|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | **Cues**  NoteGem Horizontal Line    What does CSS stand for?        How to write background image css tag?    How to write background size css tag?  What are the options to repeat background?      How does specificity work?        What is the box model?  What's the content area?                What's the padding area?      What's the border area?                What's the margin area?  What is content?  What is padding?  What is border?  What is margin?              How do you define border in css?        What are the different options for the display css tag and what do they do?                      What are the options for box sizing?                                      What does a css reset do?          What does the fix do?        How do display block and flex look on a page?                      What are the axes in flex direction?  What are the options for flex direction?  What does flex direction do?      What is justify content and what are its options?    What is space between?  What is space around?          What does align items do and what are its options?    What do each of the align-items options do?                        What is align-self?                          What is flex wrap?  What are its options?          What is the syntax for flex-flow?          What does align-content do?        What is safe versus unsafe?                  What is flex grow?                What is flex shrink?        What is flex basis?                What are options and syntax for flex css tag?          Why use an autoprefixer?          What are considerations when designing a website layout for large audiences?              What are the types of web design layouts?      What are some pros and cons of each?                                        What are correct css breakpoints?                What is responsive?            What are media queries for?          What is adaptive?              What does mobile first mean?            What are px?  When should you use them?                    What is em?  When should you use it?              What is rem?  When should you use it?                What are percents?  When should you use them?                    What is vh/vw?  When should you use it? | **Notes**  NoteGem Horizontal Line    **CSS:** Cascading Style Sheet, reads from top down    **http**: Hyper Text Markup Language    **background-image**: url('../images/img.jpg'); (two dots goes up 1 dir)    **background-size:**    **background-repeat**: repeat background or not    Call parent element first, then tag inside    **CSS specificity**:   * p.warning = more specific than just p * More specificity means it will have precedent for styling     \*\*INSERT PICTURE OF BOX MODEL    **content area** contains "real" content of element    content width x content height    if box sizing = content-box (default) & element is block element, content area can be explicitly defined with width, height, min-/max-width, min-/max-height    **padding area** = padding - box width and height thickness determined by padding top right bottom left    **border area** = border-box width x height  thickness determined by border top right bottom left    if box-sizing set to border-box, size can be defined within width, height, min-/max-width, min-/max-height    when background extends to outer edge of border in z order, you can change with background-clip property    **margin area:** same as previous only with word "margin"  **content** = where text and images are rendered  **padding** = space between content and border  **border** = border that goes around padding and content  **margin** = space between border and everything else on the page    using **"box-sizing: border-box"** will constrain padding inside border's boundaries set by width and height  otherwise padding is added outside as opposed to within    **border**: border-width border-style border color;   * border: 1px solid red;     **margin** creates distance between element and rest of the page  **"display: none;"** removes element from document, not the same as "visibility: hidden;"    **"display: inline;"** nested inside other elements without disrupting the content flow     * height/width properties have no effect * only takes up as much space as its content * can't do block elements * position vertical: vertical-align * position horizontal: text-align     box sizing has three possible values: content-box, padding-box, border-box    width or height + padding + border = actual width or height    "display: block;" takes up as much space as parent allows     * need width and height to control size * will break content flow on new line * examples: div or p * can rest other block or inline elements in block * horizontal center = "margin: auto;" * vertical centering tricky without flexbox     "display: inline-block;" has default behavior of inline element with the added bonus of controlling size with width/height   * allows elements to stack next to each other     **CSS reset will remove all default styling provided by browsers (box model/font properties to Oneinherit)**   * **wipes browser styles clean** * **Eric Meyer, Normalize (makes browser styles consistent)**     **universal box sizing:**  **\*,\*: before, \*: after {}**  **box-sizing: border-box;**    **pro tip:**  put border-box and border in \* CSS while designing  plan B for design/set-up -- draw boxes on paper and label things    **reset with box sizing:** border-box, margin/padding 0, max-width 100%, border 1px solid red;    **\*\*\*INSERT PICTURE OF DISPLAY BLOCK & FLEX**    **flex affects children**    **\*\*\*INSERT PICTURE OF AXES**    By default, items arranged along main axis from left to right    **flex-direction =** rotates axis  **flex-direction:** column/row or with -reverse    **justify-content**: how you align items on main axis     * flex-start/end, center, space-between/around * **space-between** = equal space between each square, not between it & container * **space-around** = equal space on each side of the square     flex only goes one level deep    **align-items**: applies to cross-axis     * flex-start/end, center, stretch, baseline, same as justify-content     **stretch** - height must be set to auto or it will override     * items take up entirety of cross-axis     **baseline** = bottom of p tags are aligned     * if no p tags, bottom of squares are aligned     **row** = horizontal main axis | **column** = vertical main axis    **align-self** = ailgnment of one particular element     * overrides align-items for one square * defaults to auto     flexbox has spotty IE support, not supported below IE11, else you have to write extra code    **display:** flex or inline-flex enables flex content for container     * nests only one level deep     **flex-wrap =** default, items will appear on one line     * to change, use this property * default -->nowrap; all flex items on one line * **wrap =** all on multiple lines, top to bottom * wrap-reverse     **flex-flow =** shorthand for -direction -wrap properties     * flex-flow: direction flow;     **align-content =** aligns flex container's lines when there is only one line of flex items     * same options as align-items     **safe/unsafe =** how to handle content that becomes inaccessible  properties for flex-items children    **order =** default is flex items in source order     * controls order of their appearance * order: #; default 0     **flex-grow =** dictates what amount of available space item should take up in proportion to other items within flex parent    if all items but one = one + one = two, the one with value two will take up two times space over others    negative numbers invalid    **flex-shrink =** defines ability for flex item to shrink     * works like flex-grow     **flex-basis =** defines default size of element before remaining space is distributed     * can be length or a keyword * **auto** means space distributed based on flex-grow * **O** = space around content not factored in * **content** = size based on item's content     **flex =** shorthand for flex-grow, -shrink, -basis combined     * flex: flex-grow flex-shrink flex-basis * default 0 1 auto (shrink/basis optional)     to handle the most browsers possible, run css thru **autoprefixer**     * css-tricks.com/autoprefixer/     when designing a website layout for large audiences, consider the following potential differences:     * screen resolution * browser choice * browser maximized or no * extra toolbars in browser * operating system and hardware     **Responsive web design layouts**     * **fixed =** early web, good on desktop, worse smaller vport      * much cheaper to build and maintain * CSS widths usually hard-coded pixels * no media queries * layout does not move * quicker deployment * sometimes bad with rule of thirds      * **fluid =** opposite of fixed, can expand and contract in proportion to device      * liquid design, often divided between desktop/tablet/phone * percent-based units instead of pixels must equal 100% * designed to proportionally shrink * no media queries      * **problems:**    + images get too small   + buttons grow too large   + layouts to spacious on desktop     **breakpoints resolution:** 600 900 1200 1800px    **responsive =**     * combines features of fixed, feature, adaptive layouts * take more time and effort * often divided between desktop, tablet, phone * responsive units used * yes media queries * accommodates thousands of devices * example: can use max-width to control max width of container while also providing percent flexibility below it (also min-width)     media queries used to constrain responsive units so that it functions across many devices and screen sizes    **adaptive =** borrow from speed of fixed layout but accommodates devices at fixed breakpoints     * design divided between phone, tablet, desktop * yes media queries * layout widths hard-coded at each media query * quick deployment for specific devices     **mobile first** means you design and code for mobile layouts first and expand layout/features as you gain screen space towards desktop    can specify fallback unit {font-size: 14px; font-size: 1.4rem;}    **px =** pixels, absolute length unit     * does not adjust for surroundings * easiest unit to read from design files * modern browsers allow pixels to grow * no reliance on cascade for styling * not responsive in layouts * hard-coded font size values, not accessible for users that change default settings     **em =** relative length unit     * will adjust to surroundings * can be too flexible and create cascading font sizes that were unexpected (size compounds with nested) * if font size = 10px & h1 font size = 2em then h1 font size = 20 px     **rem =** root em unit     * relative length unit * unit looks to root element in a page * use 62.5% as baseline font size so that 1rem = 10px throughout document * can scale whole project by changing root font sizes * html {font-size: 62.5%}     **percent =** relative length unit     * usually for building responsive layouts, more than font sizing * start at viewpoint width and cascade down through nested units * avoid going over 100% * can get too small very easily * font percent everything vh hero img * rem     **vw/vh -** completely reliant on viewport to render     * do not use except in full-screen layouts * height 100% = 100% vw/vh good * rarely used to make font fill screen, sized 33vh/one-third of viewport   + example slideshows or landing page headers     use rem for sizes and spacing  use ems for media queries  do not use px if possible    when you do media queries and font sizes, make sure max-width and min-width do not overlap (example max-width 63.99em min-width 64em)     * will take longer, adds complexity, increases file sizes | |
| |  | | --- | | **Summary** | |