Cascading Style Sheet (CSS

Junghoo Cho

cho@cs.ucla.edu

CSS (Cascading Style Sheet)

- A set of rules for specifying document formatting and preser
- Rule = Selector + Declaration block
  - Selector: tag, class, ID, *, …
  - Declaration block:
    - Enclosed inside `{ }`
    - List of “`property: value;`” pairs

CSS Example

```
p {                     /* p element */
    background-color: grey;
}
p.notes {               /* p element of notes class */
    background-color: yellow;
}
p .notes {              /* notes-class element that is a descendant 
    background-color: blue;
}
#text3372 {             /* id text3372 */
    background-color: green;
}
img[src$=".svg"] {  /* attr src ends with .svg */
    width: 20em;
}
```

Adding CSS Rules to Page

- CSS can be specified either
  - directly inside `<style> ... </style>`
  - in a separate file via `<link rel="stylesheet" href="example.css`
- Browsers has “default style values” for some tags
- To format a particular part, add `<div>` or `<span>` tags if need

Inheritance

- CSS can be specified in three places:
  1. Browser default
  2. User preference
  3. Web page
- If not set in any of the three places, an element *inherits its pa properties*

Cascading Rule

- *Cascading rule* dictates which CSS rule wins in case of conflic
  1. *Specificity*: more “specific” rule wins!
     - id > class > tag
     - more detailed specificity rule: https://www.w3.org/TR/css3-selectors/#specific
  2. Source order
     - if equal specificity, later rule wins
     - browser default < user preference < web page

CSS Custom Properties

- "*CSS variables*": Allows using a "logical name" to specify a val
- Example

```
body {
    --light-bg-color: white;  /* all descendants of body   */
    --dark-bg-color: brown;   /* inherits these properties */
}
code { background-color: var(--light-bg-color); }
p    { background-color: var(--dark-bg-color); }
```

  - Custom property names must start with `--`
  - Custom property values can be referenced with `var(...)` function
  - `var(--dark-bg-color, black)`: fallback values
    - Use `black` if custom property `--dark-bg-color` is not defined

Page Layout via CSS

- CSS can be used to specify the layout of a page
  - Example: http://www.nytimes.com
- Relevant CSS concepts and properties
  - CSS box model
  - `position` property
  - `block` vs `inline` element

CSS Box Model

- Every HTML element creates a virtual “box” around it
- Its dimension can be specified using the above properties
- Demo with example

`overflow` Property

- Specifies how to handle text overflow
  - `visible` (default): show overflow text
  - `hidden`: "clip" overflow text
  - `scroll`: always show scrollbar
  - `auto`: show scrollbar only if overflow

Positioning Element

- `top`, `right`, `bottom`, and `left` properties specify the element
- `position` property specifies how to interpret the “location”
  - `relative`: relative to is normal position
  - `absolute`: relative to its nearest *positioned* ancestor
  - `fixed`: relative to the “viewport” (viewable client area)
  - `static`: default. element is *unpositioned*
- Demo with example

Overlapping Elements

- `z-index`: specifies vertical location if elements overlap.
  - Higher z-index elements is placed on top of lower z-index elements

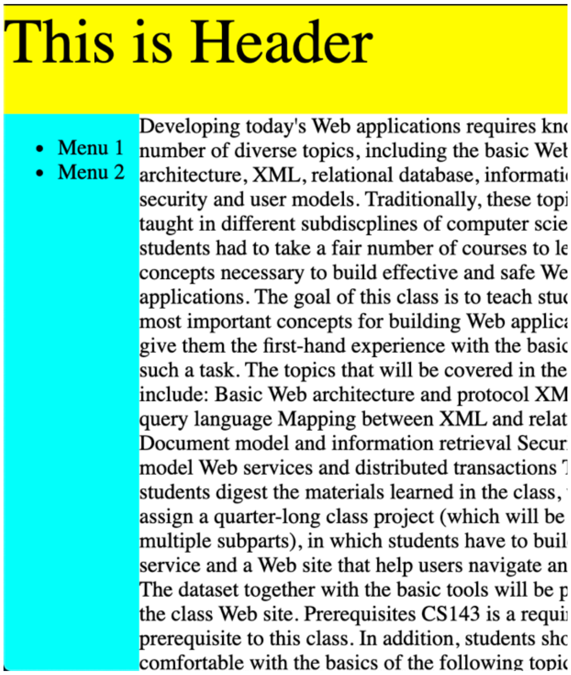
Block vs Inline Elements

- What we have seen are `block` elements
  - Block elements create a separate independent "block"
  - E.g., `<div>`, `<ul>`, `<p>`, …
- Inline elements are different
  - They do not create a separate block, but flow with surrounding text
  - `width`, `height`, `margin-top`, `margin-bottom` properties are ignored
  - E.g., `<span>`, `<a>`, …
- Demo with example

CSS Layout Example

Q: How can we create the following layout?

- Header stays at top
  - `width: 100%`
  - `height: 90px`
- Menu stays on left
  - `width: 100px`
  - `height` fills below header
- Content area fills rest
  - show scroll bar if overflow
- Code together here



This is Header

- Menu 1
- Menu 2

Developing today's Web applications requires kn
number of diverse topics, including the basic We
architecture, XML, relational database, informati
security and user models. Traditionally, these topi
taught in different subdiscplines of computer scie
students had to take a fair number of courses to le
concepts necessary to build effective and safe We
applications. The goal of this class is to teach stu
most important concepts for building Web applic
give them the first-hand experience with the basic
such a task. The topics that will be covered in the
include: Basic Web architecture and protocol XM
query language Mapping between XML and relat
Document model and information retrieval Secur
model Web services and distributed transactions
students digest the materials learned in the class,
assign a quarter-long class project (which will be
multiple subparts), in which students have to buil
service and a Web site that help users navigate an
The dataset together with the basic tools will be p
the class Web site. Prerequisites CS143 is a requi
prerequisite to this class. In addition, students sho
comfortable with the basics of the following topic

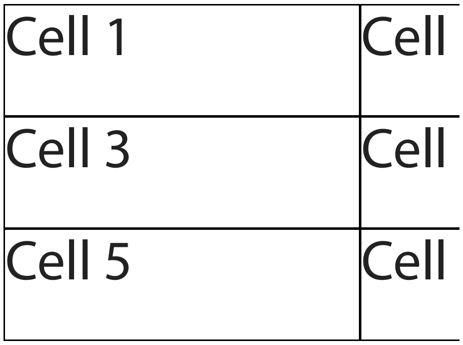
CSS Grid

- CSS grid makes it easy to place elements in a tabular arrange

  - A *grid container* (created by "`display: grid;`" property) includes *gr*
  - Grid dimension is specified using `grid-template-rows` and `grid-te` `columns` properties

CSS Grid Example

```
#container {
    display: grid;
    grid-template-rows: 2em 2em 2em;
        /* height of each row */
    grid-template-columns: 100pt 100pt;
        /* width of each column */
}
<div id="container">
    <div>Cell 1</div>
    <div>Cell 2</div>
    <div>Cell 3</div>
    <div>Cell 4</div>
    <div>Cell 5</div>
    <div>Cell 6</div>
</div>
```

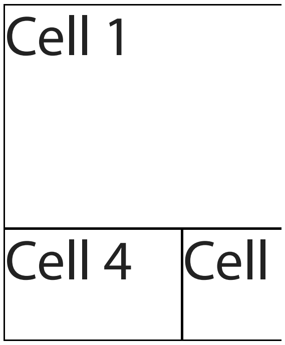


Cell 1	Cell
Cell 3	Cell
Cell 5	Cell

CSS Grid Example: Spanning

```
#container {
    display: grid;
    grid-template-rows: 2em 2em 2em;
    grid-template-columns: 100pt 100pt 100pt;
}
#c1 {
    grid-column-start: 1;
    grid-column-end: 3; /* spans columns 1-2 */
    grid-row-start: span 2; /* spans 2 rows */
}
```

```
<div id="container">
    <div id="c1">Cell 1</div><div>Cell 2</div>
    <div>Cell 3</div><div>Cell 4</div>
    <div>Cell 5</div><div>Cell 6</div>
</div>
```



Cell 1

Cell 4 Cell

Fixed vs Fluid Layout

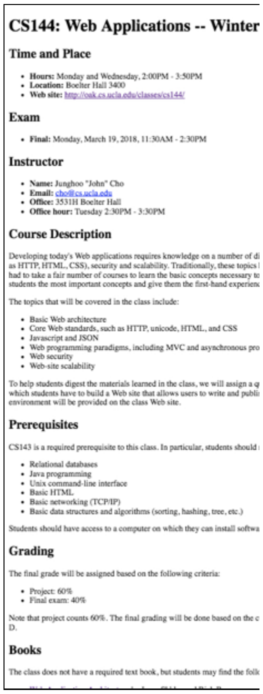
- *Fixed layout*
  - Elements have fixed width
  - Resizing the window does not change their sizes or arrangements
- *Fluid layout*
  - Elements use “percentage” of page width
  - Elements dynamically resize and rearrange to fit window width

Responsive Web Design (RWD)

- Web is accessed from a wide range of devices
  - Phone, tablet, desktop, …
- Page design should dynamically adapt to screen size
- Responsive Web design example
- General Rules for RWD
  - Do *NOT* force users to scroll horizontally (Why?)
  - Do *NOT* use fixed-width elements (Why?)
  - Do *NOT* force users to zoom in and out to read text (Why?)
  - Use **CSS media queries** to apply different styling depending on the

Viewport (1)

- In 2007, Web pages were designed for desktop
  - Large screen size
  - To display the entire page, iPhone used “fake” display width (~980px) after zooming out
- `viewport` meta tag
  - User’s visible area of a web page
  - width: viewport width
  - initial-scale: initial “zoom level”
  - iPhone adopted large value as the default `viewport` value



CS144: Web Applications -- Winter

Time and Place

- **Hours:** Monday and Wednesday, 2:00PM - 3:50PM
- **Location:** Boelter Hall 3400
- **Web site:** http://oak.cs.ucla.edu/classes/cs144/

Exam

- **Final:** Monday, March 19, 2018, 11:30AM - 2:30PM

Instructor

- **Name:** Junghoo "John" Cho
- **Email:** cho@cs.ucla.edu
- **Office:** 3531H Boelter Hall
- **Office hour:** Tuesday 2:30PM - 3:30PM

Course Description

Developing today's Web applications requires knowledge on a number of di
as HTTP, HTML, CSS), security and scalability. Traditionally, these topics
had to take a fair number of courses to learn the basic concepts necessary to
students the most important concepts and give them the first-hand experienc

The topics that will be covered in the class include:

- Basic Web architecture
- Core Web standards, such as HTTP, unicode, HTML, and CSS
- Javascript and JSON
- Web programming paradigms, including MVC and asynchronous pro
- Web security
- Web-site scalability

To help students digest the materials learned in the class, we will assign a q
which students have to build a Web site that allows users to write and publi
environment will be provided on the class Web site.

Prerequisites

CS143 is a required prerequisite to this class. In particular, students should

- Relational databases
- Java programming
- Unix command-line interface
- Basic HTML
- Basic networking (TCP/IP)
- Basic data structures and algorithms (sorting, hashing, tree, etc.)

Students should have access to a computer on which they can install softwa

Grading

The final grade will be assigned based on the following criteria:

- Project: 60%
- Final exam: 40%

Note that project counts 60%. The final grading will be done based on the c
D.

Books

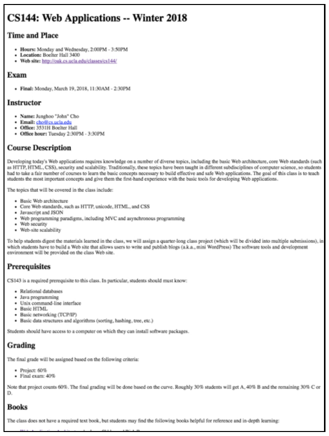
The class does not have a required text book, but students may find the foll

Viewport (2)

- But today, we design Web pages specifically for mobile devic
- Override the default setting with `viewport` meta tag

```
<meta name="viewport" content="width=device-width, initial-sca
```

  - Do not use the default viewport size and zoom level!



CS144: Web Applications -- Winter 2018

Time and Place

- **Hours:** Monday and Wednesday, 2:00PM - 3:50PM
- **Location:** Boelter Hall 3400
- **Web site:** http://oak.cs.ucla.edu/classes/cs144/

Exam

- **Final:** Monday, March 19, 2018, 11:30AM - 2:30PM

Instructor

- **Name:** Junghoo "John" Cho
- **Email:** cho@cs.ucla.edu
- **Office:** 3531H Boelter Hall
- **Office hour:** Tuesday 2:30PM - 3:30PM

Course Description

Developing today's Web applications requires knowledge on a number of diverse topics, including the basic Web architecture, core Web standards (such as HTTP, HTML, CSS), security and scalability. Traditionally, these topics have been taught in different subdisciplines of computer science, so students had to take a fair number of courses to learn the basic concepts necessary to build effective and safe Web applications. The goal of this class is to teach students the most important concepts and give them the first-hand experience with the basic tools for developing Web applications.

The topics that will be covered in the class include:

- Basic Web architecture
- Core Web standards, such as HTTP, unicode, HTML, and CSS
- Javascript and JSON
- Web programming paradigms, including MVC and asynchronous programming
- Web security
- Web-site scalability

To help students digest the materials learned in the class, we will assign a quarter-long class project (which will be divided into multiple submissions), in which students have to build a Web site that allows users to write and publish blogs (a.k.a., mini WordPress) The software tools and development environment will be provided on the class Web site.

Prerequisites

CS143 is a required prerequisite to this class. In particular, students should must know:

- Relational databases
- Java programming
- Unix command-line interface
- Basic HTML
- Basic networking (TCP/IP)
- Basic data structures and algorithms (sorting, hashing, tree, etc.)

Students should have access to a computer on which they can install software packages.

Grading

The final grade will be assigned based on the following criteria:

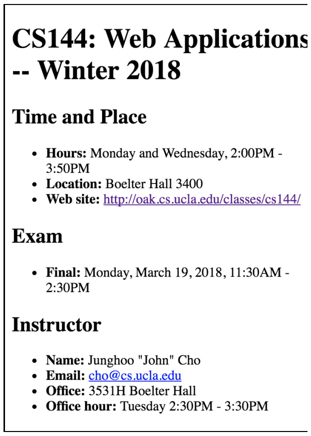
- Project: 60%
- Final exam: 40%

Note that project counts 60%. The final grading will be done based on the curve. Roughly 30% students will get A, 40% B and the remaining 30% C or D.

Books

The class does not have a required text book, but students may find the following books helpful for reference and in-depth learning:

Before



CS144: Web Applications -- Winter 2018

Time and Place

- **Hours:** Monday and Wednesday, 2:00PM - 3:50PM
- **Location:** Boelter Hall 3400
- **Web site:** http://oak.cs.ucla.edu/classes/cs144/

Exam

- **Final:** Monday, March 19, 2018, 11:30AM - 2:30PM

Instructor

- **Name:** Junghoo "John" Cho
- **Email:** cho@cs.ucla.edu
- **Office:** 3531H Boelter Hall
- **Office hour:** Tuesday 2:30PM - 3:30PM

After

Media Queries

- Mechanism to apply custom CSS rules for specific devices
- Example

```
@media (max-width: 800px) {
    /* CSS rules */
}
```

- Syntax: `@media condition { /* CSS rules */ }`
  - Enclosed rules are applied only if `condition` is true
  - `condition` can be a complex boolean condition

Media Query Conditions

- Media types
  - `screen`, `print`, `speech`, and `all` (default)
- Media features
  - `orientation`, `min-width`, `max-width`, `min-height`, `max-height`, `reso`
- Boolean operators
  - `,`=OR, `and`=AND, `not`=NOT
  - Precedence: `not` > `and` > `,`

Media Query Example

- Q: When does the following rule apply?

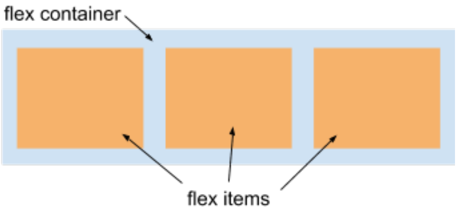
```
@media screen, (orientation: landscape) {
    /* ... */
}
```

CSS Flexbox

- “Flexible box”
  - New addition to CSS to enable flexible layout of elements
  - Elements are dynamically resized or rearranged based on available s

Flex Container and Flex Item

- Flexbox consists of a flex container and flex items
  - A *flex container* (created by "`display: flex;`") includes many *flex ite*



  - All children of a flex container become flex items

Flexbox: Changing Size

- By default, flex items change its size to fit available space
- The exact resize behavior can be specified
  - `flex-basis`: default size of an element
  - `flex-grow`: when there is remaining space, extra space is divided an items according to their flex-grow factor
  - `flex-shrink`: when there is space shortage, spaces are taken away f items by the factor of flex-shink * flex-basis

Flexbox: Rearranging Items

- Flex items in a container can be dynamically rearranged and based on available space
  - `flex-wrap: wrap`
- Wrapping direction
  - Horizontally: “`flex-direction: row;`” or
  - Vertically: “`flex-direction: column;`”

Flexbox Example

- In the earlier demo

```
#menu-container { display: flex; }
#nav    { flex: 1 1 200px; }
          /* flex-grow flex-shrink flex-basis */
#main   { flex: 2 1 400px; }
          /* Note flex-shrink 1! (not 2) */
#aside { flex: 1 1 200px; }

#large-box { display: flex; flex-wrap: wrap; }
```

Animation

- Q: How can we create animation effects on a page?
  - e.g., scrolling news tickers, flying boxes, …
- Two approaches
  - JavaScript
  - CSS animation

JavaScript Animation

- Basic idea: Periodically update CSS property of an element
- Calling a function periodically
  - `setInterval(callback, interval)`: invoke `callback` every `interv` milliseconds
- `style` property: CSS properties of an element
  - Example: `body.style.background`

Animation Demo

- Ticker example

```
let loc = 0;
let ticker = document.getElementById("ticker");
let timer = setInterval(tickerSlide, 100);

function tickerSlide() {
    loc += 10;
    ticker.style.left = String(loc) + "px";

    if (loc > 300) clearInterval(timer);
}
```

- Q: Why does the text move?
- Q: Why does it stop moving?

Animation: Another Example

```
<body>
    <div id="box" style="border: solid 5px black;"></div>
</body>
<script type="text/javascript">
let box  = document.getElementById("box");
let size = 0;
let timer = setInterval(callback, 100);

function callback() {
    box.style.width = String(size) + "px";
    box.style.height = String(size) + "px";
    size = (size + 10) % 200;
}
</script>
```

- Q: What will this do?

Relevant API

- `setInterval(callback, interval, param1, ...)`
    - Invoke `callback(param1, ...)` *repeatedly* every `interval` millisecc
- `setTimeout(callback, interval, param1, ...)`
    - Invoke `callback(param1, ...)` *once* after `interval` milliseconds
- `clearTimeout(timer)` or `clearInterval(timer)`
    - Clear existing `timer`
    - `timer`: return value from `setTimeout()` or `setInterval()`

CSS Animation

- Two possibilities
  - Simple: `transition` property
  - Complex: `@keyframes` rule

CSS `transition` property

- Creates “transition effect” when an element’s CSS changes
  - Makes the changes “gradual”
  - Example: `transition: height 1s;`
    - When the element’s height changes, “animate” the change over 1s

CSS transition Example

- Example
- Code

```
<style>
    div {
        height: 1em;
        transition: height 1s;
    }
    div:hover {
        height: 10em;
    }
</style>
<body><div>CSS Transition</div></body>
```

CSS @keyframes Rule

- @keyframes allows specifying the “keyframes” in animation

```
@keyframes css3animation {
    0%    { background: red; }
    50%   { background: yellow; }
    100%  { background: green; }
}
```

  - Update background property gradually using provided keyframes

- Apply a @keyframe rule with animation property

```
#header1 { animation: css3animation 3s; }
```

  - Apply css3animation keyframe rule to #header1 over 3 seconds

- Demo

Relevant CSS Properties

- `animation-delay`
  - When the animation will start
- `animation-play-state: paused|running`
  - Whether the animation is running or paused
- `animation-iteration-count`
  - # of times animation is played (or `infinite`)
- Complex shape transformation: `transform`
  - e.g., `transform: rotate(45deg) scale(1.5);`

CSS Preprocessor

- Creating CSS rules manually for every element or class is rep
- “CSS preprocessors” generate CSS rules from a higher-level specification
  - e.g., SASS, LESS, Stylus, …

CSS Frameworks

- Creating all UI elements with basic HTML and CSS from scrat
- Many CSS “libraries” and “frameworks” exist that help creatin interactive and responsive user interfaces
  - Bootstrap: JavaScript + CSS UI framework
  - Bulma

What We Learned

- CSS custom property
- Page layout via CSS
  - CSS box model
  - `position`
  - CSS grid
- Responsive Web Design (RWD)
  - fixed vs fluid layout
  - `viewport`, media query
  - CSS flexbox
- JavaScript animation: `setTimeout()`
- CSS transition and animation

References

- CSS standard
- CSS Flexbox
- CSS Transitions
- CSS Animations
- Flexbox tutorial
- Bootstrap