Css Styles

-The basic anatomy of CSS can be written in both inline styles and stylesheets which can be internal or external
-CSS inline styles can be written inside the opening HTML tag using the style attribute.
-Inline styles can be used to style HTML, but it is not the best practice.
-An internal stylesheet is written using the <style> element inside the <head> element of an HTML file
-Internal stylesheets can be used to style HTML but are also not best practice.
-An external stylesheet separates CSS code from HTML, by using the .css file extension.
-External stylesheets are the best approach when it comes to using HTML and CSS.
-External stylesheets are linked to HTML using the <link> element.

Css Selectors

-CSS can select HTML elements by type, class, ID, and attribute.
-All elements can be selected using the universal selector.
-Multiple CSS classes can be applied to one HTML element.
-Classes can be reusable, while IDs can only be used once.
-IDs are more specific than classes, and classes are more specific than type. That means IDs will override any styles from a class, and classes will override any styles from a type selector.
-Nested elements can be selected by separating selectors with a space.
-Multiple unrelated selectors can receive the same styles by separating the selector names with commas.

Visual Cues

-The font-family property defines the typeface of an element.
-font-size controls the size of text displayed.
-font-weight defines how thin or thick text is displayed.

-The text-align property places text in the left, right, or center of its parent container.
-Text can have two different color attributes: color and background-color. color defines the color of the text, while background-color defines the color behind the text.
-CSS can make an element transparent with the opacity property.
-CSS can also set the background of an element to an image with the background-image property.
-The !important flag will override any style, however it should almost never be used, as it is extremely difficult to override.

THE BOX MODEL

-The box model comprises a set of properties used to create space around and between HTML elements.
-The height and width of a content area can be set in pixels or percentages.
-Padding is the space between the content area and the border. It can be set in pixels or percent.
-Margin is the amount of spacing outside of an element's border.
-Horizontal margins add, so the total space between the borders of adjacent elements is equal to the sum of the right margin of one element and the left margin of the adjacent element.
-The overflow property can be set to display, hidden, or scroll, and dictates how HTML will render content that overflows its parent's content area.
-The visibility property can hide or show elements.

View Box Model Dimensions with DevTools
-In windows
control + shift + i
F12
Chrome 3 dot menu ⁝ > More Tools > Developer Tools

DISPLAY AND POSITIONING

-The position property allows you to specify the position of an element.
-When set to relative, an element's position is relative to its default position on the page.
-When set to **absolute**, an element's position is relative to its closest positioned parent element. It can be pinned to any part of the web page, but the element will still move with the rest of the document when the page is scrolled.
-When set to **fixed**, an element's position can be pinned to any part of the web page. The element will remain in view no matter what.
-When set to **sticky**, an element can stick to a defined offset position when the user scrolls its parent container.
-The **z-index** of an element specifies how far back or how far forward an element appears on the page when it overlaps other elements.
-The **display** property allows you to control how an element flows vertically and horizontally in a document.
-**inline** elements take up as little space as possible, and they cannot have manually adjusted width or height.
-**block** elements take up the width of their container and can have manually adjusted heights.
-**inline-block** elements can have set width and height, but they can also appear next to each other and do not take up their entire container width.
-The **float** property can move elements as far left or as far right as possible on a web page.
-You can **clear** an element's left or right side (or both) using the clear property.

COLOR

-**Named colors**—there are more than 140 named colors
-Hexadecimal is a number system that has sixteen digits, 0 to 9 followed by “A” to “F”.
-Hex values always begin with # and specify values of red, blue, and green using hexadecimal numbers such as #23F41A.

-**RGB colors** use the rgb() syntax with one value for red, one value for blue and one value for green.
-RGB values range from 0 to 255 and look like this: rgb(7, 210, 50).
-**HSL** stands for hue (the color itself), saturation (the intensity of the color), and lightness (how light or dark a color is).
-Hue ranges from 0 to 360 and saturation and lightness are both represented as percentages like this: hsl(200, 20%, 50%).
-You can add **opacity** to color in RGB and HSL by adding a fourth value, a, which is represented as a percentage.

TYPOGRAPHY

-Typography is the art of arranging text on a page.
-Text can appear bold or thin with the **font-weight** property.
-Text can appear in italics with the **font-style** property.
-The vertical spacing between lines of text can be modified with the **line-height** property.
-Serif fonts have extra details on the ends of each letter. Sans-Serif fonts do not.
-Fallback fonts are used when a certain font is not installed on a user's computer.
-The **word-spacing** property changes how far apart individual words are.
The **letter-spacing** property changes how far apart individual letters are.
-The text-align property changes the horizontal alignment of text.
-**Google Fonts** provides free fonts that can be used in an HTML file with the <link> tag or the @font-face property.
-**Local fonts** can be added to a document with the @font-face property and the path to the font's source.

Flexbox-

-It is a layout which is used to arrange elements in 1D arraignment i.e row wise or column wise
Terminologies-
main axis – The main axis of a flex container is the primary axis along which flex items are laid out
main-start | main-end – The flex items are placed within the container starting from main-start and going to main-end.
main size – A flex item's width or height, whichever is in the main dimension, is the item's main size. The flex item's main size property is either the 'width' or 'height' property, whichever is in the main dimension.
cross axis – The axis perpendicular to the main axis is called the cross axis.
cross-start | cross-end – Flex lines are filled with items and placed into the container starting on the cross-start side of the flex container and going toward the cross-end side.
cross size – The width or height of a flex item, whichever is in the cross dimension, is the item's cross size.

Parent Properties-

-display-enables a flex context for all its direct children.
-flex-direction-defining the direction flex items are placed in the flex container.
    row (default): left to right in ltr; right to left in rtl
    row-reverse: right to left in ltr; left to right in rtl
    column: same as row but top to bottom
    column-reverse: same as row-reverse but bottom to top
Flex-wrap-allow the items to wrap as needed
    nowrap (default): all flex items will be on one line
    wrap: flex items will wrap onto multiple lines, from top to bottom.
    wrap-reverse: flex items will wrap onto multiple lines from bottom to top.
flex-flow-shorthand for the flex-direction and flex-wrap properties

Justify-content

flex-start (default): items are packed toward the start of the flex-direction.
flex-end: items are packed toward the end of the flex-direction.
start: items are packed toward the start of the writing-mode direction.
end: items are packed toward the end of the writing-mode direction.
left: items are packed toward left edge of the container, unless that doesn't make sense with the flex-direction, then it behaves like start.
right: items are packed toward right edge of the container, unless that doesn't make sense with the flex-direction, then it behaves like end.
center: items are centered along the line
space-between: items are evenly distributed in the line; first item is on the start line, last item on the end line
space-around: items are evenly distributed in the line with equal space around them. Note that visually the spaces aren't equal, since all the items have equal space on both sides. The first item will have one unit of space against the container edge, but two units of space between the next item because that next item has its own spacing that applies.
space-evenly: items are distributed so that the spacing between any two items (and the space to the edges) is equal.

Align-items

stretch (default): stretch to fill the container (still respect min-width/max-width)
flex-start / start / self-start: items are placed at the start of the cross axis. The difference between these is subtle, and is about respecting the flex-direction rules or the writing-mode rules.
flex-end / end / self-end: items are placed at the end of the cross axis. The difference again is subtle and is about respecting flex-direction rules vs. writing-mode rules.
center: items are centered in the cross-axis
baseline: items are aligned such as their baselines align

The gap property explicitly controls the space between flex items.

Child properties
Order- To mention which item should come first the item with low value comes first.
flex-grow-This defines the ability for a flex item to grow if necessary. It accepts a unitless value that serves as a proportion. It dictates what amount of the available space inside the flex container the item should take up
flex-shrink-This defines the ability for a flex item to shrink if necessary.
flex-basis
This defines the default size of an element before the remaining space is distributed
If set to 0, the extra space around content isn't factored in. If set to auto, the extra space is distributed based on its flex-grow value
flex-This is the shorthand for flex-grow, flex-shrink and flex-basis combined.
Align self
This allows the default alignment (or the one specified by align-items) to be overridden for individual flex items.

Grid Layout-
It is a layout which is used to arrange elements in 2D arraignment i.e row wise and column wise
Parent properties
-display-grid-enables a grid context for all its direct children.
-**grid-template-columns-**It is used to specify how many columns you want to divide a grid into.you can mention in px,% and fractions
grid-template-rows-It is used to specify how many columns you want to divide a grid into.you can mention in px,% and fractions
Grid-template-areas-Defines named grid areas within the grid container.
Grid-gap: A shorthand for grid-row-gap and grid-column-gap
justify-items-Aligns grid items along the inline (row) axis.
Align-items: Aligns grid items along the block (column) axis.

Child properties
grid-column-start, grid-column-end:
Specifies the start and end lines for a grid item
Grid column is shorthand notation for this .

grid-row-start, grid-row-end:
Specifies the start and end lines for a grid item.

grid-area:
A shorthand for setting grid-row-start, grid-column-start, grid-row-end, and grid-column-end.

-

-