CSS3

Notes open for creative commons use by CryptoLain at https://github.com/cryptolain *and* klabweb@tuta.io

Learning Sources
MDN Web Docs – MDN Web Docs – CSS3 ← Primary
CSS3 Docs – W3 ← Reference

Intro to CSS3

-CSS - Cascading Style Sheets. Creates rules which are applied to a document to detail how to display the content. Rules made of properties & selectors:
-Property - A characteristic (ex. *color:*) whose value defines how to display element(s)
-Declaration - A property paired with a value (ex. *blue*)
-Selector - Describes what element(s) the property will match and apply to
-Rules contained within stylesheets

-External stylesheet applied to HTML doc by including *<link rel="stylesheet" href="style.css">* in *<head>*
-Can also include styles inside HTML doc by including *<style>css code</style>* in *<head>*, but not very efficient

-General Syntax: *selector {*
property: value;
}

-ex. *h1 {* *<!--sets <h1> headers to blue text w/ yellow BG w/ black borders-->*
color: blue;
background-color: yellow;
border: 1px solid black;
}

-Order of operations for processing: Browser loads HTML and parses HTML, then loads CSS & parses CSS, then creates DOM (Document Object Model) tree, then displays.

-DOM - built as a tree structure generated from HTML (or XHTML) doc. Each element, attribute, and piece of text in the DOM is a node. Each node is related to other nodes by where they are nested (parent, child, grandparent, etc.).

CSS Basic Syntax
-Declaration syntax - *property: value* (ex. *background-color: red*)
-If unknown property or invalid value, declaration ignored by CSS browser engine

Declaration blocks
-Declarations grouped together in opening and closing braces with declarations separated by semi-colons:
{ propertyA: valueA;
propertyB: valueB;
propertyC: valueC}
-An empty declaration block is acceptable.

Selectors & Rules
-Selector - prefaces declaration block and tells block what to apply to. Can be element name, multiple elements selected by a comma, element but only w/ a specific id, etc.
-ex: *div, p, #lain-box { declarations}* */* applies to all div & p, & unique lain-box */*
-Element may be matched by several separate selectors

-Declaration block + selector = ruleset (ie rule)

CSS Statements
-At-rules - Convey metadata, conditional info, and other descriptive info.
-Syntax: *@identifierX 'syntaxBlock';* ex. *@import 'custom.css';*

-Nested statements - conditional requirement for at-rule w/ nested rule that is only applied if requirement is met.
-ex. *@media (min-width: 801px) {*
body {
margin: 0 auto;
width: 800px
}
}

-Types:
-*@media* - checks condition on device running the browser
-*@support* - checks condition to see if browser supports feature
-*@document* - checks if page matches xyz condition(s)

-Comments syntax: /* this is a comment */

-Can shorthand some properties:
-ex. can do *background: red url(bg-graphic.png) 10px 10px*
instead of *background-color: red;*
background-image: url(bg-graphic.png);
background-position: 10px 10px;

Selectors
Types of Selectors:
A) Simple selectors - match 1+ element based on element type, class, or id
B) Attribute selectors - match 1+ element based on their attributes/attribute values
C) Pseudo-classes - match 1+ element based on state (ex. being hovered over, is child of x, etc.)
D) Combinators - Combine selectors (ex. only <*p*> that come directly after headings)
E) Multiple selectors - multi selectors w/ comma separators in same rule. Apply to all elements in selector
*note D) and E) not actually selectors, but similar to

Simple Selectors
Type selectors (aka element selectors) - Simple case sensitive match to an HTML element(s)
-ex. *p { declarations }* applies to all <*p*>

-*ex. h1, p, ul {declarations }* applies to all *<h1>*, *<p>*, and *<ul>*

Class selectors - apply to a named HTML class.
-Syntax: *.classname { declarations }*
-HTML class def: <element class="className Optional2ndClassName xClassName">
-Multiple class names allow you to apply multiple rules on same element (ie. one rule applied to *.className {..}*, a second rule via *.classNameTwo{…}*, ...)

ID Selectors - applied to unique HTML ID (html *id*="" attr) . Selector similar to doc frag in HTML.
-Syntax: *#uniqueID { declarations }*

Universal Selectors - apply to all elements on page. Rarely used. Sometimes used as part of combinator.
-Syntax: ** { declarations }*

Attribute Selectors
-match 1+ element based on their attributes/attribute values
-syntax: *[attributeName<optionalCondition]*
-ex. *[attr^=value]* -ex. *[attr=value]*

Presence & Value Attr Selectors
-For conditionals of exact values

-*[attr]* - applies to all elements with attr (of any value)
-*[attr="val"]* - applies to all elements w/ attr of specified value

-*[attr~="val"]* - applies to elements w/ specified value separated by spaces in any of attribute values
-ex. *<ul>*
*<li data-quantity="3" data-vegetable>Garlic
*
*<li data-quantity="700g" data-vegetable="not spicy like chili">Red pepper
*
*
* */* [data-vegetable~="spicy"] would apply to second <li> */*

Substring Value Attr Selectors
-Offer similar functions to regex selection

-*[attr^="val"]* - applies to all elements where attr starts with val
-also *[attr|=val]* - applies to all elements where attr is exactly *val* or starts with *val-* (ex. *en-us*)
-*[attr$="val"]* - applies to all elements where attr ends with val
-*[att*="val"]* - applies to all elements where attr contains val (ie exists as substring)

Pseudo-classes & Pseudo-elements
Pseudo-selectors -Apply to certain parts of elements or only elements in certain contexts

-Two types: pseudo-classes & pseudo-elements

Pseudo-classes
-Styles an element, but only when in a selected state (ex. being hovered by mouse)
-Syntax: *element:state { … }* -ex. *a:visited { color: red }*

:active	:indeterminate	:only-child
:checked	:in-range	:only-of-type
:default	:invalid	:optional
:dir()	:lang()	:out-of-range
:disabled	:last-child	:read-only
:empty	:last-of-type	:read-write
:enabled	:left	:required
:first	:link	:right
:first-child	:matches()	:root
:first-of-type	:not()	:scope
:fullscreen	:nth-child()	:target
:focus	:nth-last-child()	:valid
:focus-within	:nth-last-of-type()	:visited
:hover	:nth-of-type()	

Pseudo-elements
-Apply to only part of the element or after/before/etc. part of the element
-Syntax: *element::keyword { … }*
-ex. *p::first-letter { font-size: 140%}*
-ex. *[href^=”http”]::after {content: ' ';} /*all w/ href starting in http will have arrow following*

-Some common ones: *::after, ::before, ::first-letter, ::first-line, ::selection, ::backdrop*

Combinators & Multiple Selectors

Group of selectors - *A, B {…}*
-Applies to any element of A and/or B.
-ex. *p, h1 {…}* /*Declarations applied to all <*p*> and <*h1*>*/

Descendant combinator - *A B {…}*
-Applies to any elm matching B that is a descendant of A
-ex. *table td thead th {..}* /*Applies to all *th* inside a *thead* inside a *table**/

Child combinator - *A > B {…}*
-Applies to any elm B that is a direct child of A

Adjacent Sibling Combinator: *A + B {…}*
-Applies to any elm B that immediately follows A, where both are children of same parent
-ex. *table th + td {...}* /*Applies all <*td*> directly following <*th*>, where both children of <*table*>*/

General Sibling Combinator: *A ~ B {…}*
-Applies to any elm B that follows A (though does not need to be immediately after), where both are children of same parent

CSS Values & Units
Numeric Values
-Length/size units - can be provided via *#px* (ex. *width: 350px*) or relative units, which are sized in relation to *font-size*, viewport size, etc..

Relative units

-*em* - specified *#em*, where *1em* is the size as the font size. Most common relative unit. Note, that since element font sizes are inherited from parents, *1em* will be equal to parent font size if not overridden by child, meaning if parent font size changes, so will *em*.

-ex. *.element {*
font-size: 20px
width: 4em; */*.element is 80px x 40px (w x h)*/*
height: 2em;
}

-*rem* - root em. Same as *em*, except *1rem* is equal to base font size inherent to root (usually *<html>)*, disregarding inherited sizes of parent. Prefer *rem* over *em* as makes maintenance and modding code way easier.

-*em* and *rem* generally used when layout needs to scale to font size (ex. font size increases, so does onion layers around surrounding box(es)

-*vw, vh* - 1/100th of the width, height of the viewport (ex. *40vh* is 40%*)*

-Unitless Values - can either be zero (ex. *margin: 0;*), a multiplier value (ex. *line-height: 1.5;*), number of times to perform action xyz (ex. *animation-iteration-count: 5;*), etc.

Percents

-Syntax - *property: x%;*

-Element sized with % value changes in size based on size of parent. Called a liquid layout, vs a fixed width layout (set px). Useful for responsive dev.

-Fixed width useful for if want element (ex. map) to stay same size & allowing scrolling, dragging, etc. through

-ex. If *<img>* or * * set to *width: 70%* and their text elements set to *font-size: 200%*, where *<body>* is parent of *<img>* and * * and *<html>* is root (default font size usually 16px), * * and *<img>* will resize to 70% of viewport size and font will resize to 200% of *<html>* (32px)

Colors

-16.7 million colors

-Keywords - 150 defined colors (ex. *red, magenta, black*, etc.) available

-Hexadecimal - (ex. *#000ff*)

-RGB - *rgb(###, ###, ###)* where # is value between 0 and 255. (ex. *rgb(255, 0, 0)* is red)

-HSL - Hue-Saturation-Lightness. Hue = base color shade with 0 to 360 value. Saturation = 0 to 100% value. Lightness = 0 to 100% value. (ex. *hsl(240, 100%, 50%)* is blue)

-HSL & RGB also have HSLA & RGBA modes, where 'A' is alpha & sets transparency with 0 to 1 value

-Note that *rgb(int, int, int), hsl(int, int, int), etc* are CSS functions, as is anything else with *functionName(…)* syntax

Opacity
-Sets transparency of all selected elements & their children
-Syntax: *opacity: #; /* 0 to 1 value*/*

Cascade & Inheritance
The Cascade
-*Cascading* Style Sheets
-Controls which rule applies to an element when multiple rules exist that select the same element. More specifically, what property from what rule overrides what other property.

-ex. (selector exists for *p*, *p* with *id=”blep”*, and *p* with *class=”bloop*” all specify *color:*… Which one wins and is applied?)

-Three components determine the cascade, listed in order of weight: 1. Importance, 2. Specificity, 3. Source order

Importance
-Can make sure a specific declaration will always win via *!important* syntax
-Syntax: *property: value !important;* (ex. *color: blue !important;)*
-Avoid using unless absolutely necessary as changes way cascade works and makes debugging very hard

-User applied stylesheets with *!important* tag will override all other stylesheets applied to page (author stylesheet, user agent stylesheet, etc)

Specificity
-The most specific selector is applied if multi w/ varying specificity exist
-In order of least to most specific: *element → class → id → !important*

-Can measure specificity via 4 digit number: ABCD. Higher numbers win.
  -A - 1 if deceleration contained in *style*, else 0 (if A = 1, BCD = 000)
  -B - sum of unique ids in selector
  -C - sum of each class selector, attr selector, & pseudo-class
  -D - sum of each element selector & pseudo-element
  -ex. *h1 + p::first-letter /*0003 as 2 element selectors + 1 pseudo element*/*
  -ex. *li > a[href*=”en-US”] > .inline-warning */0022 as 1 attr, 2 elm, 1 class*/*

Source Order
-If rules still conflict after importance and specificity, then whichever rule is later in the source code wins

Inheritance
-Specifies whether a property value is inherited by children via *property: inheritanceDetails;*
  -ex. color: inherit;

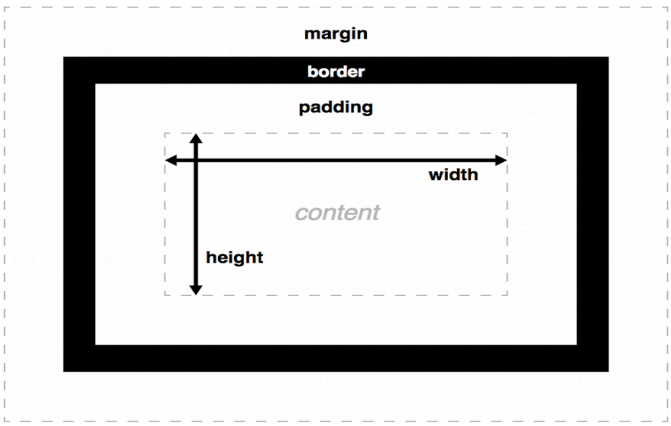
Inheritance Details
-*inherit* - sets property value to be inherited from parent
-*initial* - sets property value to be inherited from browser default style sheet, or parent if no browser default
-*unset* - sets property value to default value to prevent inheritance from properties that are inherited by default. Will then inherit from parent instead.
-*revert* - reverts the property value of the origin (and user value is applied instead, or user agent stylesheet, etc. if no user)

-Can apply an inheritance detail to all properties via *all: inheritanceDetail;* Useful for setting a temp blank slate during dev.

-CSS reference details inheritance details for all properties, included if inherited or not

The Box Model
-The foundation of web layout. Everything on a page (header, footer, main, image, etc.) can be seen as a box where each element is represented as a rectangular box, and set to arrange around other boxes via it's layers.

Box Properties
-Box made of a series of "onion layers" of various size. Note that all of HTML follows this design, even if the details of the box are not expanded.



-Content layer - Innermost layer. Size determined by *width* and *height* properties. Can also set relative via *min-width*, *max-width*, *min-height*, *max-height.*

-Padding - The inner margin surrounding content. Uniform on all four sides with *padding* or defined on all four sides via *padding-top, padding-right, padding-bottom,* and *padding-left.* Empty space. Often set via % of block.

-Border - Outside of padding. Default = 0px. Same options as padding, except *border* or *border-someSide*. Also, *border-style, border-color, border-width* and *border-top-width, border-left-style,*

border-right-color, etc. Ignores % values.

-Margin - Surrounds border and acts as white space between box and other elements. *margin, margin-left, margin-right, etc.*. Note that when margins touch, the distance between is value of larger margin only, not sum of both.

-When short-handing above, is *padding: top right bottom left*, etc., and *margin: H x W*, etc.

min-width, max-width, min-height, max-height
-Can set so box doesn't pass a certain width (ex. 1280px)
-*margin: 0 auto;* - centers box inside it's parent
-Can stop an item from overflowing out of container/viewport with:

```
display: block;          //makes into block element
margin: 0 auto;          //centers inside parent
max-width: 100%;         //shrinks element to fit w/i container
```

Overflow
-Occurs when box set to fixed dimensions & content overflows outside of box
-*overflow: value* - controls what happens when overflow occurs. Common values:
-*visible* - overflowing content show outside box. Default.
-*auto* - overflowing content hidden and scroll bars shown to navigate
-*hidden* - overflowing content hidden

Background Clip
-By default, box backgrounds stack on top of each other under box and stretch to outer edge of border
-Background image set in block elements such as *div* via *background-color or background-image: url("url or fileLocation Here").*
-Background size set via *background-size: value*;

-*background-clip: value* - property to select how far background stretches out. Common values:
-*border-box* - stretches across padding and border. Default.
-*padding-box* - stretches across padding, stops before border
-*content-box* - only inside content, stops before padding
-If set *border* to be transparent, that section of background-clip will inherit transparency

Types of Boxes
-Notes so far have all been for block level elements
-Type specified by *display* property.

Most common *display* values:
-*block* - content before and after box on separate line break (aka stacked)
-*inline* - flows smoothly with inline elements with no breaks. Layers outside of content update the position of surrounding inline elements but have no affect on *block* boxes. Can not be sized w/

width and *height*.
-*inline-block* - flows with other inline elements like *inline*, but can be sized with *width* and *height* and maintains *padding*, etc with *block* elements.

Styling Text

-Good standard to set <*html*> font size at 10px, then manually set all main sizes. 10Px makes it easy to multiply.
-Ex. Set <*html*> *to 10px*. Then, body font-size to 1.6rem. Then customize indi items as needed.
-*color* already detailed previously

Font Families
-*font-family: fontName;* - sets font (ex. Arial). Only applies if font available on machine, else browser default. Ex. *font-family: helvetica, arial, sans-serif;*

-Web safe fonts (Font name (generic type): Arial (sans-serif) using Helvetica as preferred alternative, Courier New (monospace) using Courier as preferred alt, Gerogia (serif), Times New Roman (serif) w/ Times as preferred alt, Trebuchet MS (sans-serif) beware as often not on mobile OS, Verdana (sans-serif)
-Web safe font list: `https://www.cssfontstack.com/`

-Font stack - listing multi fonts in case first font(s) not available for browser. First listed tried first.
-syntax: *font-family: fontNameA, "Font NameB", fontNameC; /*if font name has space, put in ""*/*

-*font-size* - property that can take px, em, or rem value. Inherited from parent element, in inheritance chain, with *html* being the root element. Default *html* usually = 16px.

Styling Fonts
-*font-style* property - for italics. Values: *normal* (no italics), *italic*, *oblique* (simulated italic font)

-*font-weight* property - for bolds of various weight. Values: *normal* (no bold), *bold*, *lighter* (bold one step lighter than parent's boldness), *bolder* (bold one step bolder than parent's boldness), *100* to *900* numeric value

-*text-transform* property - sets text as specified. Values: *none*, *uppercase*, *lowercase*, *capitalize* (First Letters Capitalized), *full-width* (puts in full width square)

-*text-decoration* property - line interactions with text. Values: *none*, *underline*, *overline* (like underline but above text), *line-through* (~~example~~). Can take multi values at once and can shorthand by declaring *text-decoration* then multi values.

-*text-shadow* property - takes up to four values, *#px #px #px colorName*, where values represent horizontalOffset verticalOffset blurRadius (higher value = more dispersed shadow) colorOfShadow.

-px & color can be replaced w/ other size vals. Can also use -px to set shadow in opposite dir.

-can give text multi shadows by specifying multi of above separated by commas

Text Layout

-*text-align* - property for where text is aligned in containing box

-values: *left, center, right, justify* (spreads text out so that lines of text are all same width)

-*line-height* - sets space between lines. Can take various length & size units. Multiplier (ex. *1.5*) often best and multiplies on *font-size* to get *line-height.*

-*letter-spacing* - property to set space between letters. Takes most length & size units.

-*word-spacing* - property to set space between words. Takes most length & size units.

More Common Text Properties

Font styles:

- `font-variant`: Switch between small caps and normal font alternatives.
- `font-kerning`: Switch font kerning options on and off.
- `font-feature-settings`: Switch various OpenType font features on and off.
- `font-variant-alternates`: Control the use of alternate glyphs for a given font-face.
- `font-variant-caps`: Control the use of alternate capital glyphs.
- `font-variant-east-asian`: Control the usage of alternate glyphs for East Asian scripts, like Japanese and Chinese.
- `font-variant-ligatures`: Control which ligatures and contextual forms are used in text.
- `font-variant-numeric`: Control the usage of alternate glyphs for numbers, fractions, and ordinal markers.
- `font-variant-position`: Control the usage of alternate glyphs of smaller sizes positioned as superscript or subscript.
- `font-size-adjust`: Adjust the visual size of the font independently of its actual font size
- `font-stretch`: Switch between possible alternative stretched versions of a given font.
- `text-underline-position`: Specify the position of underlines set using the `text-decoration-line` property `underline` value.
- `text-rendering`: Try to perform some text rendering optimization.

Text layout styles

- `text-indent`: Specify how much horizontal space should be left before the beginning of the first line of the text content.
- `text-overflow`: Define how overflowed content that is not displayed is signaled to users.
- `white-space`: Define how whitespace and associated line breaks inside the element are handled.
- `word-break`: Specify whether to break lines within words.
- `direction`: Define the text direction (This depends on the language and usually it's better to let HTML handle that part as it is tied to the text content.)
- `hyphens`: Switch on and off hyphenation for supported languages.
- `line-break`: Relax or strengthen line breaking for Asian languages.
- `text-align-last`: Define how the last line of a block or a line, right before a forced line break, is aligned.
- `text-orientation`: Define the orientation of the text in a line.
- `word-wrap`: Specify whether or not the browser may break lines within words in order to prevent overflow.
- `writing-mode`: Define whether lines of text are laid out horizontally or vertically and the direction in which subsequent lines flow.

Font shorthand

-When short-handing, write in following order:

font: font-style font-variant font-weight font-stretch font-size line-height font-family

-Only *font-size* and *font-family* are required when short-handing

Styling Lists

-*list-style-type* - property. Sets “type of bullet” values for lists. `Bullet types here`

-*list-style-positon* - property. Sets whether bullets sit as part of list (*inside* value) or separated (*outside* value. default)

-Set bullet image with *background* family discussed earlier by setting *ul li {...}* ruleset with *background* declarations. Allows to set padding, position, size, etc. Set image via *background-image* property. Example:

```
ul {
  padding-left: 2rem;            //important to have to prevent overlapping
  list-style-type: none;         //sets to no bullet type by default, so can replace with own
}

ul li {
  padding-left: 2rem;
  background-image: url(image.svg);
  background-position: 0 0;              //tells to appear at top left of every <li>
  background-size: 1.6rem 1.6rem;        //set to be same size as text
  background-repeat: no-repeat;          //set to only show once per <li>
}
```

-Can shorthand via *list-style: list-style-type list-style-image list-style-position*

List Counting
-*start=”#”* - html attribute, where # is number to start on
-*reversed* - html attribute. Binary. Count down to 1 (or *start)*
-*value=”#”* - html attribute. Sets list item as # specified. Should set for each <*li*> if using

Styling Links
-Link states (can be styled by pseudo-classes):
-*:link* (default state) -*:visited* -*:hover* (being hovered over)
-*:focus* (focused on by tabbing to via keyboard, etc) -*:active* (when it is being clicked on)

-There are certain longstanding styling people expect links to have (ex. blue underlined).

-Common to include icon next to link (ex. external link icon)
-example:
```
a[href*="http"] {
    background: url(externalLink.svg) no-repeat 100% 0;
     background-size: 16px 16px;
     padding-right: 19px;
}
```

Links as Buttons
-Can do by putting links in <*ul*>, then setting <*li*> to display as *inline* so sit next to each other, then set <*a*> as *inline-block* with different *background-color* for different states. Make sure to also set proper width, text color, margins, etc.

Web Fonts
-Fonts specified in a stylesheet, which are then downloaded and used by the browser in styling
-Syntax:
```
@font-face {
```

font-family: “fontName”;
src: url(“fontLocation.ttf”) format(“fontType”);
}

-Can then add font to site as would any other font via *font-family: “fontName”;*

-See here for free and paid font sources

-To ensure cross-browser comparability, use web font generator to generate multi font types for any fonts used. Generator also generates required *@font-face {…}* rules, which can be copy pasted into stylesheets for use of fonts.

Review of Important Design Elements

-Set proper height, border, padding, etc. on elements to create smooth, detailed box modeling
-Use proper DOM modeling (*main, body, section, article,* etc.)

-When you start a site, here are some basics to cover:

1) set site *font-size* to 10px (recommended)
2) set relative font sizes for headings, paragraphs, lists, etc /*(if want to default all to same size as 16px inheritance, simply set *body { font-size: 1.6rem; }* after setting *html* font size to 10px*/
3) set your body *line-height*
4) center top level heading
5) Set pseudo-element state values for links (*:hover, :visited,* etc.)
6) Make sure lists match with styling of page
7) Set up navigation and footer and make sure match with styling

Styling Boxes

Box Model Note
-*box-sizing: border-box;* - Declaration. Normally a box width/height is sum of content + padding + border. *Border-box* sets is so box width/height is only content w/h

Background Styling
-By default background sits under content, padding, and border
-See *Backgroud-clip* section of this doc for previous notes & review

Background Basics
-*background-color* - property. Transparent by default.

-*background-image: url(fileLocation);* - declaration.

-*background-repeat* - property. By default *background-image* will repeat across background. This property sets how or if it will do so. Values:
no-repeat (only shown once) *repeat-x* (repeat horizontally across BG),
repeat-y (repeat vertically across BG) *repeat* (repeat horizontally & vertically across BG)

-*background-position* - property. Sets in specific position in box. Set in *#horizontal #vertical* value (ex. *background-position: 200px 25px*). Can take absolute, relative, percent, and keywords (*left, center, bottom*, etc.) values.

-*background-size: # #;* - declaration. #Width #height. Can take a variety of size values.

Gradients
-*background-image : linear-gradient(to direction, firstColor, secondColor);*
-ex. linear-gradient(to bottom, orange, yellow); //color changes from Y to O towards bottom
-Can set more than two colors and specify when to switch to next color by specifying *#%* after color. ex. *linear-gradient(to bottom, orange, yellow 40%, red)*
//move from orange to yellow for 40% of box, then yellow to red

Multiple Backgrounds
-Can include multiple backgrounds. Use same *background-image, background-position* etc. properties, and separate backgrounds with commas.
-Note that backgrounds stack on top of each other, with first listed being on top, last listed on bottom

Shorthand Review
font: font-style font-variant font-weight font-size/line-height font-family;

background: color image repeat position/size attachment;

border: width style color;

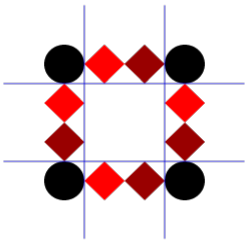
margin: top right bottom left;
padding: top right bottom left;
//may declare with 1-4 values

list-style: type position image;

Styling Borders
-See existing border section for review

Border Radius
-*border-radius: #px;* - Declaration. Rounded corners.
-Can take up to four *values - #pxTopLeft #pxTopRight #pxBottomLeft #pxBottomRight*
-Can also take other length units (*rems, %,* etc.)
-Can also make elliptical corners: *border-radius: #px / #px* or *border-radius: #px #px / #px #px*

Border Images
-Allows you to set images as border



-Start with a 3x3, 4x4, etc grid design image, which the browser then slices
-Then, tell browser how to parse image, by giving it pixels relative to the image size to split at. Example: image is 180px and a 3x3 grid, so each slice = 30px

-Syntax:
```
element {
    background-clip: padding-box;     //background color stops at padding
    border-image-source: url(imageFile.type);
    border-image-slice: #;
    border-image-repeat: type;
}
```

-*border-image-slice* can take up to four values for if slices different sizes: top, right, bottom, left

-*border-image-repeat* values: *stretch*, *repeat* (may result in image fragments), *round* (repeated, then stretched slightly so no image fragments), *space* (repeated, then small amount of space added between each copy so no fragments appear)

-Short-hand - *border-image: url(fileLocation) slice repeat;*

Styling Tables
-*table-layout* - property. Sets table to either *auto* or *fixed*, where *fixed* sets fixed with rows and *auto* auto sizes for each entry (messy looking). Set width via *width: #sizeValue;*

-Setting widths for columns - *thead th:nth-child(#) { width: #sizeValue; }*, where # is nth column. *nth-child()* property selects *n* child of parent element. Can also set height.

-*border-collapse* - property. Sets so table is styled where elements are separated by single (table lines look like grid) or *separate* lines where table has double lines between elements. *collapse* value by standard = typical.

-*border* - standard *border* property w/ standard attr/values. Also give <*th*> and <*td*> padding to create space between elements

-*background: url(location);* - include in *tfooter, thead, tbody,* etc. to set BG
-Can alternate row backgrounds via pseudo-class calls of *:nth-child(odd)* and *:nth-child(even)* on *tbody tr*

-Caption styling - style via typical text and box model methods. Has *caption-side* property for *top* or *bottom* for placed above or below table. *text-align:* property to align left, center, right, etc.

Box Shadows
-Pretty much same as text shadows but with, surprise, boxes

-box-*shadow* - property. takes up to four values, *#px #px #px colorName,* where values represent horizontalOffset verticalOffset blurRadius (higher value = more dispersed shadow) colorOfShadow.
-px & color can be replaced w/ other size vals. Can also use -px to set shadow in opposite dir.
-can give text multi shadows by specifying multi of above separated by commas
-*inset* - value. Keyword. Puts shadow inside box. Comes before *#px* in above deceleration.

-*filter* - property. Takes a variety of function values that can overlay on elements, such as text, images, etc. to alter. Functions similar to basic image editor. Some examples:
filter: blur(5px);
filter: brightness(0.4);
filter: contrast(200%);
filter: drop-shadow(16px 16px 20px blue);
filter: grayscale(50%);
-Since very new, should duplicate all properties in deceleration w/ same values but *-webkit-filter: …;* to enable cross-browser support
-Can short-hand by listing multiple functions w/ spaces between
-See `more values here`

CSS Layout

Review
-*display* - property. Change between *inline, block, inline-block, flex, grid, float, fixed, etc.* Primary layout method.
    -ex. setting <*li*> as default block = on top of each other, *inline,* now next to each other in row

-*position* - property to control position of boxes inside other boxes

Normal Flow
-How browser handles layout when do nothing to alter and leave as default
-By default elements interact based on if *inlin*e or *block* and based on *margin* interactions.
-*Block* on separate lines. *Inline* next to each other. *Inline-block* next to & can take WxH sizing, like *block.*
-Elements exist in box model with content, padding, and margin
-Default block size = 100% width of parent element
-Margins combine to be only as big as largest.

Flexbox
-One dimensional (row or column) box layout module
-When wrap elements in box (ex. <*div*>) and set to *display: flexbox,* elements in box will auto arrange themselves in a row or column, auto sizing, arranging, etc. based on defined flex properties

-*display: flex;* - set on container element (ex. *article, section, div). E*lements inside will arrange in row or column w/ all the same height

-Flexbox container called flex container. Runs along main axis which flex items are laid out on and cross axis which is perpendicular to main. Sizes of elements along axis have cross and main size, start, & end.

-*flex-direction* - property. Default is row. Can also set to column, for vertical flexbox

-*flex-wrap* - property. Flex creates additional rows if needed to have boxes that don't overflow (wider boxes, fewer per row). Generally used with *flex: sizeValue;* declaration to set min width for each flex item.

-shorthand - Can combine *flex-direction* and *flex-wrap* via *flex-flow: directionValue wrapValue;*

-*flex* - property. Specifies sizes of flex elements. Can give different elements in flexbox different sizes with. Common values:
- -*1*: smallest portion of main axis space
- -#: any num larger than *1*. Element takes up # spaces of total proportion

-ex. first two *article*s have *flex: 1*; Third article has *flex: 2*; Flexbox axis divided into units of 2's, where articles one and two are each 1/4 size, article three is 2/4 size. Total = 4/4.

-*flex* is actually a shorthand property. Full definition:
- -*flex-grow*: *size;* - as discussed above, for portion of axis to grow to
- -*flex-shrink* - as discussed above, for portion of axis to shrink to
- -*flex-basis* - min size as discussed above

Flex Item Alignment
-Align items across main axis via *align-items* and *justify-content* properties

-*align-items* - Controls where flex items sit on cross access. Values:
- -*stretch* - default. Items stretch to fill parent in direction of cross access direction. Items all become as long as longest item
- -*center* - items maintain intrinsic dimensions, but center along cross access
- -*flex-start and flex-end* - align items at start or end of cross access, respectively

-*align-self:* sets align on indi flex items. Same values as *align-items*

-*justify-content* - Controls where flex items sit on main access. Values:
- -*flex-start* and *flex-end* - all items sit at start/end of main axis
- -*center* - items maintain intrinsic dimensions, but center along main access
- -*space-around* - items evenly distributed along main axis with a bit of space at each end
- -*space-between* - same as *space-around*, but no space at ends

-justify-*self:* sets justify on indi flex items. Same values as *justify-content*

Flex Item Ordering
-Change layout of flex items without affecting source order

-*order: #;* - Deceleration. Would use in call with :*first-child*, :*last-child*, :*nth-child(#)*, etc.. Value # set to is position to be placed in flex. If two children have same #, places via order children appear in source. Can also set to negative # to add before '1' item.

Nesting Flex Boxes
Example. section - article <!--section flexbox w/ three article flex items-->
article
article - div - button <!--div flexbox w/ 5 button flex items-->
div button
div button
button
button

-Standard nesting, where each *flex* container gets marked with *display: flex;* and direct child items get *flex*, *justify-content*, *flex-flow*, etc. values

-Can cut down num of needed repeating rules with *:nth-of-type(x)* pseudo-class rule, where *(x)* is the *nth* (ex. 2nd) child of the parent calling rule on. Can also use *:first-child* instead of *nth-of-type(1)*

Grid Layout

-Two dimensional (row and column) box layout module
-Same idea as Flexbox (items arranged in size, fill percent etc. of grid container box based on set/default grid property values). More explicit item placement allowed than with flex.

-*display: grid;* - Deceleration. Sets container element as grid.
-By default, sizes to 1 column rows stacked on top each other (similar to normal flow)

-*grid-template-columns: #value #valueN;* - sets width of column, where each # val is a new col
-Can also size with *fr* unit, where each *fr* is an equal portion of the grid (same as *flex* use)
-Can shorthand long manual column entry (ex. 200px twenty times), can use *repeat(#, #fr)* function as value, where # is number times want grid item to generate and f*r* is size. ex. *repeat(20, 200p*x);

-*grid-row-gap* and *grid-column-gap* - set gaps between rows and columns. No *fr* units allowed.
-*grid-gap* - sets for both row and col gaps

Implicit vs Explicit Grid
-Explicit only specifies # columns and adds more rows (or vice versa) as more items added.

-*grid-auto-rows* - property. Implicit grid auto-sizes by default. Property takes size values & sets rows to that size height. Content overflows if larger than specified size.
-ex. *grid-auto-rows: 195px;*

-*min-max(sizeVal, sizeVal)* - function. For min and max size values.
-ex. *grid-auto-rows: min-max(100px, auto);* //min value 100px, max = auto

As Many Columns as Will Fit in Container
-Can set by do by calling *repeat(auto-fill, minmax(rowHeight, size));* for *grid-template-columns*

Line-based Placement
-properties taken by grid items to place them on grid

-*grid-column-start, grid-column-end, grid-row-start, grid-row-end* - 4 properties. Start tells to start at *x* position in grid, end to end at y position.
-Can shorthand with *grid-column: startVal / endVal;* and *grid-row: startVal / endVal*;
-*grid-*row and *grid-column* - also have one value (ex. *grid-r*ow: 1;)

-Note that # values represent edges of grid frames, not actual frames.
-Ex. *grid-column: 1 / 5;* covers the first 4 grid frames: 1 = edge of frame 1, 5 = edge of frame 4

-Useful for styling header, article, aside, footer, etc

Positioning with g*ri-template-areas*
-Define rows via *"nameOne NameTwo …."* where each name is a placeholder for an element that will be placed there after that name is assigned to that element

-element desired to place in name placeholder spot on grid assigned name via property *grid-area: nameX;*

-example:

```
.container {
    display: grid;
    grid-template-areas:
      "header header"
      "sidebar content"
      "footer footer";
    grid-template-columns: 1fr 3fr;
    grid-gap: 20px;
}

header {grid-area: header;}
article {grid-area: content;}
aside {grid-area: sidebar;}
footer {grid-area: footer;}
```

Grid Frameworks
-Typical frameworks (styles repeated throughout site) are either 12 or 16 columns wide
-Inspect with Firefox Grid Inspector

Floats
-Element set to float is removed from normal flow and set to flow to left or right of previous element, with surrounding elements "floating" around *float* element

-Created so image could float inside middle of text w/ text wrapping around sides of it. Using floats to create site columns = legacy design.

-*float* - property. Pulls element out of normal flow and set to *left* or *right*. Floats on side given on parent element. Ex. give *float*: *left*; to <*div*> with <*body*> as parent, will float on left side of <*body*>

-Text will float around body, the only box that moves is the float. Paragraph text will wrap around float, but float will be sitting on top of paragraph box too, etc..

-Margin must be set on float, not surrounding elements, to function

-Suggested to contain float and floating content in containing box (<*div*>), etc. to make handle content scaling easier

Positioning Techniques
-Moving an element out of normal flow to a specified other location. Good for fine tuning specific items after main layout implemented. *position* property based.

Main Types of Positioning
A) *static* - default *position* value. Normal flow standard.

B) *relative* - element's position on page is based relative to its position in the normal flow (including overlap). Leaves empty gap in flow where moved item once was. Example: move up

50px.

-once set element *relative*, move out of normal *static* position with *top, bottom, left, right* properties, which take size values.

C) *absolute* - element moved fully out of normal flow, then fixed to position on <*html*> doc edge (or nearest ancestor edge). Good for overlapping boxes where boxes hidden/shown as desired, hidden/expanding menus, etc.

-Unlike *relative*, leaves no gap behind in flow. Essentially sits on own layer separate from other items flow

-once set element *absolute,* item flows will by default be at top of container that it's *static* position was removed from. Then, push

-set location with *top, bottom, left, right* properties, where properties push element away from those locations (ex. *top: 30px;* would move 30px away from top)

-absolute elements default to last parent container with a position other than *static* and if none, *html*. To create container for absolute items, set container to *relative* position

-*z-index* - property. Sets elements to be under or on top of one another (if overlapping positions). Takes int values only. Greatest int value on top, etc.. Height axis called z-axis.

D) Fixed - Similar to absolute positioning, except element set relative to browser viewport. Ex. persistent nav and header that stays in same place on screen as rest of content scrolls

- *top, bottom, left, right* properties act the same way as absolute. Make sure to set *z-index*

E) Sticky - Acts as static positioned until hits defined offset in viewport, after which *position* is fixed (ex. scroll down to element and element is pulled up as scrolling until hits top of page, then stay at top as other elements scroll up past it)

-also *top, bottom, left, right* properties. When element makes contact with specified distance, it sticks. ex. set item mid-way down page to *stick* when it reaches *top: 30px* (30px from top)

Multi-column Layout

-Set up is similar to flow of newspaper, with columns flowing down and across page. Set num of columns and col width. Aka multicol.

-*column-count* - property. Int values. Num of columns. Sets container to multicol.
-*column-width* - property. Size values. Columns auto-size if only *column-count* specified
-*column-gap* - property. Size values. Gap between columns.

-*column-rule* - property. Size values. Acts as border between columns. Takes up three values: *size, style, color* specified in that order. Is shorthand for three properties. Full-hand:

-*column-ruler-color, column-rule-style, column-rule-width*

Columns & Fragmentation

-Columns often generate where split between columns and text layout is not ideal. Ex. column splits just after a header.

-*break-inside* - property. Controls where columns break. Sets columns not to break element into multiple columns via *avoid* value.

-Ex. Have a multicol container set around a bunch of <*div*>, where each <*div*> has a <*h2*> and a <*p*>. Set *break-inside: avoid*; for <*div*>, which prevents paragraph and header separation

-Also use *page-break-inside* with same values, which does same thing but is older property for older browsers

Media Queries
-Allow you so detect a current media state (ex. browser width), checks a condition, and if true applies specified rules

-*@media typeHere(feature: value) {…}* - types describe general category of device
-ex *@media (max-width: 800px){…}* *//rule applied only to screens <=800px*

-Types
a) *all* - all devices
b) *print* - paged & print preview material
c) *screen*
d) *speech* - speech synthesizer

-Media features - act as condition for *type*. See list here
-Some commonly used features:
-*width* - viewport width -*height* - viewport height
-*orientation* - device orientation (values *portrait, landscape*)

-Logical operators - Can combine queries with logical operators:
-*@media (feature:state) and (feature2:state2) {…}* *//AND*
-*@media (feature:state), (feature2:state2) {…}* *//OR*
-*@media not (feature:state))* *//NOT*

-*NOT* does not apply to rules following comma. Only feature specified for before comma.
-If limiting logical operator query to just one media type: *@media type and (…) and (…) {...}*

-Can also use >, =, <, <=, with *width*, etc. ex. *@media (200px <= width <= 1920px)*

-Can target multi devices at same time. ex. *@media screen, print {...}*

Cross Browser & Older Browser Support
-Grid is new, only since March 2017. IE ended in 2015, so no support. IE global use is now only 2.87%. Edge replaced IE, and has full support for Grid.

-MDN Docs specify supported features on browsers at bottom of page for property, etc.

-Basic way to manage is to create a layout that is in proper order and arranged well in just HTML, so that even if support is lacking, site is still at least, readable

-Can create compatible layouts by setting the containing element (ex. *section*) as a wrapper class (via class name) for a newer layout feature (ex. grid), then giving inner items (ex. *div*) classes for older features (ex. *floats*). If newer feature is supported it will apply, if not, inner will apply.

Fallback Methods
-*display: inline-block;* = to create columns. *In-line* block is ignored if it becomes a flex or grid item
-*display: table* - items set to be table elements lose this as they are set to be flex/grid items
-multi column layouts - " "
-*flex* - fallback for grid. If flex container/element becomes grid, loses flex properties