|  |  |
| --- | --- |
| **Cue Column**   * 1. Overview * 2. Prepare * 3. Flex model * 4. Columns vs rows * 5. Wrapping * 6. Flex-flow shorthand * 7. Flexible sizing * 8. Shorthand vs longhand * 9. Horizontal and vertical alignment * 10. Ordering flex items * 11. Nested boxes * GRIDS * 1. Overview * 2. Grid Layout * 3. Grid placement * CSS LAYOUTS | **Notes Column**  1a. flex CSS simplifies design layouts in horizontal and vertical layouts  1b. Items fill/shrink according to spaces  1c. Flexbot is better for apps and small scale layouts  1d. Grid is intended for larger layouts  1e. When to use-  1ei. Vertically centering a block of content inside its parent  1eii. Making all items take up an equal amount of space regardless of how much space is available  1eiii. Make all columns the same height regardless of content  2a. Flexbot is a layout method for arranging items in rows or columns to adapt to a given space.  3a. main axis- direction the items are laid out in  3b. main start- where items start  3c. main end- where items end  3d. main size- length from start to end  3e. cross axis- runs perpendicular to item layout  3f. cross start- same as main start but perpendicular  3g. cross end- same as main end but perpendicular  3h. cross size- length from cross start to cross end  3i. Flex container- item that has display: flex;  3j. flex items- items in container being arranged  Three flex items in a left-to-right language are laid out side-by-side in a flex container. The main axis — the axis of the flex container in the direction in which the flex items are laid out — is horizontal. The ends of the axis are main-start and main-end and are on the left and right respectively. The cross axis is vertical; perpendicular to the main axis. The cross-start and cross-end are at the top and bottom respectively. The length of the flex item along the main axis, in this case, the width, is called the main size, and the length of the flex item along the cross axis, in this case, the height, is called the cross size.  4a. flex-direction- which direction main axis runs. Row is preset. Change to column; for columns  5a. to help eliminate horizontal scroll- use flex-wrap  5b. flex-wrap: wrap; & flex-direction: row reverse; makes it show up on the other side of the page  6a. substitute flex-direction and flex-wrap with flex-flow  Flex-direction: row;  Flex-wrap: wrap; 🡪  Flex-flow: row wrap;  7a. you can determine item size rations using flex: 1 (optional px size ex. 100px); for specific article use article:nth-of-type(#) {flex: (ratio, ex. 2) (optional pixel size, ex. 100px)}  7b. must have a minimum of 100px  8a. can be written three different ways   * Flex-grow- (unitless proportion value found in 7.) * Flex-shrink- (unitless proportion- how much an item will shrint in order to prevent overflow) * Flex-basis- minimum size discussed above   8b. don’t use unless understood and necessary  9a. “align-items” property   * Default is normal which is the same as stretch in flexbot * Center value makes items maintain their intrinsic dimensions but be centered across the cross axis * Flex-start, self-start or start and flex-end, self-end or end will align items on the cross axis respectively * Baseline- “the bottom of each flex items first line of text will be lined up with the bottom of the first line of the element with the greatest distance between the cross start and that baseline”   9b. you can override align-items with align-self  9c. “Justify-content” controls where items sit on main axis   * Default value is normal, which behaves as start (all items sit at start of main axis” * “End” or “flex-end” make items sit at end * “left” and “right” “start” and “end” * “Center” = “justify-content” items sit middle of main axis * “space-around” distributes items evenly * “space-between” similar to “space-around” but doesn’t leave any space at either end * Justfy-items is ignored in flexbot layouts   10a. can change layout order without adjusting code  10b. “order: 1;”  10c. changing order can negatively effect keyboard user because tabbing code remains the same  11a. flex item can be a flex container  11b. to do that, make sure you call item correctly  -----------------------------------------------------------------------------------------  1a. Grid is a 2D layout system to make complex layouts straight forward  1b. parent elements are given a grid layout as display: grid; immediate children respond but not grand children and beyond  2a.  explicit- manually placed  Implicit- automatic order  2b. grid-template-columns🡪 however many length columns we give it  Rows wrap around automatically (created implicity)  2c. grid gap to areas between columns and rows (grid-gap)  2d. fr = fractional unit of percentages- only takes whats available  1fr 1fr 1fr equals one third each of whole. Responsive to screen size  Repeat,  2e. You can nest grids  3a. children can be placed in single places or in more than one place  3b. you can see grid in the developer tools  3c. grid-column-start: 1, grid-column-end: 3 where item is to start and end grid-column 1/3  3d. pay attention to stacking order- z-index 0 default, positive is forward, negative back  3e. to center child item of grid vertically- align-self: center  -----------------------------------------------------------------------------------------  1a. |
| Time:  2:47- 5:02  Activity- 5:05-5:32 | **Summary**   * Flex is appropriate to use with small scale layouts * Flex lines items up to a main and cross axis * Items can be flexed and nested with in eachother- just need to use right call |