Advanced Ideas

Pseudo Classes & Elements

- pseudo-classes = elements that are dynamically populated or dependent on tree structure
- types of psuedo-classes: link
  user action
  forms (interface)
  structural/positional

- link: :link
  :visited

- user actions: :hover
  :active
  - holding down mouse button over an element

  :focus
  - tabbing checkpoints

- forms (interface): :enabled
  :checked
  :disabled
  - e.g. grey out sections that can't be filled until a previous box is completed

- structural/positional:

:first-child	:first-of-type
:last-child	:last-of-type
:nth-child()	:only-of-type
:only-child	:empty

- pseudo-elements = elements that aren't part of the DOM
  used to style specific/unique parts of the page

- types of pseudo-elements: textual
  positional/generated
  fragments

- textual: :first-letter
  :first-line

  - style first ___ w/ different font, color, size, etc

- positional/generated: :before
  :after

  - generate things to show up before or after elements

- fragments: ::selection
  - style fragments of different selections

Transitions

- when elements transition from one state to another, you can alter their appearance
- properties: transition-property
  - what is it you want to change? (size, color, position, etc)

  transition-duration
  - how long should each transition last?

  transition-timing
  - should it be a smooth transition (linear)? or differing speeds?

  transition-delay
  - how long should the wait be before the transition begins?

- setting up/steps: 1) define your element
  2) choose the properties for transition
  3) define the new values
     - must use pseudo-class

- using shorthands:

  $\longrightarrow$ (eg. transition: background .2s linear, border-radius 1s ease-in 1s;)

- use transitions sparingly! don't overwhelm page/user
- accessibility is an issue $\rightarrow$ don't require certain state
  make all content available

Transforms

- another way to change the appearance of elements
- often combined w/ state changes
- typically requires browser prefixes
- types: 2-dimensional
  3-dimensional

- 2D transform options: translate
  rotate
  scale
  skew
  matrix

- translate: +x move right
  -x move left
  +y move up
  -y move down

  - format: transform: translate(x,y);

  $\longrightarrow$ (eg. transform: translate(100,75);

- rotate : spin the element a certain # of degrees
  - format: transform: rotate (deg);
  - ⟶ (eg. transform: rotate (30 deg);)
- scale : change the width & height of element
  - format: transform: scale (width, height);
  - ⟶ (eg. transform: scale (2,3);)
- skew : rotate element a certain # of degrees along the x & y axes
  - format: transform: skew (x-angle, y-angle);
  - ⟶ (eg. transform: skew (30 deg, 15deg);)
- matrix = combines all of the 2D transform methods into one
  complicated / don't use
- 3D rotate : rotate along the x/y/z dimension along a given degree
  - format: transform: rotateX (deg);
    transform: rotateY (deg);
    transform: rotateZ (deg);
    transform: rotate3d (x, y, z);
- other 3D transforms: 3D scale
  3D translate

Positioning

- position values: static
  relative
  absolute
  fixed
  modifiable by: top property
  right property
  bottom property
  left property
- static : default value for elements
  place in next available position
  not affected by t, b, l, r properties
  - format: position: static;
- relative : positioned relative to itself
  static position but can add t, b, l, r offset(s)
  new position doesn't affect other elements
  often used as container blocks for absolutely positioned element

- format: position: relative;

- absolute: element is removed from document flow & positioned relative to its nearest ancestor (aka the root)
  other elements behave as if element DNE
  can end up on top of another element

  - format: position: absolute;

- fixed: positioned relative to the browser window
  won't move, even if window is scrolled → can't escape / follows you

  - format: position: fixed;

    ⟶ (eg. popup ads): doesn't go away

    ⟶ (eg. navigation bar): always visible on top

- z-index: (in the case of multiple elements placed in same position / stacked on top of each other)
  # that dictates stacking order
  + value higher in stack (top)
  - value lower in stack (bottom)

- positioning elements is the key to achieving desired layout(s)
- plan properly before coding to make this^ easier