A. Responsive web design
    a. A method of creating a website layout that responds to the needs of the users and the devices they're using
    b. The layout changes based on size/capabilities
    c. This approach eliminates the distinction between the mobile version of your website and the desktop version
        i. They’re the same thing!
        ii. No maintaining two separate versions
    d. Examples
        i. http://www.northeastern.edu
        ii. https://www.bostonglobe.com
        iii. http://www.wired.com
B. Responsive vs. adaptive
    a. Both change appearance based on the browser environment (usually browser width)
    b. Responsive
        i. Respond to the size of the browser at any given point
        ii. No matter browser width, the site adjusts its layout optimized to the screen
    c. Adaptive
        i. Adapt to the width of the browser at a specific points
        ii. On phone, users might see single column view
        iii. On tablet, users might see two columns
    d. Smooth vs. snap
        i. https://css-tricks.com/the-difference-between-responsive-and-adaptive-design/
C. Mobile first philosophy
    a. Develop for the smallest mobile device first, then progressively enhance the experience as more screen real estate becomes available
        i. If UI/UX designers start a design with the desktop version, they will want to use all the advanced effects and techniques available there, like hover effects, high bandwidth visualizations, etc.
        ii. Using this method, adopting the same design on a mobile is highly challenging unless they give up a lot of their concepts.
        iii. The mobile version will be more like an afterthought, an incomplete product which has been watered down
        iv. But when we start with mobile development, which is the hardest, has the most limitations in terms of screen size, available bandwidth, etc., designers will prioritize only the necessary features -- mobile is not an afterthought!
        v. If you address mobile first, you address the smallest screens and can decide what are the most crucial features to users
            1. Mobile first gives you a chance to reconsider “extraneous” features that take up lots of space/bandwidth/development resources
            2. Do they need to be included on desktop?
            3. Do they need to be included at all?
        vi. After moving to desktop, designers are able to take advantage of advanced features to enhance the product step by step
D. Columns
    a. A display mode for easily defining multiple columns of text.
    b. Why is this useful?

        i. People have trouble reading text if lines are too long, they lose track of which line they were on.
        ii. To make maximum use of a large screen, authors should have limited-width columns of text placed side by side, like newspapers
        iii. This used to be impossible to do with CSS and HTML without creating fixed columns
        iv. Now we can use columns
    c. Column-count
        i. Sets the number of columns to a particular number
        ii. Eg. `column-count: 5;`
    d. Column-width
        i. Sets a minimum column width
        ii. `Eg. column-width: 100px;`
        iii. If you don't specify column-count along with, then the space will be filled up with as many columns of this width as possible
    e. Columns
        i. Shorthand attribute for setting column-count and/or column width
        ii. `Eg. columns: 100px;`
        iii. `Eg. columns: 5;`
        iv. `Eg. columns: 12 8em;`
        v. Setting both column-width and column-count gives you the ability to set the minimum column width if the count specified would create very narrow unreadable columns
    f. Column-gap
        i. Sets the width in between columns (defaults to 1em)
        ii. `Eg. column-gap: 50px;`

E. 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. Flexbox uses two types of boxes
        i. flex containers group a bunch of flex items together and define how they're positioned
        ii. flex item is any direct child of a flex container. Their main purpose is to let their container know how many things it needs to position
    d. Display: flex
        i. Creates the flex container (parent) for children (DIRECT children only)
        ii. Using flexboxes is all about aligning a bunch of flex items inside a container
        iii. Each flex item could be a flex container for their own items
    e. Flex container attributes
        i. Most of the flex attributes are set on the container rather than on the element to be positioned itself
            1. The container is responsible for most of the positioning, not the items
        ii. Flex-direction
            1. row/row-reverse
            2. column/column-reverse
        iii. Main axis vs. cross axis
            1. flex items mostly lay in horizontal rows or vertical columns
            2. The main axis is the primary line items are placed along in the container
                a. along horizontal line for row flow, vertical line for column flow
            3. The cross axis is the perpendicular line to the main axis

            a. Along the vertical line for row flow, horizontal line for column flow

  iv. Flex-wrap
    1. By default, values will all try to fit along main axis, but wrapping can be set
    2. Nowrap (default)
    3. Wrap/wrap-reverse
  v. Flex-flow: shortcut for flex-direction + flex-wrap
    1. Eg `flex-flow: row wrap;`
  vi. Justify-content
    1. Defines alignment along the main axis
    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: items are evenly distributed in the line with equal space around them
        a. visually the spaces aren't equal, since all the items have equal space on both sides
    7. Space-evenly: items are distributed so that the spacing between any two items (and the space to the edges) is equal
  vii. Align-items
    1. Defines alignment at cross axis to main axis (along vertical line for row flow, 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
  viii. 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

f. Flex children attributes
  i. Order
    1. HTML elements are laid out in the source order by default
    2. Flex children 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. These attributes dictate what amount of the available space inside the flex container the item should take up

5. Flex-grow
    a. how much of the remaining space in the flex container should be assigned to that item
    b. Default is 0 -- any extra space should be placed after the end of the last item and not allocated to any of the elements
    c. Flex-grow: 1; added to an element means take all the extra space and add it to that element
    d. Take the total empty space in the container divided by the sum of all the flex-grow factors of all the items. This is the amount that needs to be added to each flex item. Multiply this amount by the flex-grow factor and then add to the natural width of the flex item. This is the final size of the flex item in the container.
6. Flex-shrink
    a. determines how much needs to be removed from the flex item relative to the rest of the flex items in the flex container when there isn't enough space on the row
    b. If the size of all flex items is larger than the flex container, items shrink to fit according to flex-shrink
    c. Default is 1 -- all items shrink equally to fit into the container
    d. Take the overall size of the flex container divided by the sum of all the flex-shrink factors of all the items. This is the amount that needs to be removed from each flex item. Multiply this amount by the flex-shrink factor and then subtract from the natural width of the flex item. This is the final size of the flex item in the container.

iii. Flex-basis
1. The size of flex item before it is placed into a flex container
2. It's the ideal or hypothetical size of the item.
3. Flex-basis is not a guaranteed size! It's dependent on the other items in the container.
4. As soon as the browser places the items into their flex container, things change.
5. If no flex-basis is specified, the default value is auto and the flex-basis falls back to the item's width property.
6. If no width is specified, then the flex-basis falls back to the computed width of the item's contents.

iv. Flex
1. Shorthand for flex-grow (+ flex-shrink + flex-basis) [shrink/basis optional]
2. `Eg flex: 1;`
3. `Eg flex: 0 1 auto; (default)`
4. `Eg 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

F. Grid
a. https://css-tricks.com/snippets/css/complete-guide-grid/
b. two-dimensional grid-based layout system
c. Picture newspaper layout with rows and columns
d. 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

e. Grid container concepts
   i. 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. In a left-to-right language, line 1 is on the left-hand side of the grid, in right-to-left, line 1 is on the right-hand side
   ii. 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, means a fraction of the available space in the grid container
         a. Eg `1fr 1fr 1fr`
         b. each track = 3 equal width tracks that fill available space
   iii. grid area
      1. elements spanning one or more cells by row or by column
      2. must be rectangular – can't create an L-shaped area, for example
   iv. grid cell
      1. single "unit" box of the grid
      2. smallest unit on a grid
   v. Gutters
      1. Spaces between grid cells
      2. Created with column-gap & row-gap properties

f. Setting up a grid
   i. display: grid
      1. Creates the grid container (parent) for children (DIRECT children only)
      2. Grids can be nested by making children grid containers w/ display: grid
   ii. display: inline-grid
      1. Creates the grid container where the size is constrained to the size of the content
   iii. Explicit vs. implicit grid
      1. Rows and columns defined with grid-template-columns and/or grid-template-rows is called the explicit grid
      2. Rows and columns created when content goes beyond the bounds of the explicit grid is called the implicit grid
      3. Eg defined our column tracks with the grid-template-columns property, but the grid also created rows on its own
   iv. grid-template-rows/grid-template-columns
      1. Values are used to create explict grid
      2. Track size values can be any non-negative, length value (px, %, em, etc.)
      3. Grid-template-rows
         a. An explicit row track is created for each value specified for grid-template-rows
      4. Grid-template-columns

a. An explicit column track is created for each value specified for grid-template-columns

5. Any rows/columns created beyond the explict tracks are implicit and their size is defined by the content OR by grid-auto-rows/grid-auto-columns

v. grid-auto-rows/grid-auto-columns
   1. Values are used when the rows/columns are more than what's defined in the explicit grid
   2. Eg, Define the first row of columns and then define what happens for the next infinity rows/columns after that
   3. Track size values can be any non-negative, length value (px, %, em, etc.)
   4. Grid-auto-rows
      a. Values used for every implicit row track created
   5. Grid-auto-columns
      a. Values used for every implicit column track created

vi. Grid-auto-flow
   1. By default, new rows will be created as the implicit grid grows
   2. Flow can be changed to creating new columns by setting `grid-auto-flow: column`

vii. column-gap / row-gap
   1. Specify the spacing between columns and rows, respectively
   2. Gap size values can be any non-negative, length value (px, %, em, etc.)
   3. Can use the shortcut value gap for row-gap + column-gap

viii. repeat()
   1. grids with many tracks can use repeat()
   2. repeat all or a section of the track listing
   3. The repeat() function accepts 2 arguments:
      a. the first represents the number of times the defined tracks should repeat
      b. the second is the track definition
   4. Eg `1fr 1fr 1fr` is the equivalent of `repeat(3, 1fr)`

ix. minmax()
   1. Tracks sizes can be defined to have a minimum and/or maximum size with the minmax() function
   2. It's a way of defining how track rows or track columns can expand/collapse and never go under/over a certain size
   3. The minmax() function accepts 2 arguments
      a. the first is the minimum size of the track
      b. the second the maximum size
   4. In addition to any non-negative, length value (px, %, em, etc.), the values can also be auto, which allows the track to grow/stretch based on the size of the content

g. Positioning via grid lines

i. Line numbers can be used as reference for where grid cells should appear, taking them out of the normal flow of the grid and inserting them where specified, including spanning multiple rows/columns. The displaced elements will continue to flow in the grid as expected.
   1. Grid-row-start: put the top of the element on this line
   2. Grid-row-end: put the bottom of the element on this line
   3. Grid-column-start: put the left of the element on this line

            4. Grid-column-end: put the right of the element on this line
        ii. These values are flipped/reversed if the grid is reversed due to directionality of right-to-left
        iii. Can also be written with shorthand
            1. Grid-row: 2 / 5 is equivalent to grid-row-start: 2, grid-row-end: 5
        iv. Positioned grid areas can be overlapping and affected by z-index (equivalent to absolute positioning)
        v. Grid lines can be named when setting up the grid-template-columns/ grid-template-rows
            1. Eg. `grid-template-columns: [one] 1fr [two] 1fr [three] 1fr [four];`
            2. Grid line would then be referenced via the name, eg. `grid-column: one / four;`

G. 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