Display: grid;

display: grid is a CSS property that allows you to create grid layouts in web design. It is a powerful and flexible way to arrange elements on a web page, providing control over both rows and columns. Grid layouts are particularly useful for creating complex and responsive designs.

Container and Items: To use grid layout, you first define a grid container by setting display: grid; on an element. This container then becomes the parent for grid items, which are the elements you want to arrange within the grid.

Grid Lines: Grids consist of horizontal and vertical lines. You can specify the number of rows and columns in the grid using the grid-template-rows and grid-template-columns properties. You can use various units (like pixels, percentages, or fractions) to define the size of rows and columns.

```
.grid-container {
  display: grid;
  grid-template-rows: 1fr 2fr; /* Two rows with 1:2 height ratio */
  grid-template-columns: 1fr 1fr 1fr; /* Three columns with equal width */
}
```

Grid Gap: You can create spacing between grid items using the grid-gap or its individual properties, grid-row-gap and grid-column-gap.

CSS POSITION:

In CSS, the position property is used to control the positioning of elements within a web page. There are several values for the position property, each of which determines how an element is positioned relative to its normal position in the document flow. Here are the main values for the position property:

Static (Default):

position: static;

This is the default value, and it means that the element is positioned in the normal document flow. It is not affected by the top, right, bottom, or left properties.

Relative:

position: relative;

When an element is set to position: relative;, it is positioned relative to its normal position in the document flow. You can use the top, right, bottom, and left properties to offset the element from its original position.

Absolute:

position: absolute;

Elements with position: absolute; are positioned relative to their nearest positioned ancestor (an ancestor with position relative). If there is no positioned ancestor, it will be positioned relative to the initial containing block (usually the viewport).

Fixed:

position: fixed;

Elements with position: fixed; are positioned relative to the viewport, so they stay in the same place even when the user scrolls the page. Commonly used for creating headers, footers, or other elements that should always be visible.

sticky:

position: sticky;

Elements with position: sticky; behave like position: relative; until they reach a specified scroll position, at which point they become "sticky" and are positioned relative to the nearest scrolling ancestor or the viewport.

CSS TRANSFORM

In CSS, the transform property allows you to apply various 2D and 3D transformations to elements, altering their size, position, and orientation.

translate, scale, rotate, and skew.

Translate:

transform: translate(x, y);

The translate transformation moves an element along the X and Y axes. It displaces the element from its original position without changing its size or shape.

Scale:

transform: scale(x, y);

The scale transformation enlarges or shrinks an element along the X and Y axes. A scale of 1 is the original size, values less than 1 make it smaller, and values greater than 1 make it larger.

Rotate:

transform: rotate(angle);

The rotate transformation rotates an element by a specified angle (in degrees) clockwise. Positive values rotate clockwise, while negative values rotate counterclockwise.

Skew:

transform: skew(x-angle, y-angle);

The skew transformation skews an element by a specified angle along the X and Y axes. Positive values create a clockwise skew, and negative values create a counterclockwise skew.

You can combine multiple transformations by applying them sequentially within the transform property:

```
.element {
  transform: translate(50px, 20px) rotate(45deg) scale(1.5);
}
```

Transformations are great for creating interactive and visually appealing effects on web elements, such as animations, transitions, and hover effects.

Keyframes and animations:

In CSS, keyframes and animations are used to create dynamic and visually engaging effects on web elements. Keyframes define the specific points in an animation sequence, while animations apply these keyframes to elements. Let's break down keyframes, animations, and the relevant animation properties:

Keyframes define a series of styles or transformations that an element should go through during an animation. These keyframes are defined using the @keyframes rule. Each keyframe specifies the properties and values at a particular point in time during the animation.

```
@keyframes slide-in {
  0% {
    transform: translateX(-100%);
  }
  50% {
    transform: translateX(0);
    opacity: 1;
  }
  100% {
    transform: translateX(100%);
    opacity: 0;
  }
}
```

In this example, we've defined a keyframe animation called slide-in that starts with the element translated to the left (-100%), gradually moves it to its original position (0%), and then translates it to the right (100%) while fading out.

Common Animation Properties:

animation-name: Specifies the name of the keyframes you want to use.

animation-duration: Sets the duration of the animation in seconds (s) or milliseconds (ms).

animation-timing-function: Defines the timing function to control the acceleration and deceleration of the animation. Common values include ease, ease-in, ease-out, ease-in-out, linear, and cubic-bezier() for custom curves.

animation-delay: Specifies the delay before the animation starts.

animation-iteration-count: Determines how many times the animation should repeat (e.g., infinite, 3, 2.5).

animation-direction: Sets the direction of the animation (normal, reverse, alternate, alternate-reverse).

Media Queries:

Media queries in CSS are a powerful technique for making your web designs responsive to different screen sizes and devices. They allow you to apply specific CSS rules based on the characteristics of the user's device, such as its screen width, height, orientation, and more. Media queries are essential for creating designs that adapt and look good on various devices, from desktop computers to smartphones and tablets.

Media queries are introduced using the @media rule, followed by a set of conditions enclosed in parentheses. If the conditions in the parentheses are true, the CSS rules within the media query block will be applied. Here's the basic structure:

```
@media (condition) {
  /* CSS rules for the specified condition */
}
```

Example 1: Adjusting Font Size for Smaller Screens:

```
/* Default styles for all screen sizes */
p {
  font-size: 16px;
}
```

```
/* Media query for screens with a maximum width of 768px */
@media (max-width: 768px) {
  p {
    font-size: 14px;
  }
}
```

In this example, paragraphs will have a font size of 16px by default. However, for screens with a maximum width of 768px (typically smaller screens like smartphones and tablets), the font size is reduced to 14px.

Example 2 :Changing Layout for Different Screen Widths:

```
/* Default styles for wider screens */
.container {
  width: 960px;
  margin: 0 auto;
}

/* Media query for screens with a maximum width of 768px */
@media (max-width: 768px) {
  .container {
    width: 100%;
    padding: 10px;
  }
}
```

In this example, the .container element has a fixed width and is horizontally centered for wider screens. For screens with a maximum width of 768px or less, the container's width is set to 100%, and padding is added for better spacing.

Media queries are incredibly versatile, allowing you to tailor your website's design and layout to fit the characteristics of different devices and screens. They are a fundamental tool for creating responsive and user-friendly web designs.