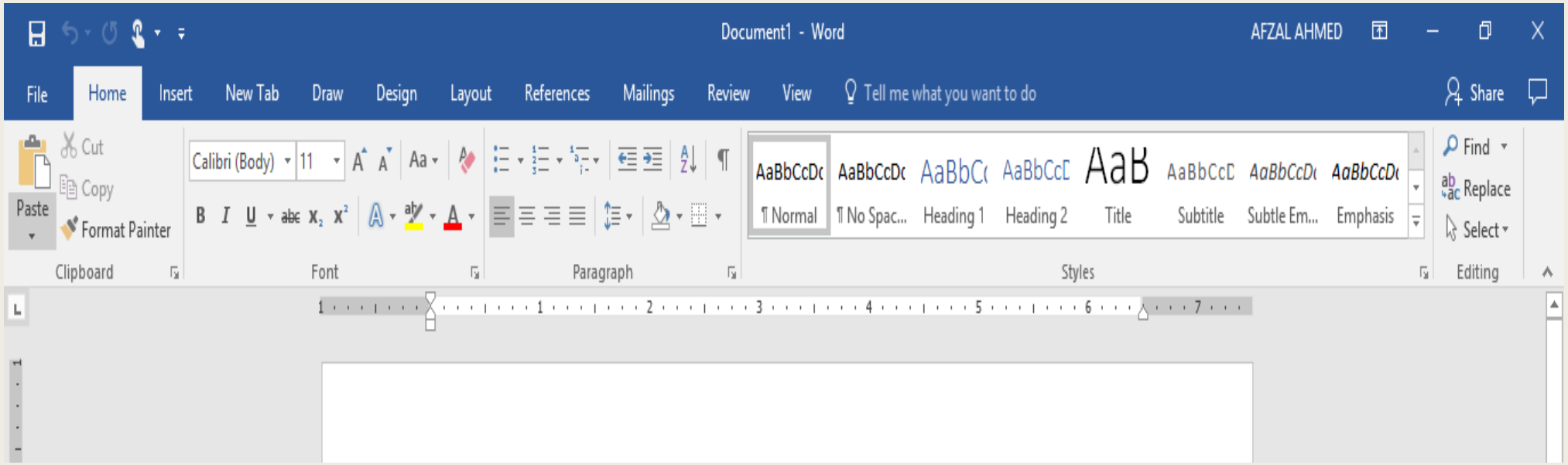
HUMAN COMPUTER INTERACTION

Lecture 7: Layout

Hall of Fame or Shame?



Today’s Topic

- CSS
- Automatic layout
- Constraints

Cascading Style Sheets (CSS)

- Key Idea: separate the structure of UI (View Tree) from details of presentation
  - *HTML is structure, CSS is presentation*
- Two ways to use CSS
  - *As an attribute of a particular HTML element*

    <button style="font-weight: bold;"> Cut </button>
  - *As style sheet defining style rules for many HTML elements at once*

    <style>

    Button {font-weight:bold;}

    </style>

CSS Selectors

- Each rule in a style sheet has a selector pattern that matches a set of HTML elements

Tag name

```
button {font-weight: bold;}
```

ID

```
#main {background-color:
rgb(100%,100%,100%);}
```

Class attribute

```
.toolbarButton {font-size: 12pt;}
```

Element paths

```
#toolbar button{display: hidden;}
```

Cascading and inheritance

- If multiple rules apply to the same element, rules are automatically combined with cascading precedence
  - *Source: browser defaults <webpage<user overrides*

    Browser says *a{ text-decoration:underline;}*

    Webpage says *a{text-decoration:none;}*

    User says: *a {text-decoration:underline;}*
  - *Rule specify: general selectors<specific selectors*

    Button {font-size: 14pt;}

    .toolbarButton{font-size:14pt;}
- Styles can also be inherited from element's parent
  - *This is the default for simple styles like font, color, and text properties*

    body {font-size:12pt;}

Declarative Styles vs. procedural Styles

- CSS

  //found in a <style> element

  Button{font-size: 12pt; font-weight: bold;}

- Jquery

  //in a <script> element

  $(“button”).css(“font-size”,”12pt”).css(“font-weight”, “bold”);

Automatic Layout

- Layout determines the sizes and positions of components on the screen
  - *Also called geometry in some toolkits*
- Declarative layout
  - *CSS styles*
- Procedural layout
  - *Write javascript code to compute positions and sizes*

Reasons to do Automatic Layout

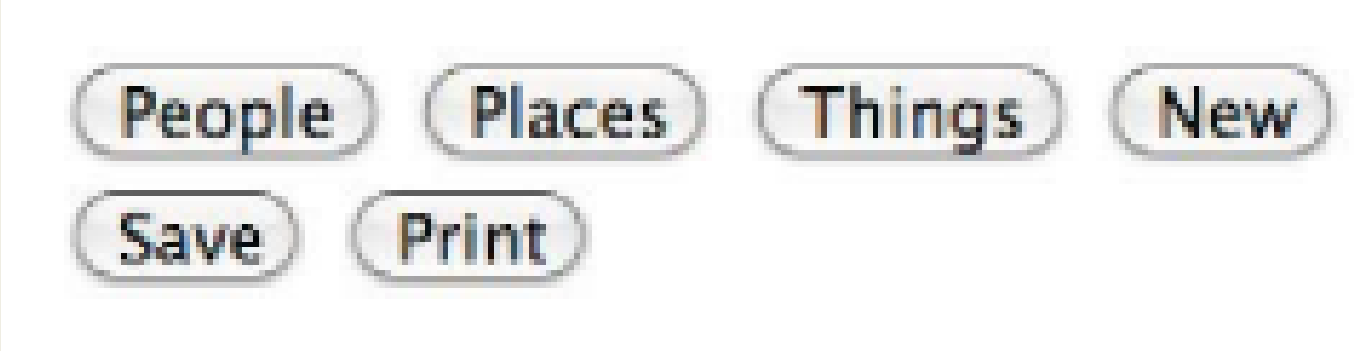
- Higher Level programming
  - *Shorter, Simpler code*
- Adapts to change
  - *Window size*
  - *Font size*
  - *Widget set (or theme or skin)*
  - *Labels (internationalization)*
  - *Adding or removing nodes*

Flow Layout

- Left-to-right, automatically-wrapping
- CSS calls this “inline” layout

  display:inline
- Many elements use inline layout by default

```
<button>People</button>
<button>Places</button>
<button>Things</button>
<button>New</button>
<button>Save</button>
<button>Print</button>
```



Block Layout

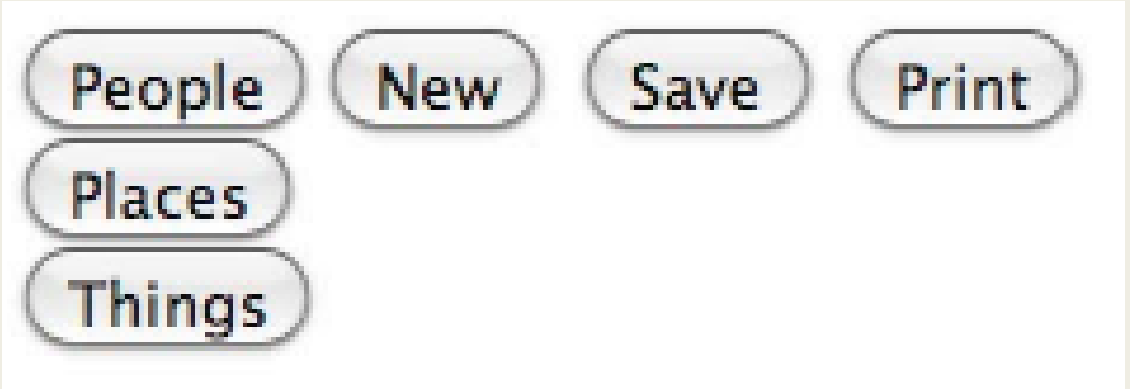
- Blocks are laid out vertically
  - *display:block*
  - *divs default to block layout*
- Inline blocks are laid out in flow
  - *display:inline-block*

```
<div><button>People</button>
<button>Places</button>
<button>Things</button></div>
<div><button>New</button>
<button>Save</button>
<button>Print</button></div>
```

Float Layout

- Float pushes a block to left or right edge

```
<Style>
.navbar{float: left;}
.navbar button {display: block;}
</style>
<div class="navbar"><button>People</button>
<button>Places</button>
<button>Things</button></div>
<div><button>New</button>
<button>Save</button>
<button>Print</button></div>
```

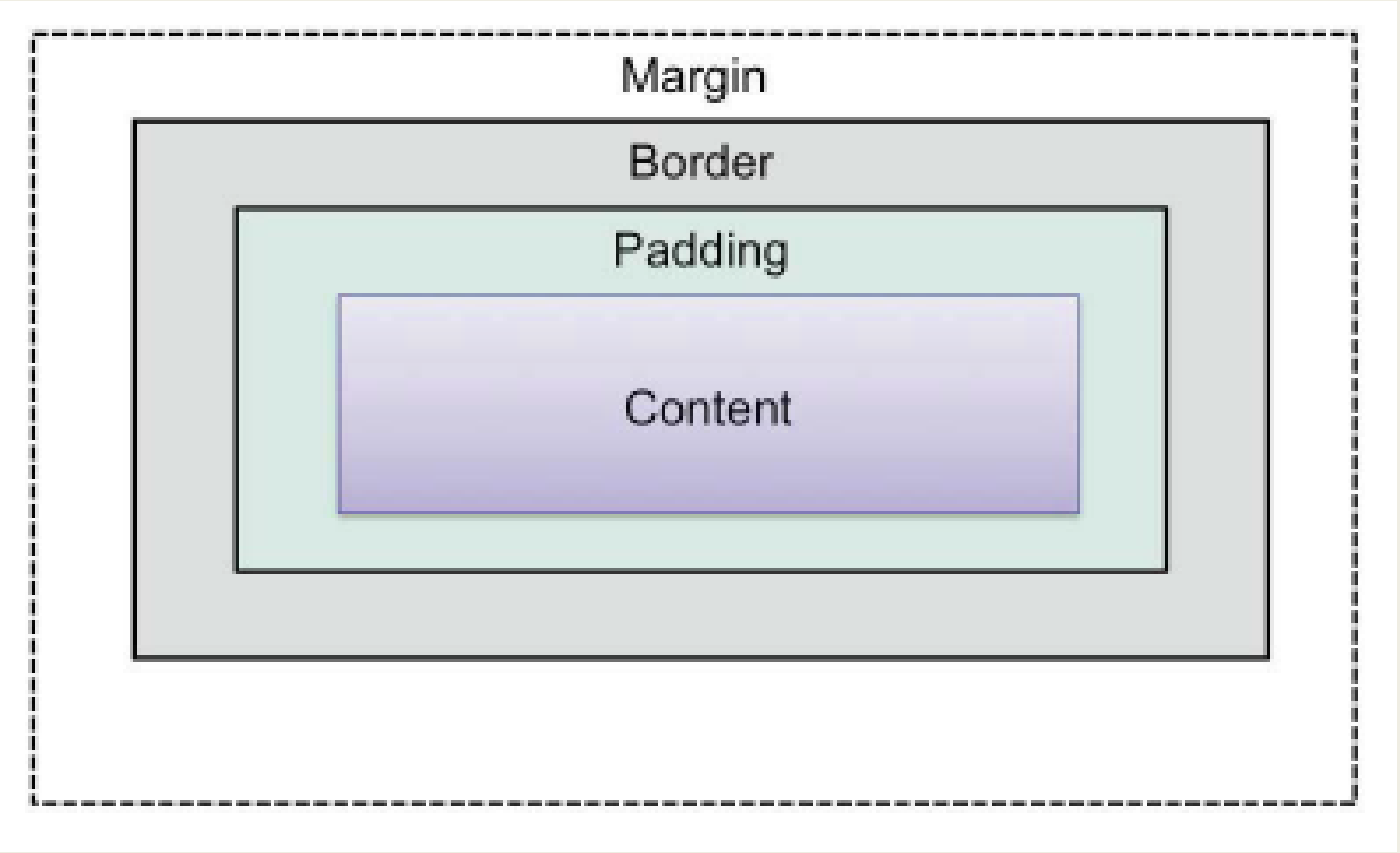


Grid Layout

- Blocks and floats are typically not enough to enforce all the alignments you want in a UI
- User tables instead

```
<table>
<tr><td>Name:</td>
      <td><input type="text" /></td>
</tr>
<tr><td>Groups:</td>
      <td><textarea></textarea></td>
</tr>
</table>
```

Margin, Borders and Padding



Space-Filling and Alignment

- Width: 100%, height: 100% consumes all of the space available in the parent
- Vertical-align moves a node up an down in its parents box
  - *Baseline is good for lining up labels with textboxes*
  - *To and bottom are useful for other purposes*
- Centering
  - *Margin: auto for boxes*
  - *Text-align: center for inlines*

Absolute Positioning

- Setting position and size explicitly
  - *In coordinate system of entire window, or of node's parent*
  - *Css has several units: px, em, ex, pt*
  - *Mostly useful for popups*

```
<style>
button { position: absolute;
         left: 5px;
         top: 5px; }
</style>
```

Constraints

- Constraints is a relationship among variables that automatically maintained by system
  - *Constraints propagation: When a variable changes, other variables are automatically changed to satisfy constraint*

Using Constraint Layout


```
label1.left = 5
label1.width = textwidth(label1.text, label1.font)
label1.right = textbox.left
label1.left + label1.width = label1.right

textbox.width >= parent.width / 2
textbox.right <= label2.left

label2.right = parent.width
```

Using Constraints for Behaviour

- Input
  - *Checker. (x,y) =mouse(x,y)*

  If mouse,button1 && mouse.(x,y) in checker
- Output
  - *Checker.dropShadow.visible=mouse.button1 && mouse.(x,y) in checker*
- Interactions between components
  - *deleteButton.enabled=(textbox.selection != null)*
- Connecting view to model
  - *Checker.x = board.find(checker).column*50*

Constraints are Declarative UI

$$\frac{\text{scrollbar.thumb.y}}{\text{scrollbar.track.height} - \text{scrollbar.thumb.height}} = \frac{\text{-scrollpane.child.y}}{\text{scrollpane.child.height} - \text{scrollpane.height}}$$