CSS233 WEB PROGRAMMING I

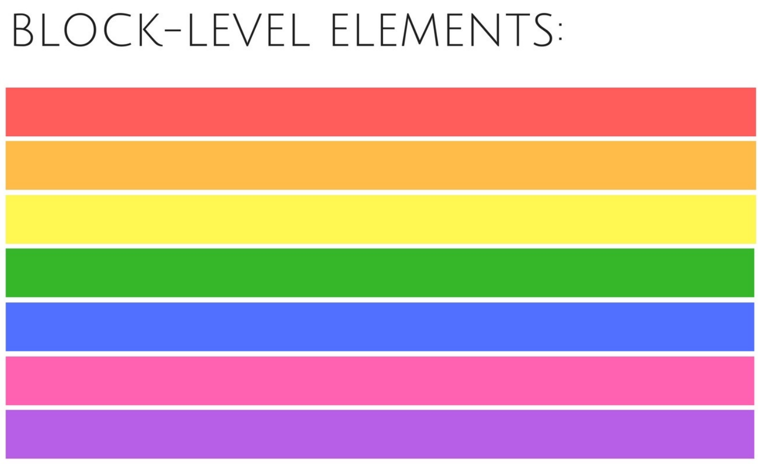
LECTURE 07 CSS PART 3

Dr. Wittawin Susutti
wittawin.sus@kmutt.ac.th

Block elements

Examples:
`<p>, <h1>, <blockquote>, <ol>, <ul>, <table>`

- Take up the full width of the page
- Have a height and width
- Width can be modified
- Can have block or inline elements as children



Cr.Emmanuel Ndubuisi

Inline elements

- **Examples:**

  `<a>, <em>, <strong>, <br>`

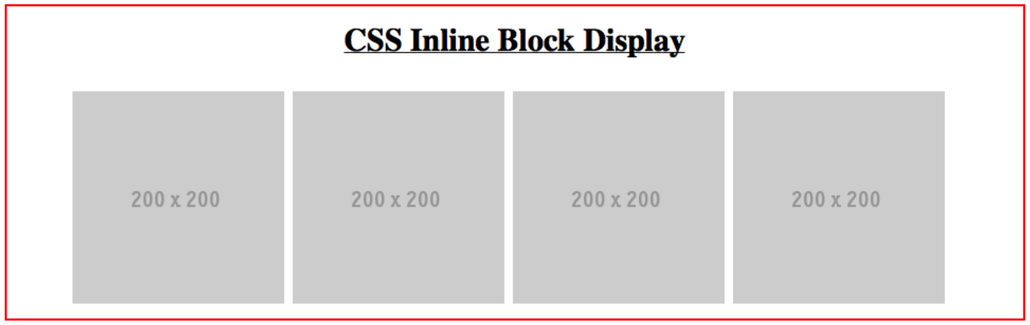
INLINE ELEMENTS:
Cr.Emmanuel Ndubuisi

- Take up only as much width as needed
- **Cannot** have height and width
- **Cannot** have a block element child
- **Cannot** set **width** on inline element, so it is ignored
- **Cannot** be positioned (i.e., CSS properties like float and position do not apply to inline elements)
  - Must position **its containing block element** instead

inline-block

- Examples:

  `<img>`, any element with `display: inline-block;`
- Width is the size of the content, i.e., it takes only as much space as needed
- **Can** have height and width
- **Can** have a block element as a child
- **Can** be positioned (i.e., CSS properties like float and position apply)

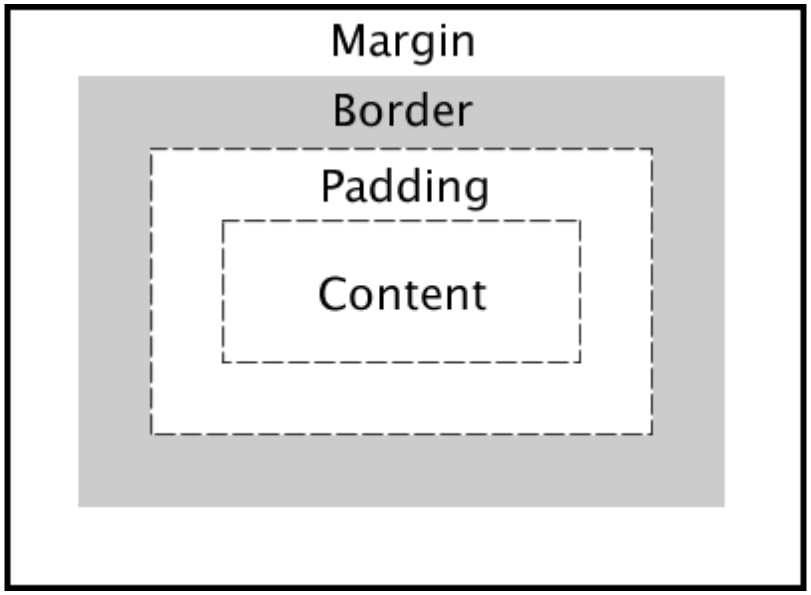


Cr.SusanLi

The CSS Box Model

- Every element is composed of 4 layers:
  - the element's content
  - the **border** around the element's content
  - **padding** space between the content and border (inside)
  - a **margin** clears the area around border (outside)

```
p {
  border: 2px solid red;
  padding: 1px;
  margin: 2px 4px 3px 1px;
}
```



- **One value**—The size of all the margins
- **Two values**—The size of the top/bottom margins and the left/right margins (in that order)
- **Three values**—The size of the top margin, the left and right margins (they are given the same value), and the bottom margin (in that order)
- **Four values**—The size of the top, right, bottom, and left margins (in that order)

Position

- Positioning determine where elements appear on the screen, and how they appear.
- Position can have those 5 values:
  - `static`
  - `relative`
  - `absolute`
  - `fixed`
  - `sticky`

Position

- **Static** positioning
  - This is the default value for an element.
  - `Static` positioned elements are displayed in the normal page flow.
- **Relative** positioning
  - The `relative` value positions the element relative to the original position in the document.
  - The element is positioned with the `top`, `right`, `bottom`, and `left` values.
- **Absolute** positioning
  - The `absolute` value removes the element completely from its normal flow in the document.
  - The element is positioned relative to their closest positioned parent element.

Position

- **Fixed** positioning
  - The `fixed` value positions an element to remain fixed in the same position, even when the page is scrolled.
  - It is similar to the `absolute` value, but it remains relative to the viewport at all times.

- **Sticky** positioning
  - The `sticky` value positions the element as a combination of relative and fixed values.
  - The sticky position behaves like relative positioning until the element reaches a certain scroll point on the screen. After that, the element sticks to the top of the viewport like a fixed element.

Transitions

- One way CSS lets us control animation in the browser with the transition property.

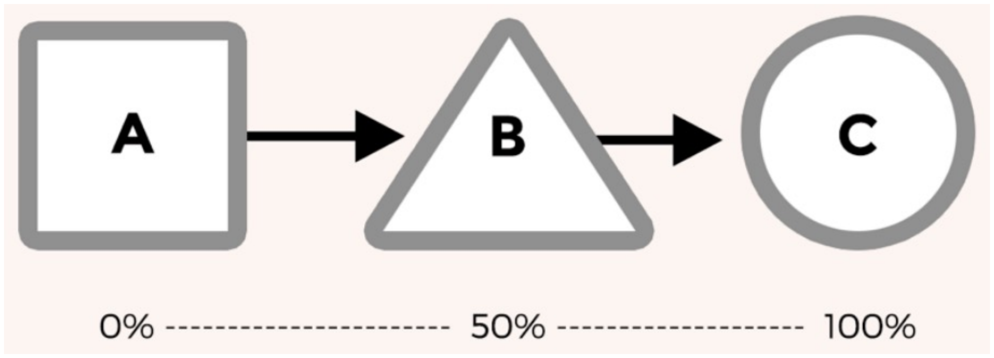
- When we use a transition on an element, we tell the browser that we want it to interpolate, or automatically calculate, the change between states.

- For example, we can change an element’s style on hover, apply a transition, and the browser will create a smooth animation between the element’s starting style and its new style.

Animations

- Transitions and animations are similar.
- Both take the form of a CSS property, and have duration, delay and other ways of controlling how the browser creates the movement.
- While transitions are all about smoothing the change from state A to state B, animations are a way to describe multiple steps.

Animations



- In the above example, there are 3 states (A, B and C).
- A transition would only go from A to C while an animation allows us to specify what step B looks like and make sure the animation follows all three steps.
- Animations also behave a little differently.
- They can begin automatically.
- While a transition might require adding a class or a change of state such as hovering, animations can start when the page loads.

Transitions in action

- A transition is a property in CSS.
- We can write a transition in CSS like this:

```
transition: background 0.5s linear;
```

- In this case we're telling the browser that a transition of the background property, will take half a second, and use the "linear" timing function.
- The above property might cause a button's background to change when hovered over:

```
button {
        background: white;
        transition: background 0.5s linear;
}
button:hover {
        background: green;
}
```

Transitions properties

- `transition-property`: the name or names of the CSS properties to which transitions should be applied
- `transition-duration`: the duration over which transitions should occur
- `transition-timing-function`: a function to define how intermediate values for properties are computed
- `transition-delay`: how long to wait between the time a property is changed, and the transition begins
- The transition shorthand CSS syntax

```
transition: <property> <duration> <timing-function> <delay>;
```

Shorthand vs Longhand

- When writing CSS, we can often summarize multiple properties into one in a shorthand property.
- For example, padding written as shorthand might look like this:

```
padding-top: 10px; padding-right: 20px; padding-bottom: 15px; padding-left: 25px;
```

```
padding: 10px 20px 15px 25px;
```

- In the same way, we can write a transition as shorthand too:

```
transition: <property> <duration> <delay> <timing-function>;
```

```
transition-property: all; transition-duration: 0.5s; transition-delay: 1s;
transition-timing-function: linear;
```

```
transition: all 0.5s 1s linear;
```

Things transitions don’t work on

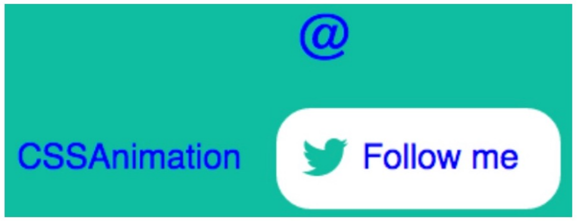
- While you can use transitions on positioning, size, color, border, background-position and many others, there are some that cannot be transitioned.
- The **font-family** cannot be transitioned, as this would mean trying to generate frames between two very different font images.
- **Background images** created with CSS, such as generated gradients, cannot have their properties animated.
- However, you can animate things like opacity and background position.
- By moving background images around or hiding them you can create interesting effects.

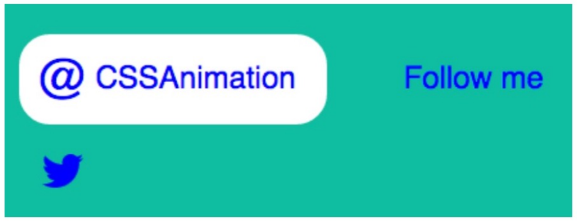
Multiple transitions

- We can combine multiple transitions into a single button for a more interesting effect.
- Example: Fancy button
  - In this example a hover effect combines several changes of state, but all are defined by a single transition:

```
transform 0.4s cubic-bezier(.72,-0.61,.25,1.51);
```

  - Break down:





Animations in action

- The animation property is applied to an element just like a transition.
- It also needs a second part, called `keyframes`.

```
element {
     animation: ...
}
@keyframes animation-name {
     /* Keyframes go here */
}
```

- One benefit of having the `keyframes` defined separately is that it allows us to create animations that can be reused multiple times.

The animation property

- Applying these keyframes to an element is done with the animation property.
- An animation could be written as the following shorthand:

```
animation: change-background 4s linear infinite;
```

- Written as individual properties it would look like:

```
animation-name: change-background;
animation-duration: 4s;
animation-timing-function: linear;
animation-repeat: infinite;
```

- Where a transition takes a property, such as “background” or “all”, the animation property is given the name of the set of keyframes that describe the animation sequence.

Keyframes

- A set of keyframes in CSS is a series of stops along the way through an animation.
- Each “keyframe” is a written as a percentage.

```
@keyframes change-background {
       0% {
              background: blue;
       }
       50% {
              background: orange;
       }
       100% {
              background: green;
       }
}
```

Animation properties

- **`animation-delay`**: to make the animation wait before starting.
- **`animation-direction`**: to control the animation direction `[normal, reverse, alternate` and `alternate-reverse].`
    - `reverse` causes it to play (and loop) from 100% to 0%,
      while `alternate` plays from 0% to 100% and back again to 0%
- **`animation-duration`**: the length of the animation
- **`animation-fill-mode`**: the state of the element after animation end
    - By default, an animation will play and then the element returns to its normal state.
    - Using the value forwards tells the animation to finish and stay on the last keyframe.
    - The value backwards returns to the first keyframe when the animation finishes.

Animation properties

- `animation-iteration-count`: the number of times the animation plays.
    - By default, it will play once. You can specify a number, or `infinite` to have it loop forever.
- `animation-name`: refers to the keyframes associated with the animation.
- `animation-play-state`: to pause or resume an animation, might be to set this value on an animation using JavaScript.
- `animation-timing-function`: applies between each keyframe.

Keyframes in action

```
@keyframes name {
     from {
          …
     }
     to {
          …
     }
}
```

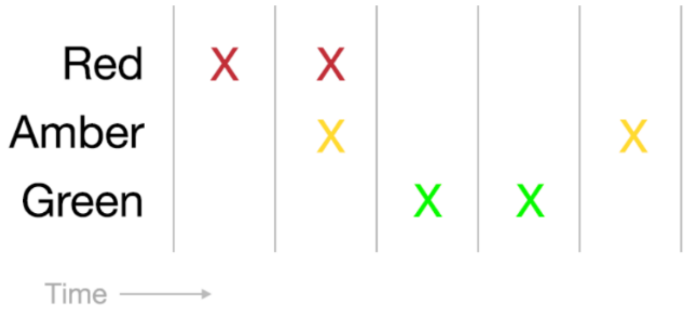
```
@keyframes name {
     0%, 20% {
          opacity: 0;
     }
     100% {
          opacity: 1;
     }
}
```

- Example: Save button wiggle effect

Multiple animations

- How we can make use of multiple sets of keyframes running at the same time.
- There are times when we want multiple animations on a page to stay in sync, but at the same time each animation has its own timing.
- Example: Traffic light
    - Decompose and plan the keyframes

```css
.red {
        animation: red 10s linear infinite;
}
.amber {
        animation: amber 10s linear infinite;
}
.green {
        animation: green 10s linear infinite;
}
```


Red
Amber
Green
X
X
X
X
X
X
X
Time