Every CSS start with selector

Every CSS property creates one particular box

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

```
/* 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>`
  - emove 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

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

```
    color: red;  /* red color for elements with enabled attribute
       */
}
[target="_blank"] { /* attribute "target" has the value "_blank" */
    color: blue; /* blue color for elements with target="_blank"
       attribute */
}
div, p { /* multiple selectors can be separated by commas */
    background-color: grey;
}
div p { /* p is a descendent of div */
    background-color: yellow;
}
div > p { /* p is a direct child of div */
    background-color: green;
}
div + p { /* p is the adjacent sibling of (i.e., immediately
   follows) div */
    background-color: blue;
}
div ~ p { /* p is a general sibling of div */
    background-color: red;
}
::first-letter { /* "pseudo element" selector */
    font-size: 2em  /* use font size 2em for the first letter */
}
```

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
  - Explain the boxes for div and span elements
  - Show the changes of span-element box when we more text is added

CSS box model

TOP, RIGHT, BOTTOM, LEFT <-- size

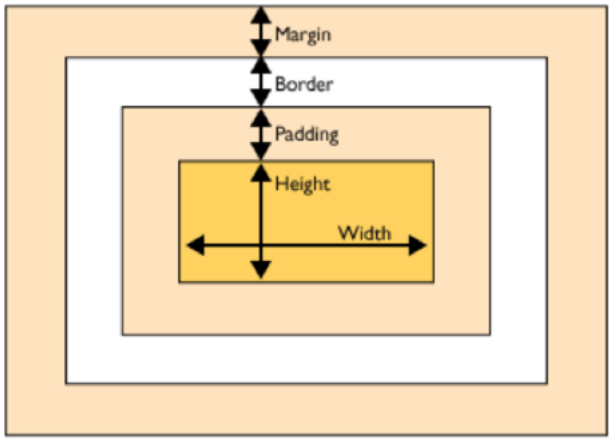


Figure 1: CSS Box Model

- 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

```
margin: 1em 2em 3em 4em;
        /* top, right, bottom, left */
        /* if any value is missing, use a "symmetric" value */
        /* 1em = 16px (~ width of M) */
padding: 10px;
width: 60%;
height: 300px;
```

- 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

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

- `overflow` property: how to deal in case of text overflow //Show the scroll bar if 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 From the original (default) location
  - `absolute`: positioned relative to its nearest *positioned* ancestor
  - `fixed`: positioned relative to the "viewport" (viewable client area) Relative to screen
  - `static`: default value. Element is *unpositioned*
- Example: http://oak.cs.ucla.edu/classes/cs144/examples/css-position.html
  - Add the following to .box class

```
position: relative; /* fixed, absolute */
```

```
top: 1em;
left: 1em;
```

- 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)

Three things need for RWD 1) Viewport 2)media query 3) CSS flexbox (flexible box)

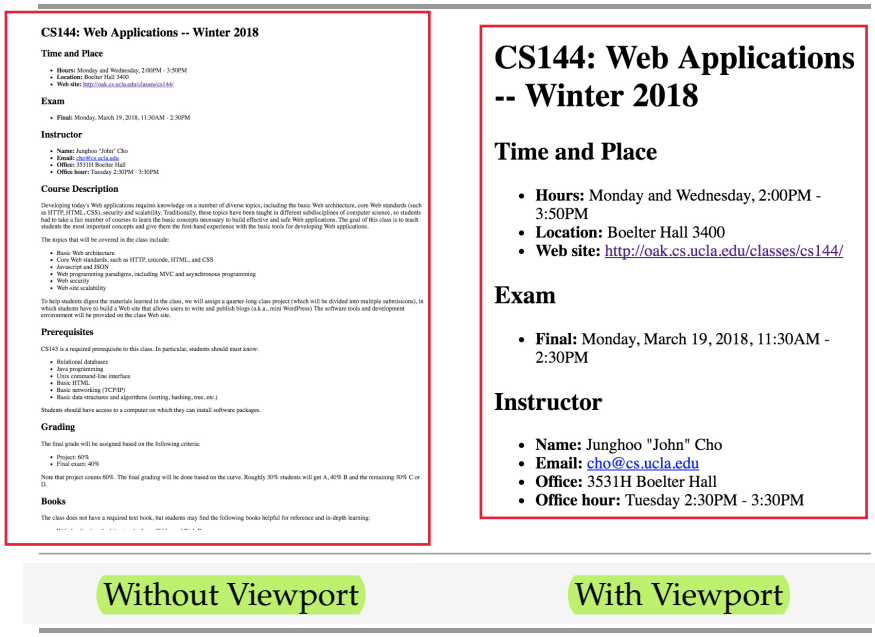
- Responsive Web Design: design Web pages, so that it is easy to see on a wide range of of devices

  - 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

1. **Viewport** => it represents actual size of the device. It sets actual size of the device screen to the webpage

Always add a viewport meta tag

- `<meta name="viewport"content="width=device-width, initial-scale=1">`
- `viewport`: user's visible area of a web page
  - width: viewport width
  - initial-scale: initial "zoom level"
- Always add a viewport meta tag
  - Otherwise, default viewport width (~ 980px) is used, which can make text too small



2. **Media queries** => declare the size of particular device screen size

syntax :: @media (max-width:80px) { P { font-size : 14pt; } }

- Media query allows applying different css rules depending on device property
- Example

2 types 1) media type: screen, print, speech, all
2)media feature:

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

  - Apply the CSS rules only if the page is displayed on screen and viewport width is 800px or less
  - *Breakpoint*: the viewport-width boundary for applying different rules. 800px in this example
- Possible conditions 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?

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

3. CSS Flexbox

It helps to shrink or expand display with the size of the browser

- "Flexible box"
  - New addition to CSS to help create flexible layout of elements
  - A *flex container* (`display: flex;`) includes many *flex items*.

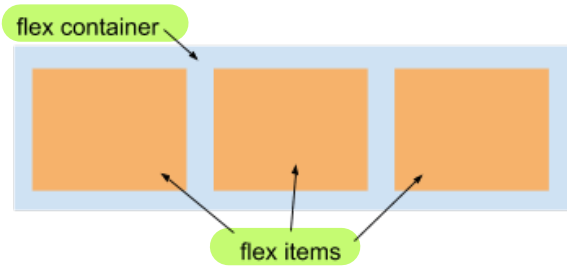


Figure 2: CSS flexbox

flex growth-> when display grow then divide the space based on the container and hence ll be larger
flex shrink-> when display shrinks, it give up space of container

  - Flex items are arranged horizontally (`flex-direction: row;`) or vertically (`flex-direction: column;`)

Growth factor is inverse of shrink factor
eg GF is 1 2 1
SF is 2 1 2

  - Flex items are dynamically resized as the container size changes
    * grow/shrink factors can be specified (`flex-growth`, `flex-shrink`, and `flex-basis`). ← Important property
  - Example: http://oak.cs.ucla.edu/classes/cs144/examples/css-flexbox.html

```
<!DOCTYPE html>
<html>
<head>
<title>CSS flexbox</title>
<style>
    #container {
```

```
        display: flex;
        flex-direction: row;
    }
    nav   { flex: 1 2 100px; }
            /* flex-growth flex-shrink flex-basis */
    #main { flex: 2 1 200px; }
    aside { flex: 1 2 100px; }

    * { text-align: center; border: 1px solid black; }
</style>
</head>
<body>
    <header>Header</header>
    <div id="container">
        <nav>Navigation</nav>
        <div id="main">Main Text</div>
        <aside>Aside</aside>
    </div>
    <footer>Footer</footer>
</body>
</html>
```

CSS Preprocessor

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

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/