These are my notes based on lessons from freecodecamp.com

* CSS variables
  + Can be used to change many CSS style properties at once by changing only one value
  + Create a variable by writing -- before name, then colon, and then value, such as:
    - --penguin-skin: gray;
  + Assign a variable to a CCS property
    - background: var(--penguin-skin);
  + Attach fall back value to the variable (which would be used if the variable doesn't work):
    - background: var(--penguin-skin, black);
  + Variables can be changed when the condition of a media query is met (by changing the width of the browser window)
* :root element can be thought of as a container for your entire HTML document
  + If variables are defined in :root, they will be available throughout the whole web page
  + Variable defined in :root can be overwritten by setting the variable again in another element
* text-align
  + Possible values: left, center, right, justify
* width
  + Relative length units (such as em), absolute length units (such as px), or as a percentage of its containing parent element
* height
  + Relative length units (such as em), absolute length units (such as px), or as a percentage of its containing parent element
* strong tag - to bold things
  + Applies CSS property: font-weight: bold;
* u tag to underline
  + Applies CSS property: text-decoration: underline;
* em tag used to emphasize text
  + Applies CSS property: font-style: italic;
* s tag to strickthrough text
  + Applies CSS property: text-decoration: line-through;
* <hr> tag to add line - self-closing tag
* background-color: rgba(45, 45, 45, 0.1);
* box-shadowproperty applies one or more shadows to an element.
  + Takes values for offset-x(how far to push the shadow horizontally from the element), offset-y(how far to push the shadow vertically from the element), blur-radius, spread-radiusand a color value, in that order. The blur-radiusand spread-radiusvalues are optional
  + Example: box-shadow: 0 10px 20px rgba(0,0,0,0.19), 0 6px 6px rgba(0,0,0,0.23);
* opacity
  + 1 - opaque; 0 - transparent
* text-transform
  + lowercase - "transform me"
  + uppercase - "TRANSFORM ME"
  + capitalize - "Transform Me"
  + initial - Use the default value
  + inherit - Use the text-transform value from the parent element
  + none - Default: Use the original text
* font-size - can be used to set size in px
* line-height - change vertical space of each line
* pseudo-class is a keyword that can be added to selectors, in order to select a specific state of the element
  + a:hover {

   color: red;

}

* CSS box model
  + Box-level items - own line
  + Inline items
* Position: relative - allows you to specify how CSS should move it relative to its current position in the normal flow of the page
  + Used with:
    - Offset direction: left, right, top or bottom
      * Moves the element away from the referenced side (effectively, the opposite direction).
    - Pixels, percentages, or ems
  + Changing the relative position of the element does not change the normal flow - other elements still appear as it was in it's default position
  + Example: moves the p elements 10 px from the bottom
    - p {

  position: relative;

  bottom: 10px;

}

* Position: absolute
  + Locks element relative to the parent positioned container (which is done through position: relative)
    - Locks to closest relative
  + Removes element from normal flow - other elements ignore it
* position: fixed
  + used with the CSS offset properties and also removes the element from the normal flow of the document
* float: left, right
  + Floating element removed from normal flow; pushed to left or right of parent container
* z-index
  + Can be used to change whether the element appears behind or on top of other elements
  + Must be an integer
  + Higher values of z-index move the element on top of other elements with lower values
* margin: auto
  + Using margin to center horizontally on the page
  + Can be applied to div and also to img (if we also change the display to block)
* Colors
  + Complementary colors - opposite each other on the color wheel
    - red (#FF0000) and cyan (#00FFFF)
    - green (#00FF00) and magenta (#FF00FF)
    - blue (#0000FF) and yellow (#FFFF00)
* RGB additive color model
  + Primacy colors: Red (R), Green (G), Blue (B)
  + Secondary colors - created by mixing primary colors
    - Cyan - G and B (00FFFF)
    - Magenta - R and B
    - Yellow - R and G
  + Tertiary colors
    - orange = #FF7D00
* Other colors
  + Raspberry FF007D
  + Teal 09A7A1
* hsl() - alternative way of picking color by stating it's properties
  + Hue - color - between 0 and 360
  + Saturation - amount of gray in a color
    - In percentage - 100% being fully saturated
  + Lightness - amount of white or black in a color
    - 0% black
    - 100% white
    - 50% normal color
  + Examples with fully saturated, normal color:
    - red: hsl(0, 100%, 50%)
    - yellow: hsl(60, 100%, 50%)
    - green: hsl(120, 100%, 50%)
    - cyan: hsl(180, 100%, 50%)
    - blue: hsl(240, 100%, 50%)
    - magenta: hsl(300, 100%, 50%)
    - Can be used to adjust the tone of a color
* linear-gradient() used to create a gradient
  + background: linear-gradient(gradient\_direction, color 1, color 2, color 3, ...); [have to specify at least two colors, have option to add more colors]
  + gradient-direction specified in deg (90deg specifies a vertical gradient; 45deg is angled as a backslash)
  + Example: background: linear-gradient(90deg, red, yellow, rgb(204, 204, 255));
* repeating-linear-gradient() - repeats the gradient
  + Example:

background: repeating-linear-gradient(

90deg,

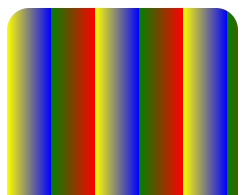
yellow 0px,

blue 40px,

green 40px,

red 80px

);



* + Example 2:
  + background: repeating-linear-gradient(

45deg,

yellow 0px,

yellow 40px,

black 40px,

black 80px

);



* background property supports url() function - can link to an image or gradient
  + Example:

background: url("<https://i.imgur.com/MJAkxbh.png>");

* border-radius: 50% can be used to create circles (if the width and height of the object are the same size)
* left: xx and right: xx can be set to percentages too (not just pixels)
  + Example: left: 65%;
* transform property with scale() function can be used to change the size of an element

p {

  transform:scale(2);

}

* Other functions of the transform property: move, rotate, skew
* Transform property can be used with pseudo-classes, such as :hover to add interactivity
  + Example: to scale the paragraph elements to 2.1 times their original size when a user hovers over them:

p:hover {

              transform: scale(2.1);

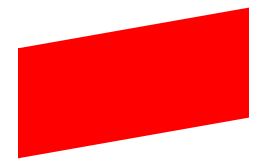
          }

* skewX() function of transform property skews the selected element by the degree provided, along the horizontal axis.
  + Examples:
    - transform: skewX(-24deg);

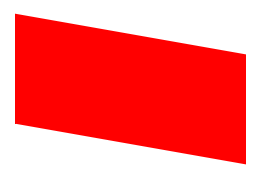


* + - transform: skewX(24deg);

* skewY() function of transform property skews the selected element by the degree provided, along the vertical axis
  + Examples:
    - transform: skewY(-10deg);



* + - transform: skewY(10deg);



* rotate() function of the transform property - works the same was as skewX() and skewY()
* border-radius: 50% creates a circle
* box-shadow property (offset-x, offset-y, blur-radius, spread-radius, color) is used to set the shadow of an element
  + blur-radius and spread-radius values are optional
* background-color can be set to transparent
* ::before and :: after pseudo-elements are used to add something before and after a selected element
  + Must have a content property that is usually used to add text or photo to the element
  + When ::before and ::after are used to make shapes, content property is set to an empty string (content: "");
* margin: auto;
  + Can be used to center an element horizontally and vertically
* @keyframes can be used to change the color of a button in the hover state
  + When used to indicate time in animations, ms stands for milliseconds
  + animation-fill-mode properties specifies the style that should be applied to the element once the animation has finished
    - animation-fill-mode: forwards;
      * Animation retains whatever was the last style
  + Example:
    - <style>

button {

border-radius: 5px;

color: white;

background-color: #0F5897;

padding: 5px 10px 8px 10px;

}

button:hover {

animation-name: background-color;

animation-duration: 500ms;

animation-fill-mode: forwards;

}

@keyframes background-color {

100% {

background-color: #4791d0;

}

}

</style>

<button>Register</button>

* offset properties *right*, *left*, *top*, and *bottom* can be used to create movement if elements have a specified *position* (such as *fixed* or *relative*)
* *opacity* values (0 to 1) can be added to keyframes as part of animation
  + Example

@keyframes fade {

50% {

opacity: 0.1;

}

}

* *animation-iteration-count* property can be used to set how many times an animation is run
  + Can be set to a number (like "3" or to "infinite")
  + It is used with the *animation-name* and *animation-duration* properties
    - Example:

animation-name: bounce;

animation-duration: 1s;

animation-iteration-count: 3;

* *animation-timing-function* property controls how quickly an animated element changes during the animation
  + Options: ease, ease-out, ease-in, linear
* cubic-bezier function
  + x-axis - duration
  + y-axis - change in animation
  + four main points: p0, p1, p2, p3
  + p0 and p3 are set for you - located at (0,0) and (1,1)
  + We set the x and y values of p1 and p2 as: (x1, y1, x2, y2)
  + Example:
    - *animation-timing-function: cubic-bezier(0.25, 0.25, 0.75, 0.75);*
    - x and y values are equivalent for each point (x1 = 0.25 = y1 and x2 = 0.75 = y2), results in a line that extends from the origin to point (1, 1).
* How to resolve the Issue of footer not staying at the bottom of the page
  + body
    - position: relative;
    - height: 100%;
  + footer
    - position: absolute;
    - height: 50px;
    - bottom: 0;
  + Reference: <https://matthewjamestaylor.com/bottom-footer>
* Accessibility Practices
  + *alt* attribute of the *img* tab - used to describe the image referenced by the *img* tab
    - Should be left blank (*alt=""*) when image is already explained with text content or if the image does not add meaning to the page
  + Heading tags (h1 to h6) should convey semantic meaning
    - There should be only one h1 tag
  + HTML 5 introduced *main*, *header*, *footer*, *nav*, *article*, and *section* tags
    - main - main content; only one per page; embedded landmark feature
      * Screen readers can jump straight to the main tag
    - article - used to wrap independent, self-contained content
    - section - grouping related content
    - div - grouping content if other tags cannot be used
    - header - introductory material, material that repeats on multiple pages; embedded landmark feature
      * Works well to contain the nav tag
    - nav - main navigation links; embedded landmark feature
    - footer - footer info, such as copyright; embedded landmark feature
  + audio - semantic tag for adding sound or audio stream content
    - controls attribute - shows play, pause, and other controls on the browser
      * supports keyboard control for playing/pausing
    - Example:

*<audio id="meowClip" controls>  
 <source src="audio/meow.mp3" type="audio/mpeg" />*  
*</audio>*

* + *figure* and *figcaption* tags - give semantic meaning to thing such as charts, figures and their caption
    - *figcaption* tag goes inside the *figure* tag
  + form - *label* element - used to add a label to a form input element
    - *for* attribute must be the same as the *id* attribute of the input element
  + *fieldset* element wraps around a set of radio buttons
    - *legend* tag is used to provide information about the radio buttons
    - *fieldset* element is not needed when the radio button choices are very clear and self-explanatory
  + Setting *type* attribute of *input* tag to "*date*"
    - Allows the user to easily select a date in the input field, from a date picker
  + *time* tag with *datetime* attribute
    - *time* tag wraps around a date or time displayed on the page
    - *datetime* attribute of the tag specifies the date or time in a machine-readable, standardized format
      * Example: <time datetime="2015-03-30">March 30 in 2015</time>
  + CSS can be used to position information meant only for screen-readers off screen
  + The following code can be used to do this:

           .sr-only {

  position: absolute;

  left: -10000px;

  width: 1px;

  height: 1px;

  top: auto;

  overflow: hidden;

    }

* + High contract text improves readability
    - Web Content Accessibility Guidelines (WCAG) recommend at least a 4.5 to 1 contrast ratio for normal text
    - 1:1 ratio indicates the same color
    - 21:1 radio indicates white against black
  + Accessibility challenges with the use of color
    - Color should not be used as the only way to convey important information
    - Foreground and background colors need sufficient color contrast
    - For a color that is defined in the hsl() format, you can change its lightness by increasing the percentage of the third argument
  + Neighboring colors on the color wheel should not be used to convey important information because they would be hard for a color blind person to see
  + *accesskey* attribute can be used to create a shortcut key
    - Can we used on any element but most useful for buttons, form elements, and links.
    - <button accesskey="c">Checkout</button>
  + *tabindex* attribute
    - Can add *tabindex=0* to an element to indicate that the element can be focused upon with a keyboard
    - Adding tabindex attribute to a tag also enables the CSS pseudo-class :focus to work on the tag
    - Value set to 1 or higher can be used to indicate the exact tab order of elements
  + Responsive Design Principles
    - Media queries - change the presentation of content based on different viewport sizes
    - Example of a media query that changes the font size to 15px when the device's width is less than or equal to 200px:

@media (max-width: 200px) {

    p {

font-size: 10px;

}

}

* + Making an image responsive

img {

  max-width: 100%;

  display: block;

  height: auto;

}

* *max-width: 100%;*
  + Image scales to take up the full width of the parent container
* *display: block;*
  + Changes the image from an inline element (its default) to a block element
* *height: auto;*
  + Keeps the original aspect ratio of the image
* Retina image
  + To optimize an image for retina displays, do the following:
    - Make height and width values only half of the original size of the image
    - Example:  img { height: 250px; width: 250px; }
* Making typography responsive
  + Use viewport units, instead of em or px, which are relative to the viewport height and width of a device; examples:
    - *width: 50vw;*
    - *height: 10vh;*
    - *width: 60vmin;* - (vmin refers to viewport's smaller dimension)
    - *width: 100vmax;* - (vmax refers to viewport's bigger dimension)
* CCS Flexbox
  + *display: flex;* can be applied to an element to make it possible to position the element and other elements in a responsive way
    - It makes the element into a flex container
  + *flex-direction* property can be applied to an element which has *display: flex;* applied to align the children elements in rows or columns
    - Possible values:
      * *flex-direction: row;*
      * *flex-direction: column;*
      * *flex-direction: row-reserve;*
      * *flex-direction: column-reserve;*
  + main axis - the direction in which the flex items are arranged
    - For rows, main axis is the horizontal
    - For columns, main axis is the vertical
  + cross axis - opposite of the main axis
    - For rows, cross axis is the vertical
    - For columns, cross axis is the horizontal
  + *justify-content* property is applied to the parent flex element to specify how should the flex items be arranged (along the main axis); possible options:
    - *justify-content: center;*
      * All items are in the center
    - *justify-content: flex-start;*
      * All items are at the starting side
    - *justify-content: flex-end;*
      * All items are at the ending side
    - *justify-content: space-between;*
      * Items are spread out with space between them
    - *justify-content: space-around;*
      * Items are spread and there is space between the first and last elements and the ends
  + *align-items* property can be used to align flex items along the cross axis; possible values
    - *flex-start*
    - *flex-end*
    - *center*
    - *stretch*
    - *baseline*
  + By default, a flex container will fit all flex items in one line (row or column)
    - *flex-wrap* property can be used to wrap items; possible values
      * *flex-wrap: nowrap;*
      * *flex-wrap: wrap;*
      * *flex-wrap: wrap-reverse;*
  + *flex-shrink* property applies to the flex items and can be used to allow a flex item to shrink to fit the width of the flex container
    - Values are defined as numbers; the higher the number, the more it will shrink
  + *flex-grow* property - allows the flex item to grow to fit the size of the flex container
    - Values are defined as numbers; the higher the number, the more it will grow
  + *flex-basis* property specifies the initial size of the item before flex-shrink or flex-grow takes affect
    - Units used: px, em, %
    - auto: sizes items based on the content
* *flex* property can be used as a shortcut to apply *flex-grow*, *flex-shrink*, and *flex-basis* properties all at once
  + Example: *flex: 2 0 50px;*
    - Will set *flex-grow: 2;*, *flex-shrink: 0;*, and *flex-basis: 50px;*
    - Default values: *flex: 0 1 auto;*
* *order* property is used to specify the order in which the flex items appear (that's different from the order specified by the code)
  + Example: *order: 1;* for one item and *order: 2;* for another item
* *align-self* property can be used to change the alignment of a flex item

 CSS Grid

* *display: grid;*
  + Sets the element to a grid container
* *grid-template-columns* property - used to create columns in a grid container
  + Example: *grid-template-columns: 100px 100px;*
    - Will create two columns of 100px each
* *grid-template-rows* property can be used to set the number of rows automatically within a grid container
* Options for defining the size of rows/columns of grid container
  + px
  + em
  + fr - fraction of the available space
  + auto - width or height of the content, automatically
  + % - percentage of the container
* *grid-column-gap* property can be used to add a gap between columns
  + Example: *grid-column-gap: 20px;*
* *grid-row-gap* property can be used to add a gap between rows
  + Example: *grid-row-gap: 15px;*
* *grid-gap* property can be used as a shorthand for the *grid-column-gap* and *grid-row-gap* properties
  + If it has one value, the same gap value will be applied to both rows and columns
  + If it has two values, the first one is used to set the gap between the rows and the second value is used for the columns
* *grid-column* property is used to set the width of a particular column, by referencing the vertical lines of the grid
  + Example: *grid-column: 1 / 2;*
* *grid-row* property is used to set the height of a row by referencing the horizontal lines of the grid
* *justify-self* property of a grid item - aligns the contents of the grid item horizontally
  + stretch (default)
  + start
  + center
  + end
* *align-self* property of a grid item - aligns the contents of the grid item horizontally
* *justify-items* property of a grid container - aligns the contents of all the items horizontally
* *align-items* property of a grid container - aligns the contents of all the items vertically
* Area - grouping of the cells of a grid
  + Period can be used to designate an empty cell
  + Example:

grid-template-areas:

  "header header header"

  "advert content content"

  "footer footer footer";

* *grid-area* property of an item - can be used to place an item in an area
* You can either create the *grid-area* first and then place an item on the *grid-area* or you can create a *grid-area* at the same time as when assigning an item to be placed within the area
* *repeat* function of the *grid-template-column* and *grid-template-rows* properties can be used to specify the number of columns or rows that should be created
  + *Example: grid-template-rows: repeat(50, 110px);*
  + *Repeat* can also be used repeat values
* *minmax* function can be used to specify the minimum and maximum sizes of an item (for cases where the container changes size)
* *auto-fill* option of the repeat function can be used to automatically insert as many rows and columns as possible (of a particular size) to fill the container
* auto-fit option of the repeat function - inserts rows and columns into a container, while stretching the size of the items to fit the container
* A grid can be created within another grid
  + Can be done by setting the *display* and *grid-template-columns* properties of the element