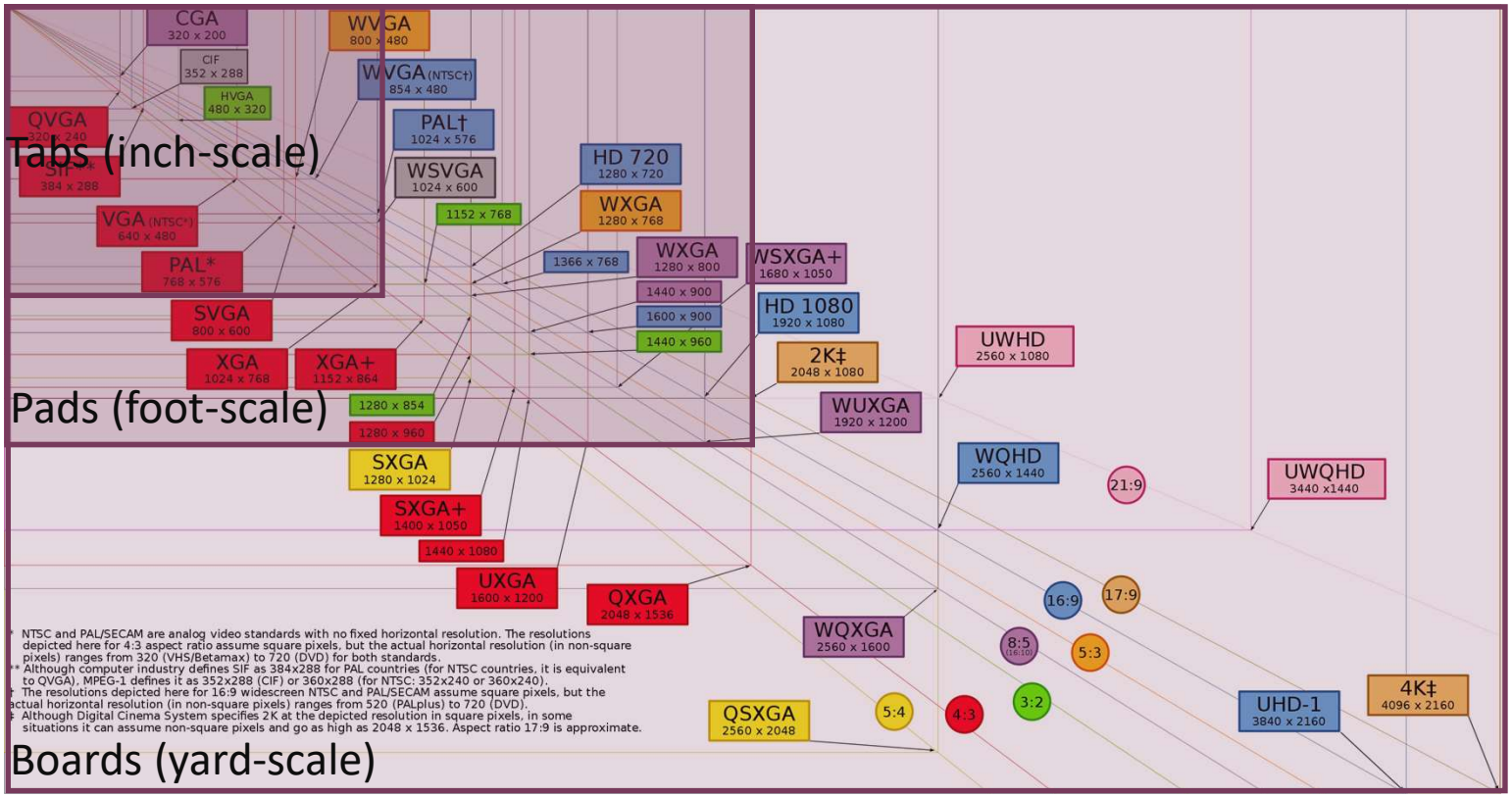
"One to many"

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 vs Adaptive

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

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 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 vs Adaptive

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

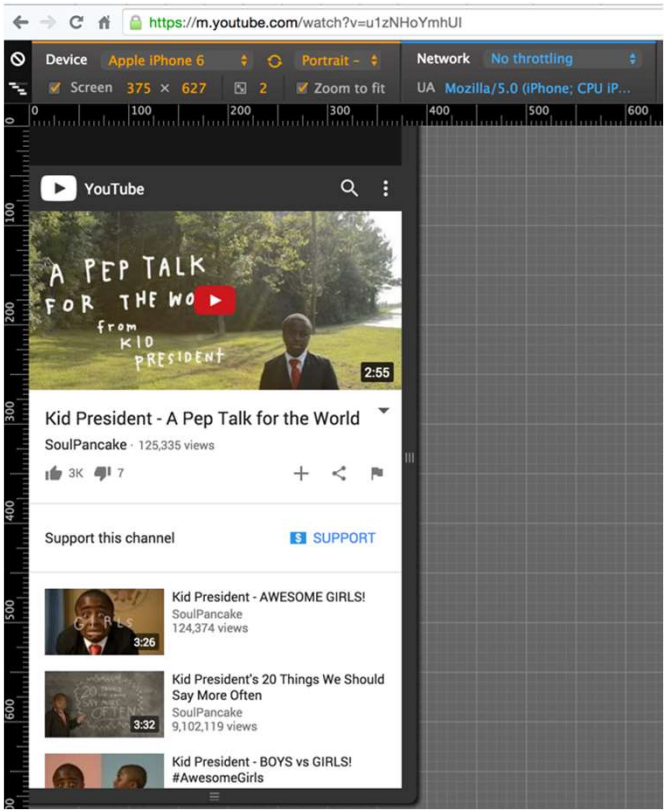
- Much more difficult to maintain separate codebases

- 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

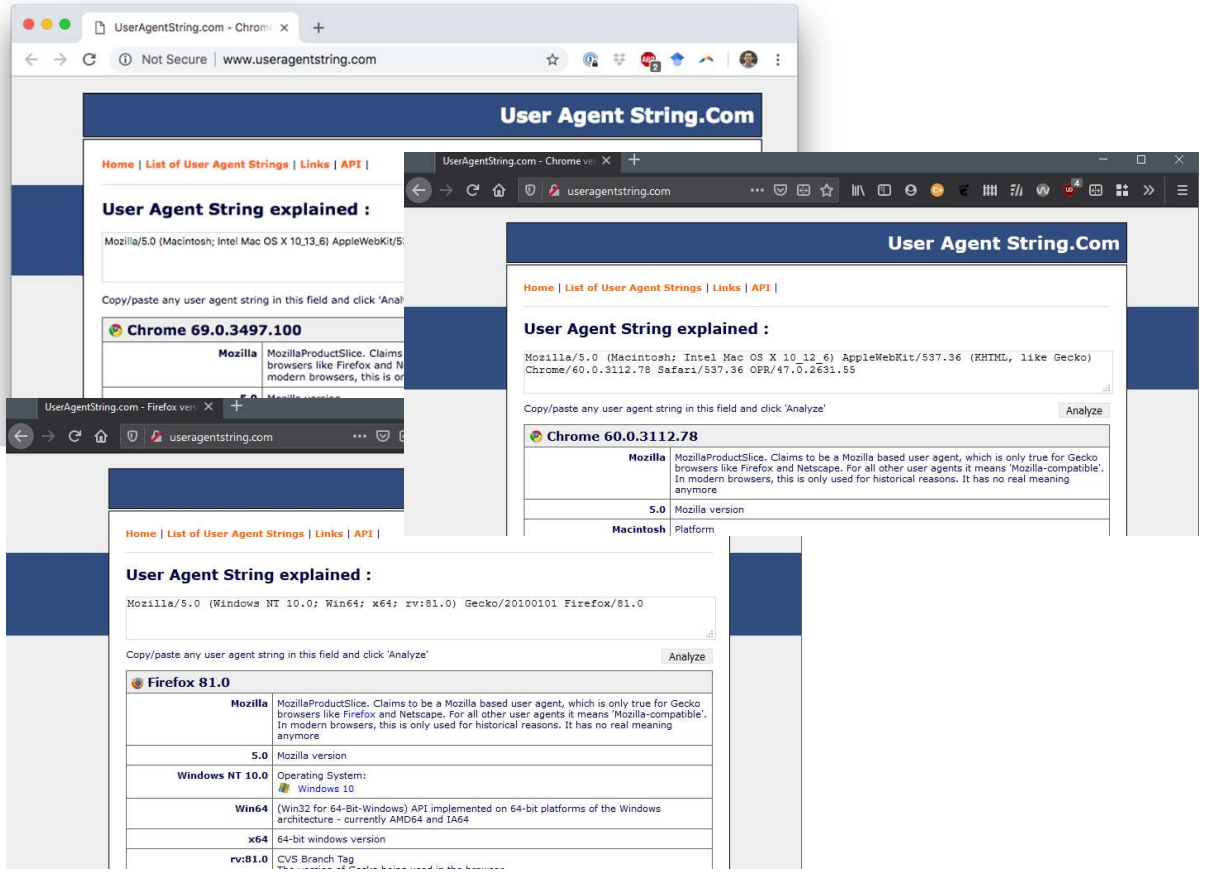
Adaptive design

- 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

Responsive design

- Fluid grids
  - Lay out content in columns whose widths can vary
  - Bootstrap (and other CSS toolkits) 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`

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

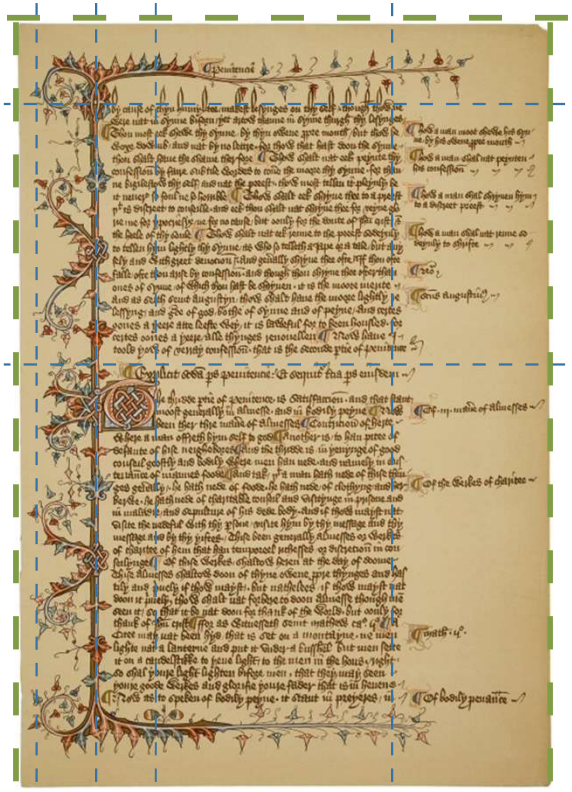
Breakpoints

```
@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 */
}
```

Grid-based layouts

Grid-based layouts

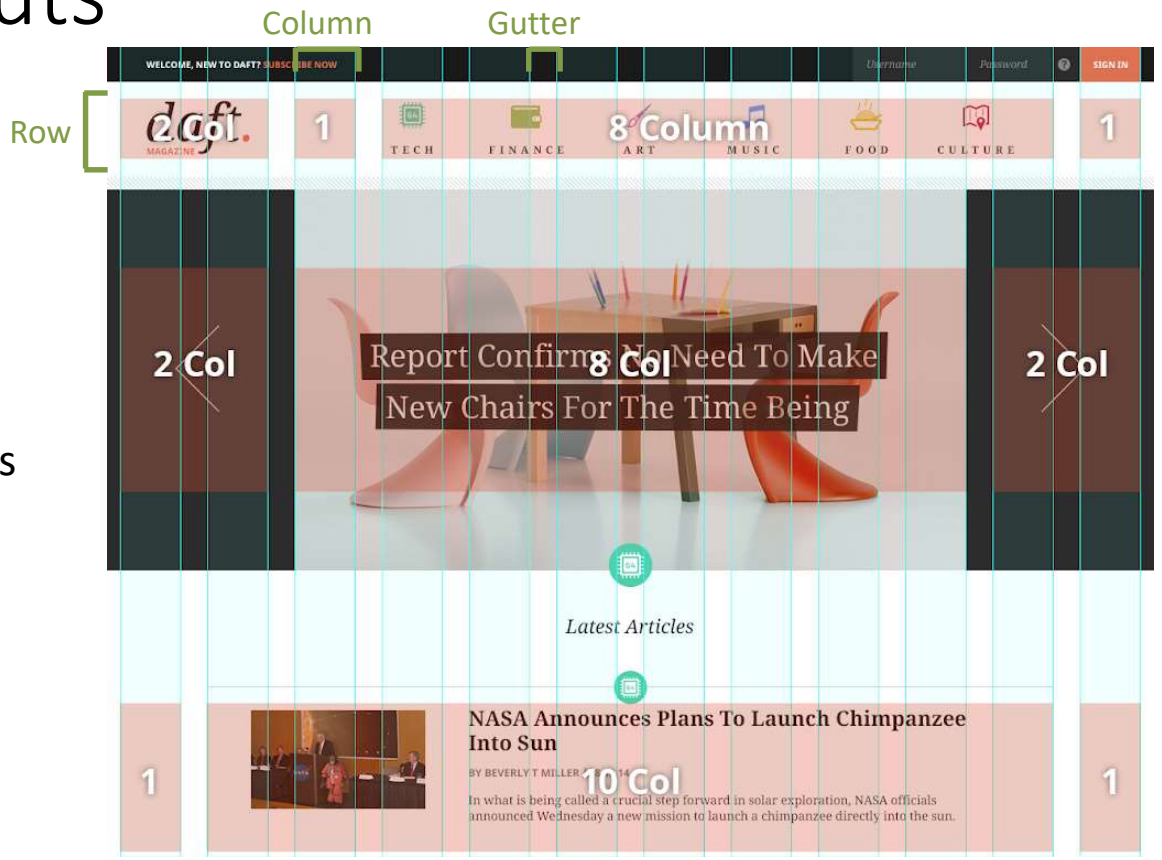
- Established tool for content arrangement
- Gridded content is familiar and easy to follow
- In general, it's good to target fewer lines
  - But breaking that rule is important for creativity and attention-grabbing



http://printingcode.runemadsen.com/lecture-grid/

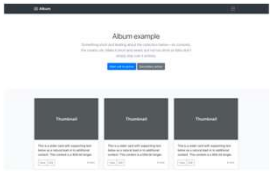
Grid-based layouts

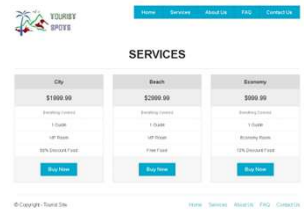
- Rows
- Columns
- Gutters
- Padding/spacing
  - Defined by specific elements



Grid-based frameworks

- Bootstrap (https://getbootstrap.com/)
  - Most popular, most extensions
- Foundation (https://foundation.zurb.com/)
  - Includes icons, drag&drop editor
- Pure.css (https://purecss.io/)
  - Small file size, 3.8KB
- Basscss (https://basscss.com/)
  - Even smaller, 3.39KB
  - Low-level (closer to raw CSS)

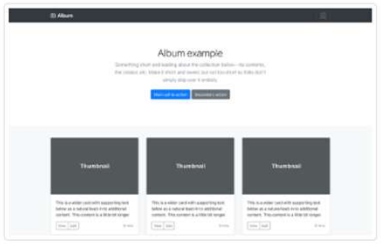




Grid frameworks
make development easier.
What are the downsides?

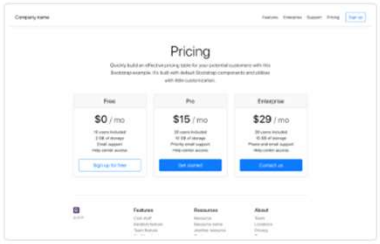
Opposition to Grid-based frameworks

Can lead to similar-looking webpages



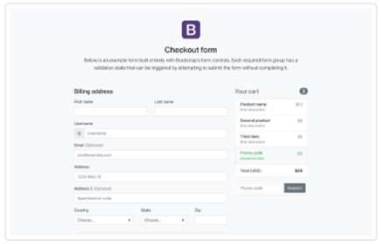
Album

Simple one-page template for photo galleries, portfolios, and more.



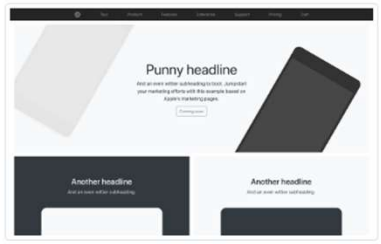
Pricing

Example pricing page built with Cards and featuring a custom header and footer.



Checkout

Custom checkout form showing our form components and their validation features.

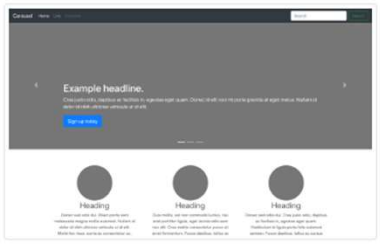


Product

Lean product-focused marketing page with extensive grid and image work.

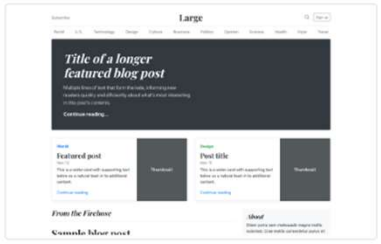
Cover

A one-page template for building simple and beautiful home pages.



Carousel

Customize the navbar and carousel, then add some new components.



Blog

Magazine like blog template with header, navigation, featured content.

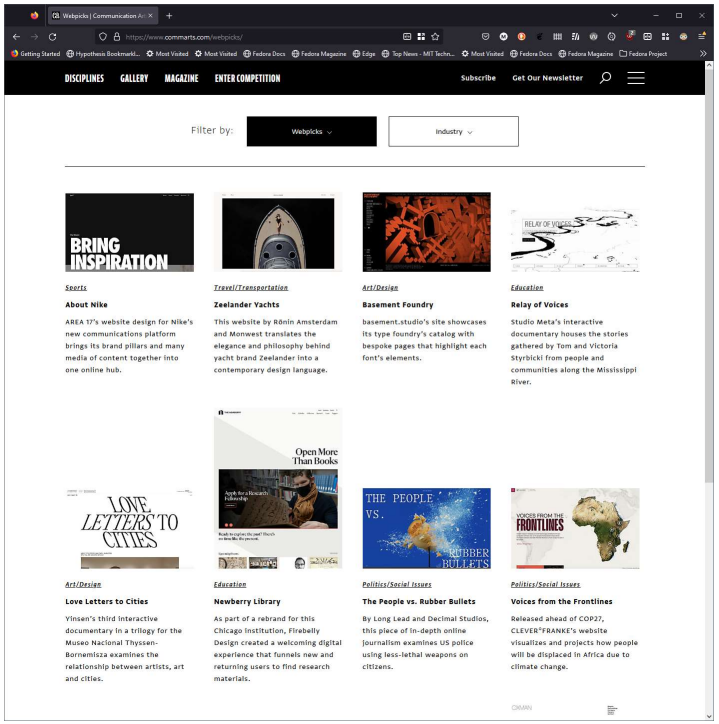
Dashboard

Basic admin dashboard shell with fixed sidebar and navbar.

Opposition to Grid-based frameworks

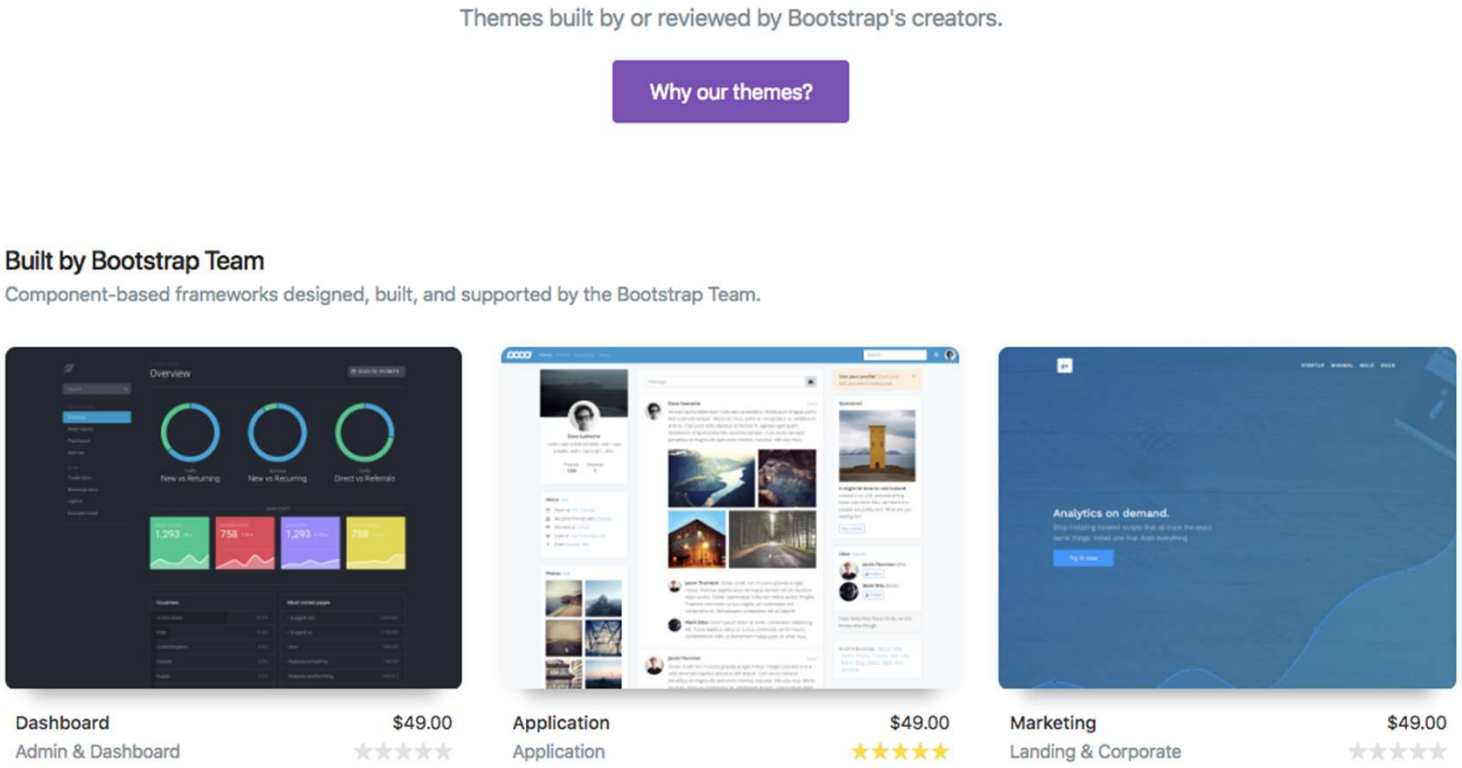
For inspiration take a look at Communication Arts Webpicks

https://www.commarts.com/webpicks/



Opposition to Grid-based frameworks

Can stifle creativity



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

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
- Explain the importance of accessible and semantically meaningful markup
- Generate markup which meets accessibility standards
- Describe how responsive and adaptive design differ
  and when you might prefer one or the other
- Explain the advantages and disadvantages of a grid-based layouts