CS144: Cascading Style Sheet (CSS)

Basic CSS

- A set of rules for specifying document formatting and presentation
- rule = selector + declaration block
- Example: http://oak.cs.ucla.edu/classes/cs144/examples/css.html

```css
/* selector: tag, class, ID, or * for all */
body { /* declaration block: list of "property: value;" pairs */
    font-family: "Arial";
}

h1 {
    font-size: 40pt;
}

.code { /* "." indicates class name */
    font-family: monospace;
    white-space: pre;
    background-color: rgb(220, 220, 220);
    border: 1px solid black;
}

#warning1 { /* "#" indicates ID name */
    color: rgb(255, 0, 0);
}

:hover { /* : indicates "pseudo class" selector */
    color: #0000ff;
    /* when mouse hovers over it, change color */
}

CSS can be specified directly inside `<style>` tag or in a separate page through `<link>` tag
– <style> ... </style>
– <link rel="stylesheet" href="example.css">
– Browsers uses its “browser default style” to format some tags
  * HTML recommendation: http://www.w3.org/TR/CSS2/sample.html
• Show the body of the page and explain how I want to format it
  – Explain CSS rules on how to interpret them
  – Format warning text by adding id attribute
  – Format code using <div>
  – Format only part of warning text by adding <span>
  – Remove comments for hover and see what happens
    * Modify to change background only for h1
• Use <div> or <span> tags to specify the part to apply a CSS rule
  – <span> for inline element (embedded in flowing text)
  – <div>: for a block element (starts a new line and creates a “block”)

Cascading, Specificity, Inheritance

• Cascading rule dictates which CSS rule wins in case of conflict
  1. Specificity: more “specific” rule wins!
    – id > class > tag
    – more detailed specificity rule: https://www.w3.org/TR/css3-selectors/
      #specificity
  2. Source order
    – if equal specificity, later rule wins
• Inheritance
  – CSS can be specified in three places:
    * web page, user preference, browser default
  – If the CSS property of an element is not set in any of the three places (including browser default), it inherits its parent’s property value

Advanced CSS selectors

```css
[enabled] { /* attribute selector. has attribute named "enabled" */ }
```

Junghoo "John" Cho (cho@cs.ucla.edu) 2
CSS Layout

- CSS can be used to specify the layout of a page
  - Show a few example pages like http://www.nytimes.com
- Relevant CSS concepts and properties
  - Inline vs block element
  - CSS box model
    - Every HTML element creates a virtual “box” around it
  - CSS properties related to the location, size, and margin of this box
* border, margin, padding, width, and height
* position: relative, absolute, and fixed

**Inline vs Block**

- Block elements create a new separate “block” from surrounding text
  - E.g., `<div>`, `<ul>`, `<p>`, ...
- Inline elements are embedded inside surrounding text
  - E.g., `<span>`, `<a>`, ...
- Default element type can be changed using CSS `display` property
  - `display: block;` (or `inline`)
- Example:
  - [http://oak.cs.ucla.edu/classes/cs144/examples/css-box.html](http://oak.cs.ucla.edu/classes/cs144/examples/css-box.html)
  - Explain the boxes for div and span elements
  - Show the changes of span-element box when more text is added

**CSS box model**

![CSS Box Model](image)

**Figure 1:** CSS Box Model

- The size and space around a CSS box can be specified using the above CSS properties
- Example: add the following properties to `#block` element and see what happens
• Inline elements do not create a separate block
  – They ignore width, height, margin-top, and margin-bottom properties
  – Example: add the same properties to #inline element and see what happens

```css
margin: 1em 2em 3em 4em;
padding: 10px;
width: 60%;
height: 300px;
```

• overflow property: how to deal in case of text overflow
  – visible (default): show overflow text
  – hidden: “clip” overflow text
  – scroll: always show scrollbar
  – auto: show scrollbar only if overflow

Positioning elements

• CSS `top`, `right`, `bottom`, and `left` properties specify the “location” of an element
• CSS `position` property specifies how to interpret the “location”
  – relative: positioned relative to is normal position
  – absolute: positioned relative to its nearest `positioned` ancestor
  – fixed: positioned relative to the “viewport” (viewable client area)
  – static: default value. Element is unpositioned

• Example: http://oak.cs.ucla.edu/classes/cs144/examples/css-position.html
  – Add the following to .box class

```css
position: relative; /* fixed, absolute */
```
• Overlapping elements and z-index
  – z-index property specifies vertical location if elements overlap.
  – Higher z-index elements is placed on top of lower z-index elements.

CSS layout example

• Q: How can we specify the layout of http://oak.cs.ucla.edu/classes/cs144/examples/css-layout.html?
  – Header always stays at the top with width=100% and height 90px
  – Menu always stays on the left with width=80px and height fills the screen below header
  – Content area is stretched to fill the screen and scrollable if overflows
  – Note:
    * Unless an element is positioned as absolute or fixed, percent height does not stretch.
    * Use calc(100% - 100px) to “calculate” length. Space needed around the operator.
    * Cross-browser compatibility is difficult due to different default margins of body, etc.

Floating Element

• float property: make the element “float” and wrap the following text and elements around it
  – Left and right are allowed, but no center
    * float: left; float to the left
  – We can center an element by setting display: block; and margin: auto;
Responsive Web Design (RWD)

- Responsive Web Design: design Web pages, so that it is easy to see on a wide range of devices
  - phone, tablet, desktop, ...

- Fixed vs Fluid layout
  - **Fixed**: elements have fixed width. Resizing the window does not change the appearance of the page
  - **Fluid**: elements uses “percentage” of page width. Elements dynamically resize to fit window width

- General rules
  - Do NOT force users to scroll horizontally (Why?)
  - Do NOT use fixed-width elements (Why?)
  - Use CSS media queries to apply different styling depending on the screen size

Viewport

- When iPhone was first introduced, web pages were designed for desktop machines
  - To display the pages, it used “fake” display width (~ 980px) with zoom
• **viewport**: user’s visible area of a web page
  - width: viewport width
  - initial-scale: initial “zoom level”

• Always add a viewport meta tag
  - `<meta name="viewport" content="width=device-width, initial-scale=1">`
  - Otherwise, default viewport width (~ 980px) is used, which can make text too small

**Media queries**

• CSS rules enclosed in `@media condition { ... }`
  - `condition` can be a complex boolean condition that specifies when the rules will be applied
  - This allows applying different css rules depending on device property

• Example

```css
@media (max-width: 800px) {
  /* CSS rules */
}
```
  - Apply the CSS rules only if the page is displayed on screen and viewport width is 800px or less

• Conditions that can be used in media queries
  - Media types
    * screen, print, speech, and all (default)
  - Media features
    * orientation, min-width, max-width, min-height, max-height, resolution, ...
  - Boolean operators
    * `,` = OR, `and` = AND, `not` = NOT
    * Precedence: `not` > `and` > `,`

• Q: When does the following rule apply?

```css
@media screen, (orientation: portrait) {
  /* ... */
}
```
CSS Flexbox

- “Flexible box”
  - New addition to CSS to help create flexible layout of elements
  - Supports elements whose size is dynamically adjusted based on screen size
  - Example: http://oak.cs.ucla.edu/classes/cs144/examples/css-flexbox.html
- Flexbox consists of a flex container and flex items
  - A flex container includes many flex items.

![CSS flexbox diagram]

**Figure 2:** CSS flexbox

- An element becomes a flex container by setting `display: flex;`
- All its children become flex items
- Flex items are arranged horizontally (`flex-direction: row;`) or vertically (`flex-direction: column;`)
- Dynamic size adjustment
  - Flex container changes its dimension to fill all available space
  - Flex items can also adjust its size dynamically depending on available space
    * `flex-basis`: default size of an element
    * `flex-grow`: when there is remaining space, extra space is divided among flex items according to their flex-grow factor
    * `flex-shrink`: when there is space shortage, spaces are taken away from flex items by the factor of flex-shink*flex-basis
- Example code at http://oak.cs.ucla.edu/classes/cs144/examples/css-flexbox.html

```html
<!DOCTYPE html>
```
CSS Flexbox

CSS Preprocessor

- Many “CSS preprocessors” exist that generate CSS rules from a higher-level specification
  - e.g., SASS, LESS, Stylus, …

UI Frameworks

- Many “libraries” and “frameworks” exist that help us creating interactive and responsive user interfaces
  - jQuery: JavaScript library for dynamic UI
  - Bootstrap: JavaScript + CSS UI framework
References

- CSS standard: http://www.w3.org/Style/CSS/current-work
- More detailed explanation on flexbox: https://css-tricks.com/snippets/css/a-guide-to-flexbox/
- Bootstrap: https://getbootstrap.com/