A. Responsive web design
    a. Definition
        i. responds to the needs of the users and the devices they're using
        ii. layout changes based on size/capabilities
        iii. Examples
            1. http://www.northeastern.edu
            2. https://www.bostonglobe.com/
            3. http://www.wired.com
    b. Responsive vs. adaptive
        i. Both change appearance based on the browser environment (usually browser width)
        ii. Responsive
            1. Respond to the size of the browser at any given point
            2. No matter browser width, the site adjusts its layout optimized to the screen
        iii. Adaptive
            1. Adapt to the width of the browser at a specific points
                a. On phone, users might see single column view
                b. On tablet, users might see two columns
            2. Traditionally what we mean re: responsive
        iv. Smooth vs. snap
            1. https://css-tricks.com/the-difference-between-responsive-and-adaptive-design/
        v. Examples
B. Mobile first philosophy
    a. Develop for the smallest mobile device first, then progressively enhance the experience as more screen real estate becomes available
        i. Mobile development = hardest, most limitations (eg screen size and bandwidth)
        ii. Do mobile first!
        iii. Smallest screens = only most crucial features
            1. Gives you a chance to reconsider “extraneous” features
            2. Do they need to be included on desktop?
C. Breakpoints
    a. a point that allows for the provision of the best possible layout that enables users to consume or understand your site's content for their current viewport size
    b. Mobile first: start small, work up
        i. As you expand that view, there will be a point at which the layout looks terrible
        ii. that’s where you add a breakpoint!
    c. Create breakpoints based on content, not devices
        i. Devices change, responsive/adaptive development should ideally work across all
        ii. Breakpoints are easier to work around content than layout/design
    d. Keep lines of text to max 70-80 chars
        i. Optimize for reading
        ii. As columns become too wide for comfortable reading/display, add breakpoints
        iii. Adaptive font sizing (ems)
    e. Don’t hide content!
        i. If content doesn’t fit at certain breakpoints, is it necessary?
        ii. Data load
    f. Don’t target devices!
        i. Set breakpoints wherever the presentation of the content degrades instead of specific device widths

        ii. This covers any device that comes along
        iii. Future proofing
    g. “Progressive enhancement”
        i. Build from bottom up
        ii. Mobile first = content first
            1. Developing within mobile parameters forces you to prioritize content
            2. Content focus = user focus
    h. “graceful degradation”
        i. build from top down
        ii. Problem w/graceful degradation
            1. Starting with the all-inclusive design = the core and supplementary elements merge
            2. Harder to distinguish and separate
            3. Treating mobile design as an afterthought -> “cutting down” the experience

D. Flexbox
    a. https://css-tricks.com/snippets/css/a-guide-to-flexbox/
    b. more efficient way to lay out, align and distribute space among items in a container especially when their size is unknown and/or dynamic
    c. give the container the ability to alter children width/height/order to best fill the available space
        i. expands items to fill available free space
        ii. shrinks items to prevent overflow
    d. Direction agnostic (can respond to direction changes, dir attribute from week 3)
    e. Most appropriate to application components and small-scale layouts
        i. grid layout is intended for larger scale layouts
    f. Flex container (parent)
        i. Display: flex
            1. Creates flex context for children (direct children only)
            2. Example + hands-on
        ii. Flex-direction:
            1. main axis, direction items are placed in container
            2. flex items mostly lay in horizontal rows or vertical columns
            3. row/row-reverse
            4. column/column-reverse
            5. Example + hands-on
        iii. Flex-wrap
            1. By default, values will all try to fit along axis, but wrapping can be set
            2. Nowrap (default)
            3. Wrap/wrap-reverse
            4. Example + hands-on
        iv. Flex-flow: shortcut for flex-direction + flex-wrap
            1. Eg flex-flow: row wrap;
            2. Example + hands-on
        v. Justify-content
            1. Defines alignment along the main axis
                a. Along horizontal line for row flow
                b. Along vertical line for column flow
            2. Flex-start: clustered towards beginning of axis
            3. flex-end: clustered towards end of axis

            4. Center: centered in middle of axis
            5. Space-between: items are evenly distributed
                a. first item is on the start axis
                b. last item on the end axis
            6. Space-around
                a. items are evenly distributed in the line with equal space around them
                b. visually the spaces aren't equal, since all the items have equal space on both sides
            7. Space-evenly
                a. items are distributed so that the spacing between any two items (and the space to the edges) is equal
            8. Example + hands-on
        vi. Align-items
            1. Defines alignment at cross axis to main axis
                a. Along vertical line for row flow
                b. Along horizontal line for column flow
            2. flex-start: cross-start margin edge of the items is placed on the cross-start line
            3. flex-end: cross-end margin edge of the items is placed on the cross-end line
            4. center: items are centered in the cross-axis
            5. baseline: items are aligned such as their text baselines align
            6. stretch (default): stretch to fill the container
            7. Example + hands-on
        vii. Align-content
            1. Aligns lines within when there is extra space in the cross-axis
            2. flex-start: lines packed to the start of the container
            3. flex-end: lines packed to the end of the container
            4. center: lines packed to the center of the container
            5. space-between: lines evenly distributed; the first line is at the start of the container while the last one is at the end
            6. space-around: lines evenly distributed with equal space around each line
            7. stretch (default): lines stretch to take up the remaining space
    g. Flex children
        i. Order
            1. laid out in the source order by default
            2. Order can be specified by integer
        ii. Flex-grow/flex-shrink
            1. https://cssreference.io/property/flex-grow/
            2. https://cssreference.io/property/flex-shrink/
            3. ability for a flex item to grow/shrink if necessary
            4. takes unitless value that serves as a proportion relative to other flex children
            5. dictates what amount of the available space inside the flex container the item should take up
            6. Flex-grow: how much of the remaining space in the flex container should be assigned to that item
                a. The remaining space is the size of the flex container minus the size of all flex items together
            7. Flex-shrink: determines how much the flex item will shrink relative to the rest of the flex items in the flex container when there isn't enough space on the row

                a. If the size of all flex items is larger than the flex container, items shrink to fit according to flex-shrink

    iii. Flex-basis
        1. Initial size of flex item before space is distributed according to the flex factors
        2. If omitted, specified value= 0
        3. Length
            a. Length 0 = extra space around content isn't factored in
        4. Auto: "use my width or height property" (depending on flow)
    iv. Flex
        1. Shorthand for flex-grow (+ flex-shrink + flex-basis) [shrink/basis optional]
        2. Eg flex: 1; flex: 0 1 auto (default); flex: 2 1 auto;
    v. Align-self
        1. Allows the default alignment (or the one specified by align-items) to be overridden for individual flex items

E. Grid
- a. https://css-tricks.com/snippets/css/complete-guide-grid/
- b. two-dimensional grid-based layout system
- c. Good for dividing a page into major regions or defining the relationship in terms of size, position, and layer, between parts of a control built from HTML tags
- d. Picture newspaper layout with rows and columns
- e. grid layout enables an author to align elements into columns and rows
    - i. NOT table
    - ii. Remember table is for tabular data that corresponds to the intersection of a row and column
- f. Definitions
    - i. Grid container (parent)
        1. display: grid
            - a. direct parent of all the grid items
            - b. Creates grid context for children (direct children only)
            - c. Grids can be nested by making children grid containers w/ display: grid
    - ii. grid line
        1. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout#Grid_lines
        2. dividing lines that make up the structure of the grid
        3. vertical ("column grid lines") or horizontal ("row grid lines")
        4. Located in between rows or columns, or around the outside
        5. numbered according to the writing mode of the document
            - a. In a left-to-right language, line 1 is on the left-hand side of the grid, in r-t-l, line 1 is on the right-hand side
    - iii. grid track
        1. space between two adjacent grid lines
        2. the columns or rows of the grid
        3. Tracks can be defined using any length unit (ie, rem, em, px, %, etc.)
        4. New unit = fr
            - a. fraction of the available space in the grid container
            - b. Eg = 1fr 1fr 1fr
            - c. each track = 3 equal width tracks that fill available space
    - iv. Grid area

                1. elements spanning one or more cells by row or by column
                    a. must be rectangular – can't create an L-shaped area, for example
        v. Grid cell
            1. single "unit" box of the grid
            2. smallest unit on a grid

    g. Explicit vs. implicit grid
        i. Rows and columns defined with grid-template-columns = explicit grid
        ii. Rows and columns created when content goes beyond bounds of explicit = implicit grid
        iii. Eg defined our column tracks with the grid-template-columns property, but the grid also created rows on its own
    h. repeat()
        i. grids with many tracks can use repeat()
        ii. repeat all or a section of the track listing
            1. Eg 1fr 1fr 1fr = repeat(3, 1fr)
    i. minmax()
        i. Way of defining how tracks (rows or columns can expand/collapse and never go under/over a certain size.
            1. Eg. may want to give tracks a minimum size, but also ensure they expand to fit any content that is added
        ii. minmax(100, auto) = may want my rows to never collapse smaller than 100 pixels, but if my content stretches to 300 pixels in height, then I would like the row to stretch to that height
            1. Auto = the size will look at the content size and will stretch to give space for the tallest item in a cell, in this row
    j. Gutters
        i. Spaces between grid cells
        ii. Created with column-gap & row-gap properties

F. Difference between flexbox and grid
    a. flexbox was designed for layout in one dimension - either a row or a column
        i. works from the content out.
        ii. Good use case for flexbox is when you have a set of items and want to space them out evenly in a container
        iii. let the size of the content decide how much individual space each item takes up
        iv. If the items wrap onto a new line, they will work out their spacing based on their size and the available space on that line
    b. grid was designed for two-dimensional layout - rows and columns at the same time
        i. works from the layout in
        ii. create a layout and then you place items into it, or you allow the auto-placement rules to place the items into the grid cells according to that strict grid