CSS - Cascading Style sheets

There are styles to code CSS:

- Inline style
- Internal stylesheets: within the HTML code, within the <style> </style>
- Externar stylesheet: style.css

Link HTML to CSS:
** place within the <head>
** Self-closing tag
Rel = relationship with the linked sheet
<link href='https://www.codecademy.com/stylesheets/style.css' rel='stylesheet'>
** if stored in the same file, the href can be './style.css'

Universal = tag all elements of a page → use * as a selector

Type 1, simple →

```
<style>
 h2 {
   color: blue;
 }
</style>
```

Type 2: create a classs, and then call it on your HTML code →

```
<style>
  .blue-text {
    color: blue;
  }
</style>
```

Call it like → `<h2 class="blue-text">CatPhotoApp</h2>`
Note that in your CSS `style` element, class names start with a period. In your HTML elements' class attribute, the class name does not include the period.
** Add several classes to an element like: <h1 class='green bold'>

Important: when styling, start using type selectors, then class selectors, and then ID. since they override eachother in that order, the more specific they are the harder it becomes to edit them in the future.

Select specific elements →
H1.special → only takes h1 elements with the class special. But NOT other elements with the class special

Add style just to descendants within other HTML elements:
<ul class='mainlist'
In CSS → **.mainlist li {}**

Ex: change color of just h4 elements within li elements →

```
Li h4 {
  Color: gold;
}
```

** Elements within a specific class → **element.class {}**

Add the same style to multiple selectors:
** Just separate by a coma

```
H1,
.menu {
  Font-family: Georgia;
}
```

Id elements: You can use an id to style a single element → Id will be applied over class element
Write it in style like #cat-photo-element → call it like id="cat-photo-element"

Attribute Selector: more
This selector matches and styles elements with a specific attribute value.
For example, the below code changes the margins of all elements with the attribute type and a corresponding value of radio:

```
[type='radio'] {
  margin: 20px 0px 20px 0px;
}
```

Or

```
[href] {
   Color: magenta;
}
```

Target specific elements, like each img →

```
img[src*='winter'] {
  height: 50px;
}
```

→ this targets images which have 'winter' on their src

```
<img src='/images/seasons/cold/winter.jpg'>
```

There are several unit options to size elements like the absolutes: in and mm, or the relatives: em (for font) or rem.

⇒ IDs override the styles of types and classes.Since IDs override these styles, they should be used sparingly and only on elements that need to always appear the same.

** classes can override the body elements.
** if an element has 2 classes, it will show the latest declaration on the style. So if .blue is after .pink on the style section, the text will be blue.
** Id attribute takes precedence over any classes
** In-line style can override Id

Override all CSS with important → `!important;` → write after color name

Inherit CSS Variables:
If you write a variable in the :root {}, all descendants inherit the value.
** Override in a specific element by

Pseudo-Class:
To change the appearance of an element depending on an interaction.

- :focus
- :visited
- :disabled
- :active

- :hover
- :link

Ex: change color when hovering mouse →

```
p:hover {
  background-color: lime;
}
```

** Can also set a css for elements in their active state vs. not active.

- Change the appearance of the pointer:

```
a {
cursor: pointer;
}
```

** Ease the color of the button hover like: `transition: color .1s ease-in, background-color .1s ease-in;`

FONTS:
Use Web Safe Fonts, to make sure your selected fonts show everywhere.

Size = `font-size: 16px;`

Family → either name one of the basic ones or add the URL for a Google font

1. `font-family: monospace;`
2. Must add the font at the top of code, before style → `<link href="https://fonts.googleapis.com/css?family=Lobster" rel="stylesheet">`
   And then, call the font → `font-family: 'FAMILY_NAME', GENERIC_NAME;`
   ** Generic name is in case it doesn't work it takes the generic font. → `font-family: 'Roboto', sans-serif;`
3. You can also download the fonts to the computer and add them using @font-face ruleset. CHECK SYNTAX IN THE LINK.

Font-Weight: normal, bold, lighter, bolder. → can also use numbers: 1 to 1000, in increments of 100

Font-style: italic, normal

Text layout:

- **Letter-spacing**: 2px; → Or 0.5em
- **Word-spacing**: 0.3em;

- **Line-height:** 1.4;
- **Text-align**: left, centerm right, justify (of it's parent element).
- **Text-Transform:** uppercase, lowercase
-

COLOR AND BACKGROUND:

- Color = styles foreground color
- Background-color

Background color =
```
.green-background {
  background-color: green;
}
```

Color:
Use either color name like red; or use hex color like #FF0000, can also write it for short as #F00.
Use RGB like → **rgb(255, 165, 0);**

Opacity: from 0 to 1 (where 1= 100%)

Hexadecimal: #FFFFFF or #FFF

HSL: Hue, Saturation, and Lightness:
Hue: up to 360
Saturation and Lightness: in percentage.
→ **Color: hsl(120, 60%, 70%);**

Add opacity to HSL → HSLA
Color: hsla(34, 100%, 50%, 0.1); → las value is alpha or opacity. From 0 to 1
** Can add to hex or rdb by adding two digits at the end of hex (from 00 to FF which is opaque).
*8Color transparent or rgb(0, 0, 0, 0)

IMAGE:
Background-image: url('link to image here'); → or use link to folder where it saved like 'images/mountains.jpeg'

Search image when it's inside folder → ./image path here
Search image when it's outside the current folder → ../image path here

Center image (if it's showing just a part of it):
Background-size: cover;

Width = `width: 100px;`

Border =

```
.thin-red-border {
    border-color: red;
    border-width: 5px;
    border-style: solid;
  }
```

Add 2 or more classes to an element like = `class="smaller-image thick-green-border"`

Round image borders with → `border-radius`. → either in pixels (px) or %

Padding: can be the same on all sides, or specify padding-right, left, bottom, and top for each. Write like padding: 10px, 20px 10px 20px

Border:

Margin: if negative, the box will fill the whole space. Can also have specific margin for each side like padding (specify all in one like → margin: 10px, 20px 10px 20px; → order is top, right, bottom, left)

Create CSS Variable at the top of style, can use it after in the code.
Ex: `--penguin-skin: gray;`
Call it: `background: var(--penguin-skin);`

Fallback Value: value to which the browser goes abc to if the variable is invalid. **Great for debugging

```
background: var(--pengiun-skin, black);
```

Visual Studio Code:

- Always call the home page → index.html

Image shadow effect
Filter: drop-shadow(1px 1px 1px black);

Static menu bar
Add this to the nav bar →

```
position: fixed;
   top: 0%;
```

+ Add a background color if you want.

Display avatar in line with text:
Display: in-line-block;

Align vertically to text:
Vertical-align: middle;

Typography:

Fonts: use ‘quoatations’
Use fallback fonts: Carlson, Georgia, ‘Times New Roman’

CSS:
The Box Model:

1. `width` and `height`: The width and height of the content area.
2. `min-height` — this property ensures a minimum height for an element’s box.
3. `max-height` — this property ensures a maximum height of an element’s box.
4. `padding`: The amount of space between the content area and the border.
    - When 4 values: top, right, bottom, left.
    - When 3 values: top, left & tight, bottom
    - When 2 values: top & bottom, left & right.
5. `border`: The thickness and style of the border surrounding the content area and padding. → can have width (thin, medium, or thick), style (none, doted, solid, and more), color (check all 140) → Ex: `border: 3px solid coral;`
6. `Border-radius:` make it squared or a circle. → Can write it as 5px or 50%
7. `margin`: The amount of space between the border and the outside edge of the element. (margins in adjacent elements is summed up, in vertical the lesser one collapses).
8. `margin: 0 auto;` will center the divs in their containing elements. The 0 sets the top and bottom margins to 0 pixels. The `auto` value instructs the browser to adjust the left and right margins until the element is centered within its containing element.
9. `overflow` property controls what happens to content that spills, or overflows, outside its box. More info here:

- `hidden`—when set to this value, any content that overflows will be hidden from view.
- `scroll`—when set to this value, a scrollbar will be added to the element's box so that the rest of the content can be viewed by scrolling.
- `visible`—when set to this value, the overflow content will be displayed outside of the containing element. Note, this is the default value.

10. `Visibility -` Elements can be hidden from view → values: hidden, visible, collapse. (doesn;t show, but the space for it does. To make it disappear enirely use display: none;)
11. `Box-sizing: border-box`. → reset the entire box model and specify a new one.

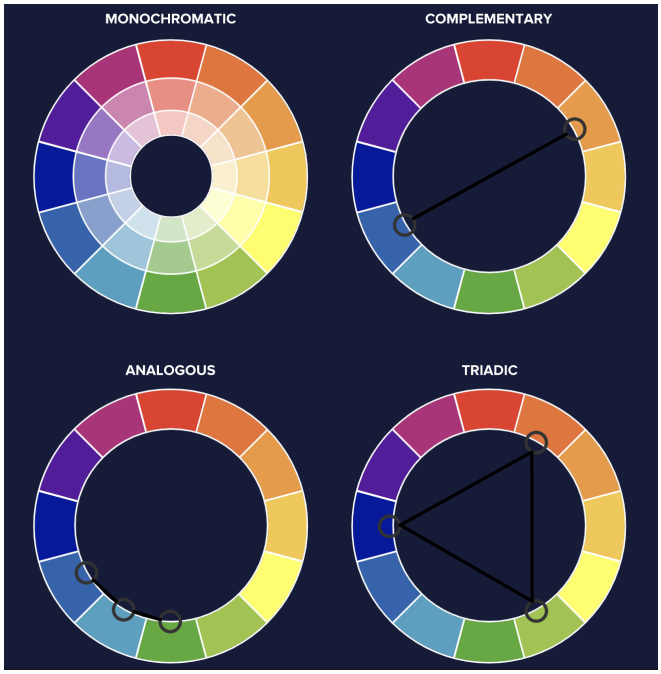
Edit position of elements:

1. `Position →` 5 values: static (default), relative, absolute, fixed, and sticky. → ** add top: 20px, bottom, left, or right with pixels. To upset position.
2. `Display →` values: inline, block, and inline-block.
3. `Z-index →` property controls how far back or how far forward an element should appear on the web page when elements overlap. **So elements don't overlap. Puts the elements with the z-index to the front (values from 1 to 10)
4. `Float is` commonly used for wrapping text around an image. ** values: left, right
5. `Clear →` specifies how elements should behave when they bump into each other on the page. **Values: left, right, both, none.

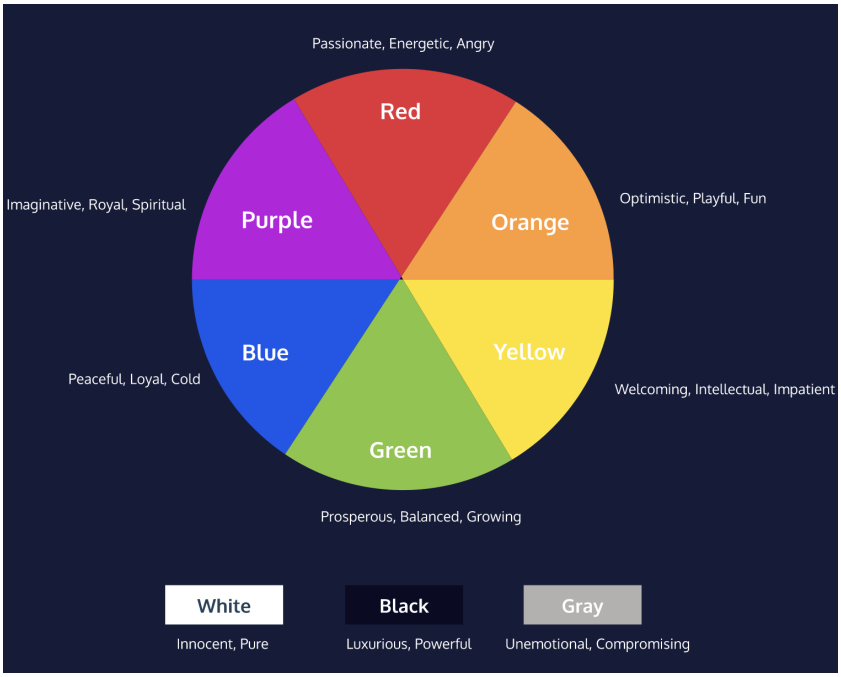
COLORS:

Create a palette thinking in how to put colors together:

Go to Adobe to match!



Color meanings:



UI:

Find the best colorL https://color.cloudflare.design/ → really nice!

Check out brands' colors; https://brandcolors.net/

Create good color palettes: http://colorsafe.co/

Check out color contrasts: https://webaim.org/resources/contrastchecker/

The keys to designing for UI:

- Select a dominant brand color and supporting accent colors
- Use contrast to define sections and differentiate actions
- Use semantic colors for error and success messages
- Incorporate default colors for text and backgrounds where needed
- Neutral colors can provide good contrast
- Keep users in mind when selecting color

Breadcrumbs:

It's the path that users expect to take to find something on a website (clothes - shirts - tshirts)

- Usually displayed next to each other (display: inline;)
- Separated by a symbol (> or /) → Ex:

```
.breadcrumb li+li::before {
  content: "/";
}
```

- Should NOT be underlined
- Should change color when hover over them.

Breadcrumb Styles:

1. Used to add cosmetic content like an emoticon or quotes before and after an HTML element.

```
.breadcrumb li a::before, .breadcrumb li a::after {
  content: "";
  position: absolute;
```

2. Also used to create forms, like making a rectangle box an arrow.

```
.breadcrumb li a::before {
  left: -10px;
  border-left-color: transparent;
```

```
}
```

Note: if you add a :hover action after a breadcrumb, you need to add it to the ::before and ::after CSS elements too (independently).

Responsive Web-Design:

70+ resources for creating liquid and elastic layouts

Media queries explained, and all main types. MDN

TO START: create this set up so everything works nicely. Follow viewport (set width of viewport in HTML element), browser reset, border box, and breakpoints.

Min and Max-Width:
Min-width property sets a minimum browser or screen width that a certain set of styles (or separate style sheet) would apply to. If anything is below this limit, the style sheet link or styles will be ignored.
The **max-width** property does just the opposite. Anything above the maximum browser or screen width specified would not apply to the respective media query.

For ipad: property called orientation (portrait or landscape)

****Min-width and max-width** are also used so an element doesn't stretch or reduce itself too much. (same for min and max height) set it as **max-width: 100% height: auto;** (or the reverse)

Great example to scale images and videos proportionally: - worth memorizing -

```
.container {
  width: 50%;
  height: 200px;
  overflow: hidden;
}

.container img {
  max-width: 100%;
  height: auto;
  display: block;
}
```

Hide elements: display: none; → works so elements from the mobile version don't show on desktop and vice-versa.

Font-Size: em and rem →

- **Em:** If you use font-size: 1em → it multiplies the previous font-size specified in pixels by the em value. If you haven't it multiplies 16px by the em value.
- **Rem**: doesn't check the value of the parent element, but of the root element. So the h1, checks for the font-size declared in html. And multiplies it so if in html 20px, in h1 2rem, it is 40px.

Percentages:
Instead of pixels, you can use % for width, height, padding, border, and margins.
** when used in a subelement, it refers to ex: the 50% of the width declared on it's parent css declaration →

```
.main {
  height: 300px;
  width: 500px;
}

.main .subsection {
  height: 50%;
  width: 50%;
}
```

**Vertical padding and margin are based on the width of the parent, and not the height (because it stretches as you change the child's height)

Background images: properties

```
  background-image: url('#');
  background-repeat: no-repeat; **because images repeat by default.
  background-position: center;
  background-size: cover;
```

Media Queries:

```
@media only screen and (max-width: 480px) {
  body {
    font-size: 12px;
  }
}
```

1. `@media` — This keyword begins a media query rule and instructs the CSS compiler on how to parse the rest of the rule.

2. `only screen` — Indicates what types of devices should use this rule. **Always use only screen.
3. You can define min and max width in 1 query or create 2. Ex of two in 1 query: `@media only screen and (min-width: 320px) and (max-width: 480px) {}`

*See queries in chrome dev. Tools here's how!.

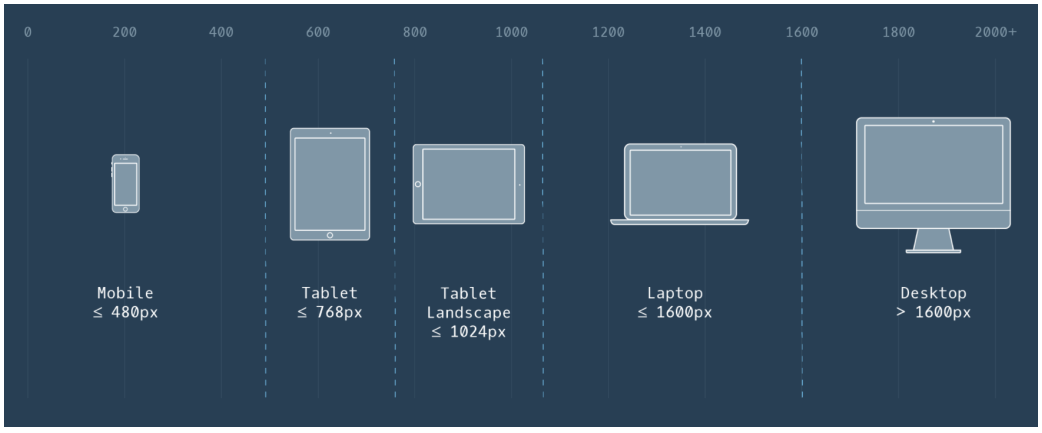
Dots per Inch (DPI):
Targets resolution, so the full resolution loads when users has a high-resolution display.
Min-resolution and max-resolution
`@media only screen and (min-resolution: 300dpi)`

Add several feature to one query:
- Can apply two rules that must be met like by separating them with **"and"**
- If only one of the rules must be met to apply, separate the features with a **coma ,**

Orientation class → can be landscape or portrait.

Breakpoints: the points at which media queries are set.
The best practice is to resize your browser to view where the website naturally breaks based on its content. The dimensions at which the layout breaks or looks odd become your media query breakpoints. Within those breakpoints, we can adjust the CSS to make the page resize and reorganize.

Average screen sizes:



FLEXBOX:
A flex container is an element on a page that contains flex items. All direct child elements of a flex container are flex items.

To designate an element as a flex container, set the element's `display` property to **`flex` or `inline-flex`.** Once an item is a flex container, there are several properties we can use to specify how its children behave. (flex containers remain lock level, but flex items don't)
Inline flex: display next to each other.
Element will shrink by default if the parent is smaller, but not grow if it's larger.

1. `Justify-content:` to one site, 5 typical: flex-start, flex-end, center, space-around, space-between.
2. `Align-items:` space flex items vertically. 5 typical: flex-start, flex-end, center, baseline, stretch.
3. `Flex-grow:` **declared on flex items not containers. It works so the item grows if there is more space to be filled in the parent element on larger screens → flex-grow: 1; (can be 1, 2, etc.)
4. `Flex-shrink:` elements that shrink with creen changes. Default shrink is 1. **max and min widths take precedence over flex-grow and flex-shrink. → flex-shrink: 2;
5. `Flex-basis:` allows us to specify the width of an item before it stretches or shrinks. Flex-basis: 60px;
6. `Flex:` different from the display property. Allows you to declare `flex-grow`, `flex-shrink`, and `flex-basis` all in one line.
    - Declare flex-grow, flex-shrink, and flex-basis (in that order): `flex: 2 1 150px;`
    - Declare flex-grow and flex-shrink: `flex: 2 1;`
    - Declare flex-grow and flex-basis: `flex: 1 20px;`
7. `Flex-wrap:` let items move to the next line when they don't fit. **Values**: wrap, wrap-reverse, nowrap (default, only used to override).
8. `Align-content:` space rows of content from top to bottom**. Values;** flex-start, flex-end, center, space- between, space-around, stretch. **Used on flex container (display: flex;)
9. `flex-direction:`The main axis and cross axis are interchangeable., switch them using this property. Values: row (default), row-reverse, column, column-reverse.
10. `Flex-flow:` used to declare both the `flex-wrap` and `flex-direction` properties in one line. `flex-flow: column wrap;`

Nested flexboxes: can position flex containers inside of one another.