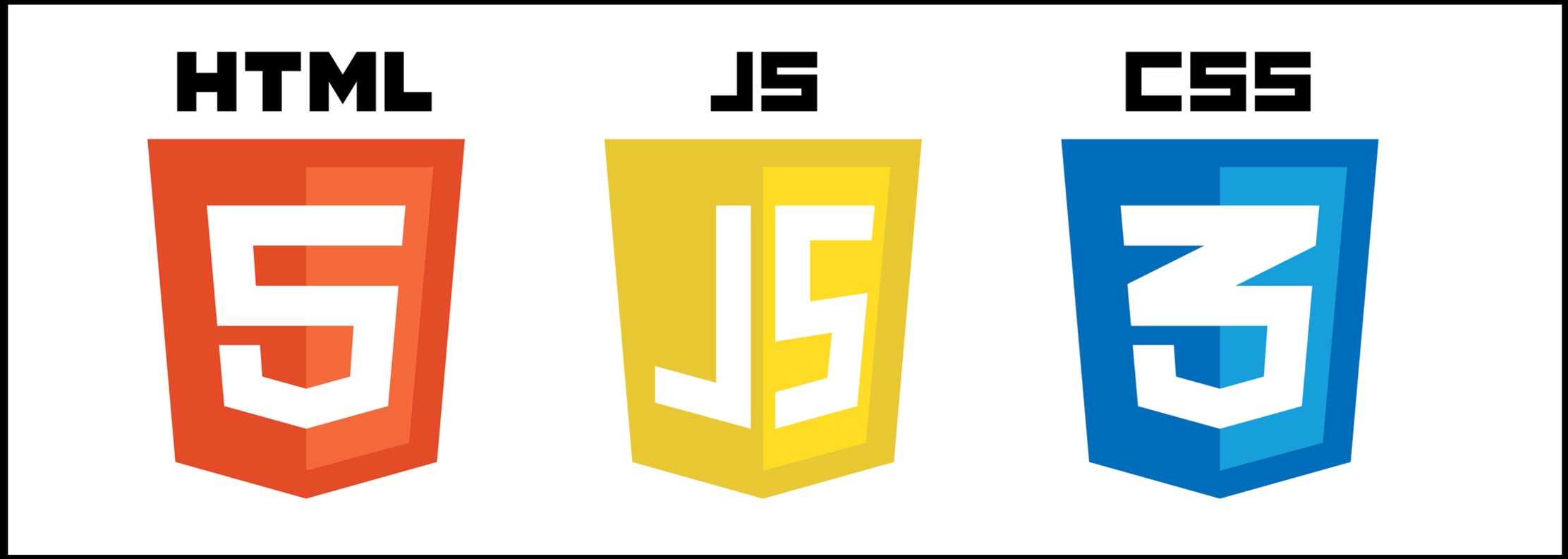
# WELCOME TO

# CSS MASTERY COURSE : THE ULTIMATE GUIDE !



# HTML

# CSS

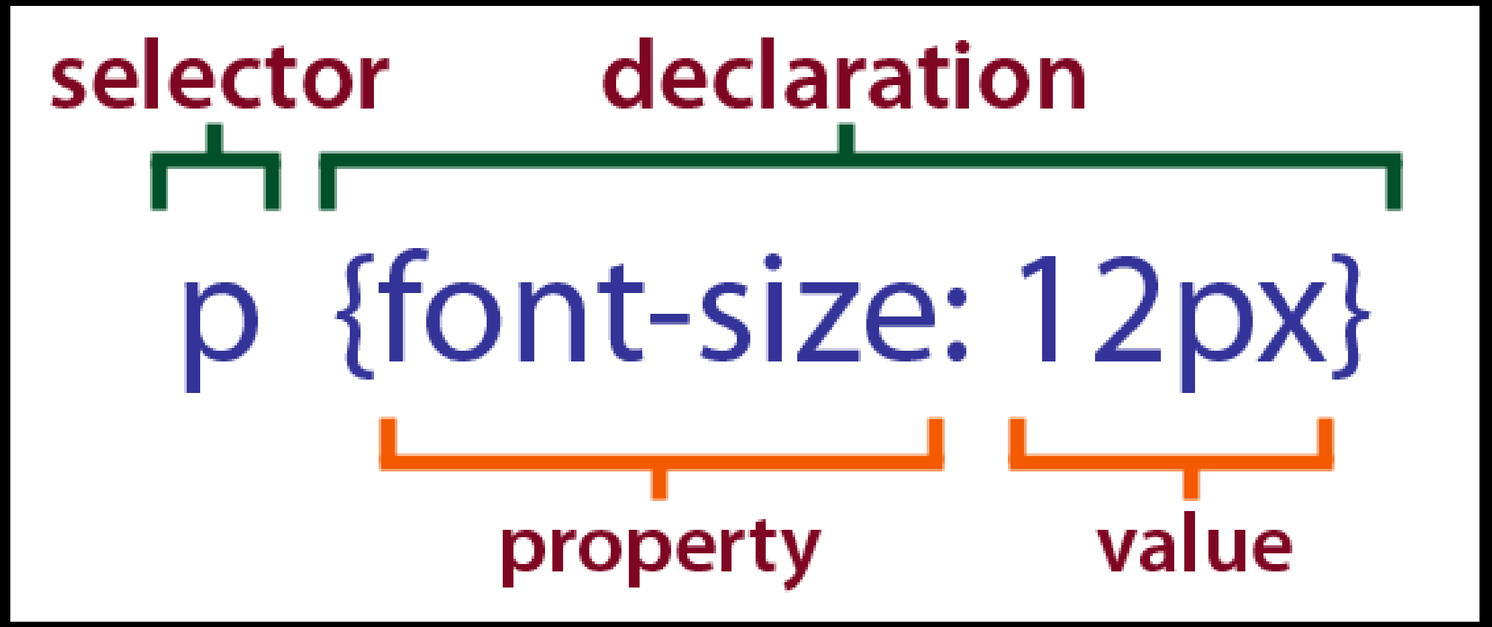
# JAVASCRIPT

# LEVEL 1

# What is CSS?

CSS, or Cascading Style Sheets, is a style sheet language used for describing the look and formatting of a document written in HTML or XML. CSS describes how elements should be rendered on screen, on paper, in speech, or on other media. It allows web developers to control the layout, appearance, and style of multiple web pages all at once.

# BASIC SYNTAX OF CSS



# INCLUDING STYLES

1. INLINE CSS
2. INTERNAL CSS
3. EXTERNAL CSS

# INLINE CSS

Inline CSS refers to the practice of including CSS (Cascading Style Sheets) directly within the HTML document, rather than in a separate external stylesheet. This is achieved by using the "style" attribute within HTML tags.

```
<p style="color: red;">Red</p>
```

# INTERNAL CSS

Internal CSS, or internal style sheets, refers to the practice of including CSS (Cascading Style Sheets) directly within an HTML document rather than in a separate external file.

When CSS is included internally, it is placed within the <style> element in the document's <head> section.

```
<style>
        h1{
            color: red;
          }
</style>
```

# EXTERNAL CSS

External CSS refers to Cascading Style Sheets (CSS) that are stored in separate files and linked to HTML documents. CSS is a stylesheet language used to describe the presentation of a document written in HTML or XML. By externalizing CSS, you can keep the style information separate from the HTML content, making your code more modular and easier to maintain.

## Here are the basic steps to use external CSS:

1. **Create a CSS File:**
2. **Write CSS Rules:**
3. **Link CSS to HTML:**

# COMMENTS IN CSS

In CSS (Cascading Style Sheets), comments are used to add explanatory notes or information within the code that will not be rendered on the webpage.

Here's how you can add comments in CSS:

## EXAMPLE

```
/* This is a comment in css */
```

# SELECTORS IN CSS

1. UNIVERSAL SELECTOR
2. ELEMENT SELECTOR
3. ID & CLASS SELECTOR
4. GROUP SELECTOR
5. DESCENDANT SELECTOR

# UNIVERSAL SELECTOR

In CSS (Cascading Style Sheets), the universal selector is denoted by an asterisk (*). It is a special selector that matches any element. When used in a CSS rule, the universal selector applies styles to every element on a webpage.

```
* {
margin: 0;
padding: 0;
box-sizing: border-box;
}
```

# ELEMENT SELECTOR

In CSS (Cascading Style Sheets), an element selector is a type of selector that targets HTML elements based on their tag names. It is one of the most straightforward and commonly used selectors in CSS. To apply styles to a specific type of HTML element, you use the element selector followed by the element's tag name.

```
p {
/* Your styles go here */
}
```

# ID SELECTOR

Certainly! In CSS (Cascading Style Sheets), an ID selector is a way to select and style a specific HTML element based on its unique identifier. The ID selector is denoted by the hash (#) followed by the ID value.

```
#myElement {
color: blue;
font-size: 16px;
}
```

# CLASS SELECTOR

In CSS (Cascading Style Sheets), the class selector is used to select elements based on their class attribute. The class attribute allows you to apply styles to multiple HTML elements using a single class name.

To use the class selector in CSS, you prefix the class name with a period (.) followed by the class name itself.

```
.myClass {
color: blue;
font-size: 16px;
}
```

# GROUP SELECTOR

In CSS, the GROUP selector is not a standard term or selector. However, you might be referring to the combination of selectors to target specific groups of elements in your HTML document.

```
h1, h2, p {
color: blue;
}
```

# DESCENDANT SELECTOR

In CSS (Cascading Style Sheets), descendant selectors are used to select all instances of a particular element that are descendants of a specified ancestor. The descendant selector is represented by a space between two selectors.

```
#ancestor p {
/* styles go here */
}
```

# LEVEL 2

# COLOR PROPERTY

Colors are specified using predefined color names, or RGB, HEX, HSL, RGBA, HSLA values.

```
p{
color: red;
}
```

# BACKGROUND COLOR

Certainly! The CSS property for setting the background color of an element is background-color. You can use it in your style sheets to define the background color for HTML elements.

```
/* Set background color to red */
.myElement {
  background-color: red;
}
```

# RGB COLOR MODEL

Certainly! In CSS (Cascading Style Sheets), RGB (Red, Green, Blue) is a color model that represents colors by combining various intensities of these three primary colors. To use RGB colors in CSS, you can employ the rgb() function.

In this example, rgb(255, 0, 0) means full intensity of red, no green, and no blue, resulting in a red color. You can adjust the values between 0 and 255 for each color to create a wide range of colors.

```
div {
background-color: rgb(255, 0, 0);
/* This sets the background color to red */
}
```

Additionally, you can use the rgba() function to include an alpha channel for transparency:

Here, the fourth parameter (0.5) represents the alpha channel, where 0 is fully transparent and 1 is fully opaque.

```
div {
background-color: rgba(255, 0, 0, 0.5);
/* This sets a semi-transparent red background */
}
```

# HEX COLOR MODEL

In CSS (Cascading Style Sheets), a hex color code is a way to represent colors using a hexadecimal notation. A hex color code consists of six alphanumeric characters, preceded by a hash (#) symbol. Each pair of characters represents the intensity of the red, green, and blue components of the color.

For example:

#RRGGBB

Here:

- RR represents the hexadecimal value for the red component,
- GG represents the hexadecimal value for the green component, and
- BB represents the hexadecimal value for the blue component.

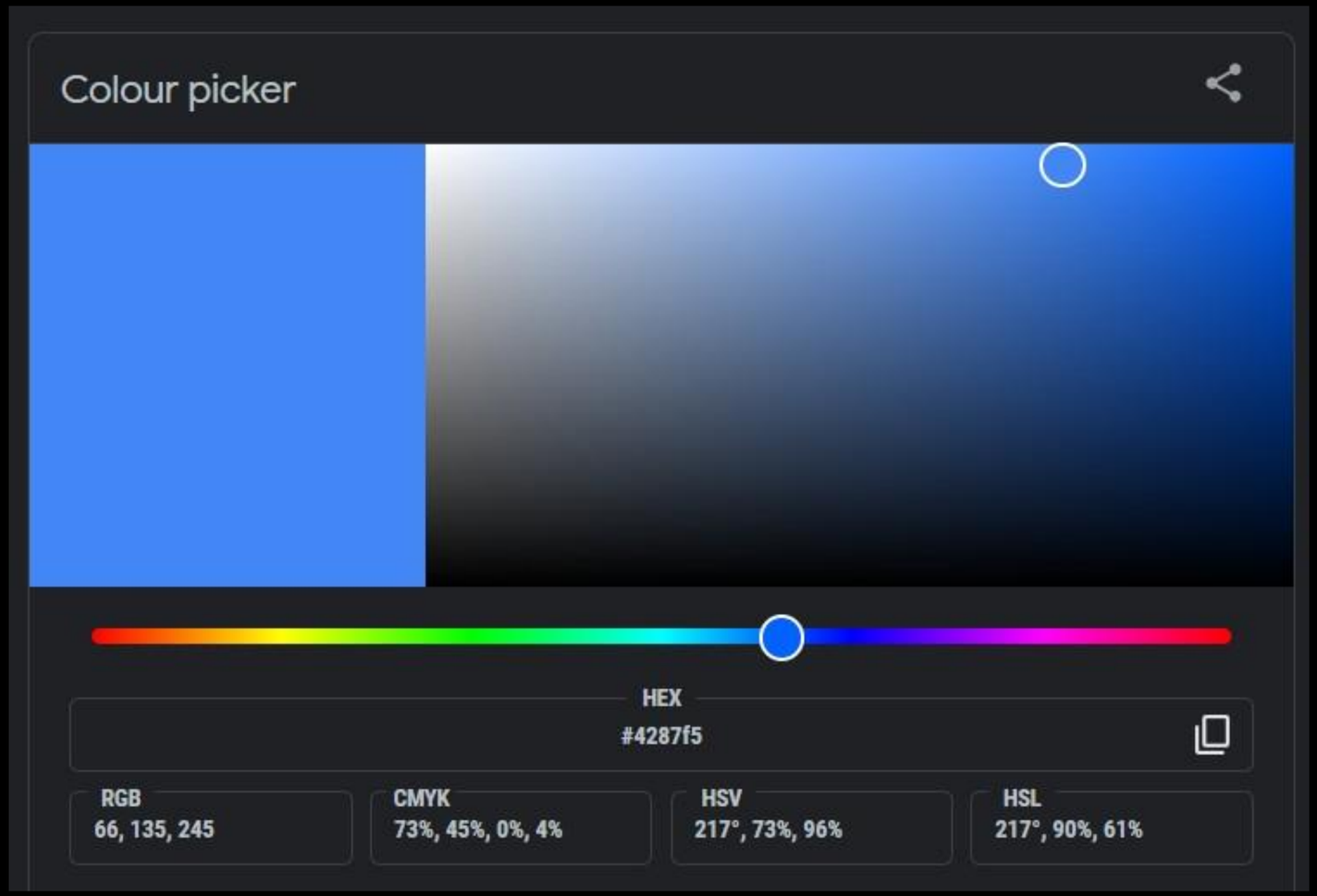
Each component can have a value ranging from 00 to FF, where 00 is the lowest intensity (0 in decimal) and FF is the highest intensity (255 in decimal).

For instance:

- #FF0000 is red,
- #00FF00 is green, and
- #0000FF is blue.

You can use hex color codes to define the color of various elements in your HTML document through CSS, such as backgrounds, text, borders, and more.

# COLOR PICKER




A color picker is a tool or software application that allows users to select and capture colors from an image, website, or any digital interface. It helps in identifying and matching specific colors, often using various color models such as RGB (Red, Green, Blue), HEX (Hexadecimal), or HSL (Hue, Saturation, Lightness).

# LEVEL 3

# WIDTH & HEIGHT PROPERTY

Certainly! In CSS (Cascading Style Sheets), you can set the height and width of elements using the height and width properties.

# Width:

- The width property is used to set the width of an element.
- It can be set in various units like pixels (px), percentages (%), em, rem, etc.

```
.example {
width: 300px;
/* Set width to 300 pixels */
}
```

# Height:

- The height property is used to set the height of an element.
- Similar to width, it can be set in various units.

```
.example {
height: 200px;
/* Set height to 200 pixels */
}
```

## Combined Example:

- You can also set both height and width in the same rule.

```
.example {
width: 300px;
height: 200px;
}
```

# BACKGROUND IMAGE

Certainly! In CSS, the background-image property is used to set an image as the background of an element. It allows you to specify a URL that points to the image you want to use. Here's a basic example:

```
element {
background-image: url('your-image-url.jpg');
}
```

Additionally, you can use other values and properties to control aspects like the background size, position, and repeat. Here's an example with more options:

```
element {
background-image: url('your-image-url.jpg');

background-size:   cover;
/* Options: auto, contain, cover */

background-position: center;
/* Options: top, right, bottom, left, center */

background-repeat:  no-repeat;
/* Options: repeat, repeat-x, repeat-y, no-repeat */
}
```

# VISIBILITY PROPERTY

Certainly! The visibility property in CSS is used to control the visibility of an element on a web page. It has three possible values:

1. visible: This is the default value. The element is visible.
2. hidden: The element is hidden, but still takes up space on the page. It is as if the element is not there, but the space it would occupy is preserved.
3. collapse: This value is typically used with table elements. It removes a row or column, but unlike hidden, it also removes the space that the row or column would have taken up.

```
/* CSS code */
.visible-element {
visibility: visible;
}

.hidden-element {
visibility: hidden;
}

.collapsed-table {
visibility: collapse;
}
```

Keep in mind that setting an element to visibility: hidden; will hide it from view, but it will still be present in the HTML structure and affect the layout. If you want to completely remove an element, you might consider using display: none; instead.

# LEVEL 4

# TEXT-ALIGN PROPERTY

Certainly! In CSS (Cascading Style Sheets), the text-align property is used to control the horizontal alignment of text within an element. Here are the common values for the text-align property:

1. left: Aligns the text to the left.
2. right: Aligns the text to the right.
3. center: Centers the text.
4. justify: Justifies the text, causing it to stretch across the width of the container.

```
.text-left { text-align: left; }
```

# TEXT-DECORATION PROPERTY

Certainly! In CSS (Cascading Style Sheets), text decoration is a property that allows you to control the decoration of text within an HTML element. The text-decoration property has several values that you can use to style the text. Here are some common ones:

```
text-decoration:  none;
/*  underline/overline/line-through  */
```

# TEXT-TRANSFORM PROPERTY

Certainly! In CSS (Cascading Style Sheets), there are various text transformation properties that allow you to change the appearance of text. Here are some common text transformation properties:

```
p {
text-transform: uppercase;
}
/*  lowercase/capitalize/none  */
```

# LINE HEIGHT PROPERTY

Certainly! In CSS (Cascading Style Sheets), the "line-height" property is used to set the amount of space above and below inline elements. It can be applied to elements like paragraphs, headings, and other block-level elements.

```
/* Set line height using a specific value */
p {
line-height:  1.5;
/* This will set the line height to 1.5 times the font size */
}
```

```
/* Set line height using a percentage */
h1 {
line-height:  150%;
/* This will set the line height to 150% of the font size */
}

/* Set line height using a specific length */
div {
line-height:  20px;
/* This will set the line height to a fixed value of 20 pixels */
}
```

# FONT SIZE PROPERTY

Certainly! In CSS (Cascading Style Sheets), you can control the font size of text on your web page using the font-size property. Here's a basic example:

```
/* Set font size in pixels */
body {
font-size: 16px;
}

/* Set font size in em units */
h1 {
font-size: 2em;
}
```

# FONT WEIGHT PROPERTY

Certainly! In CSS (Cascading Style Sheets), the font-weight property is used to specify the thickness or boldness of a font. The values for this property can range from numeric values (e.g., 100, 200, 300, ..., 900) to keywords (e.g., normal, bold, bolder, lighter)

```
font-weight: 400; /* normal */
font-weight: 700; /* bold */
```

```
font-weight: normal;
font-weight: bold;
font-weight: bolder;
font-weight: lighter;
```

# FONT STYLE PROPERTY

Certainly! CSS (Cascading Style Sheets) is used to style the appearance of web pages, including the font style. Here's a basic example of how you can set the font style in CSS:

```
/* Set font style for a specific element */
h1 {
font-style: normal;
/* italic/oblique */
}
```

# FONT FAMILY PROPERTY

In CSS (Cascading Style Sheets), the font-family property is used to specify the font of text in an HTML document. It allows you to set a prioritized list of font family names or generic family names for the browser to use.

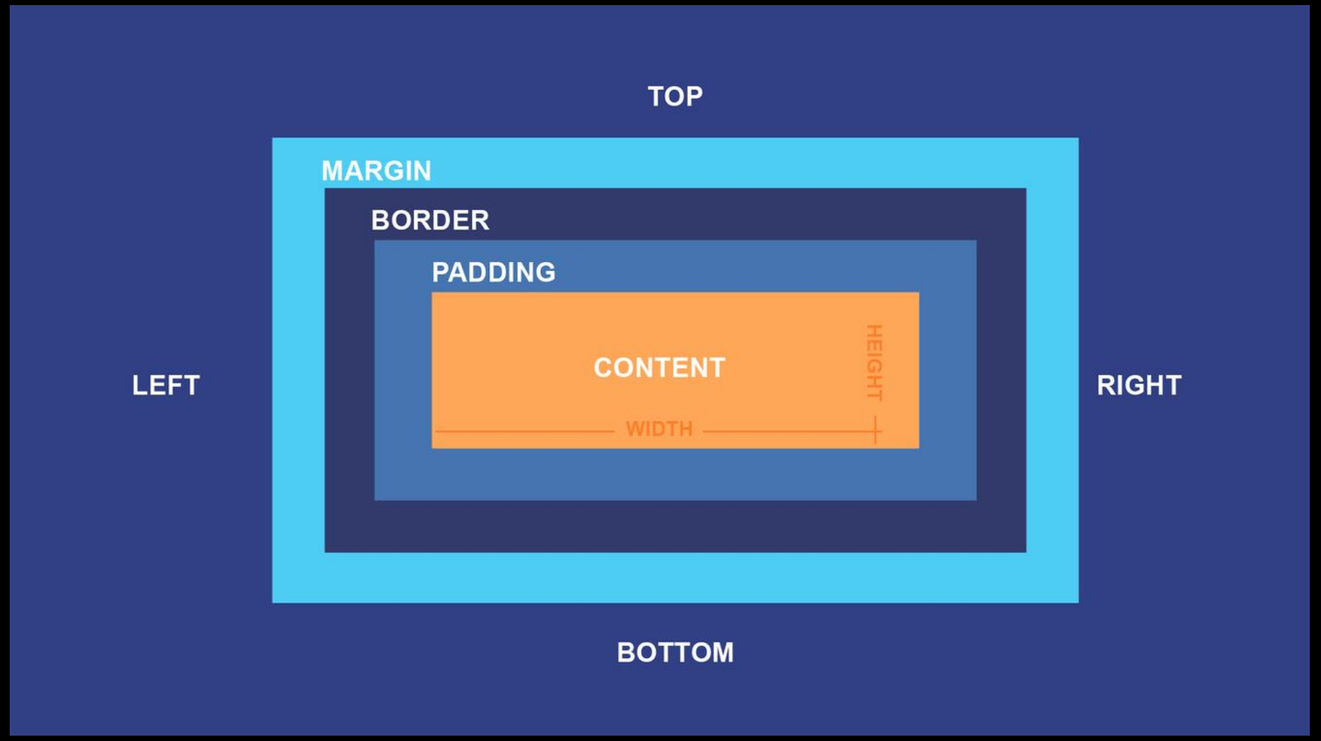
NOTE : YOU CAN ALSO USE GOOGLE FONTS FOR SPECIFIC FONTS FOR YOUR WEBSITE

```
body {
font-family: Arial, sans-serif;
}
```

# LEVEL 5

# BOX MODEL

The box model in CSS (Cascading Style Sheets) is a fundamental concept that describes how elements are rendered and how their dimensions are calculated. It consists of four main components: content, padding, border, and margin.



## Content:

- This is the actual content of the element, such as text, images, or other media.

## Padding:

- Padding is the space between the content and the element's border. It adds space inside the element.

## Border:

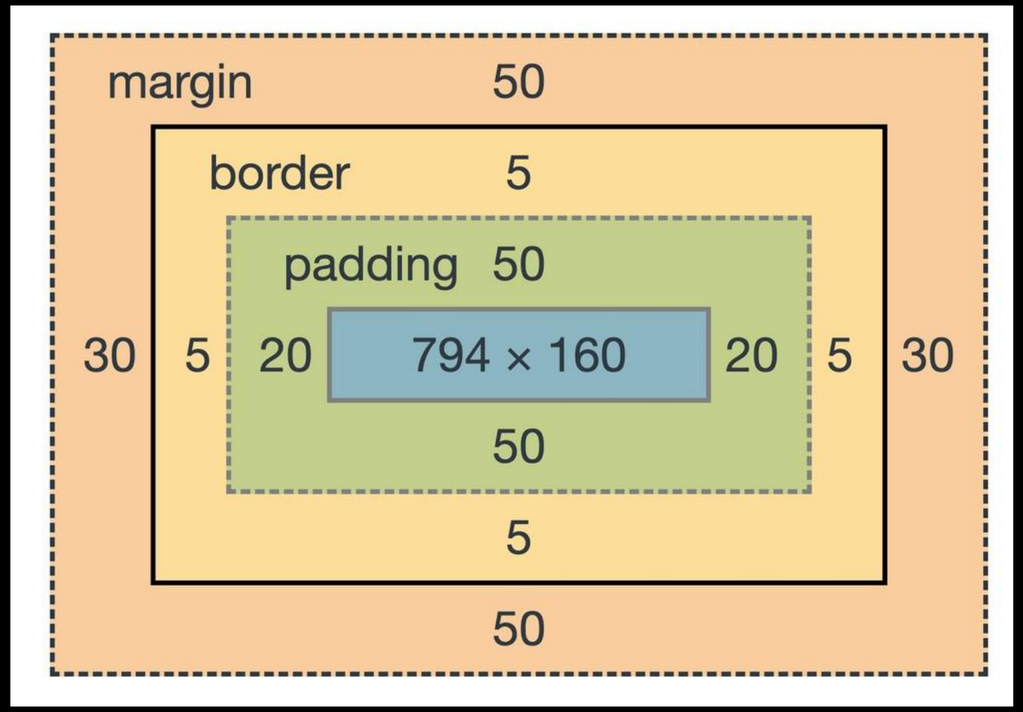
- The border surrounds the padding and content of the element. It can have a specified width, style, and color.

## Margin:

- The margin is the space outside the element's border. It creates space between the element and its surrounding elements.

# PADDING IN BOX MODEL

Certainly! In the context of the CSS box model, "padding" refers to the space between the content of an element and its border. It is used to create space around the content within the element. Padding can be applied to all four sides of an element (top, right, bottom, and left) or individually.



You can set the padding using various units such as pixels, ems, or percentages. For example:

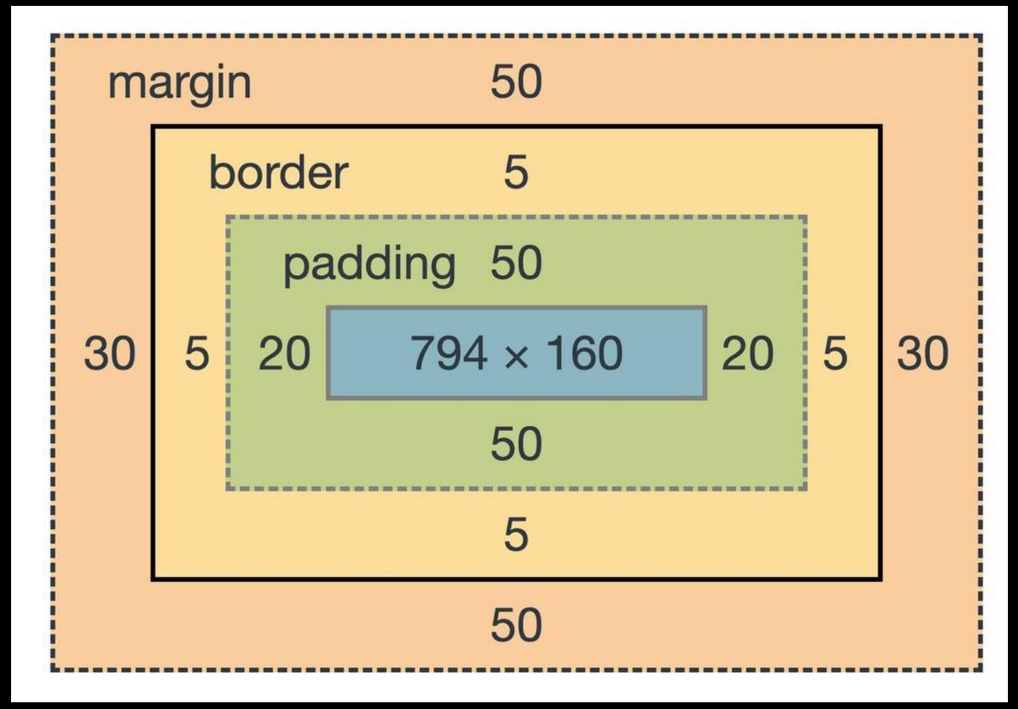
```
.element {
padding-top: 10px;
padding-right: 20px;
padding-bottom: 10px;
padding-left: 20px;
}
```

```
.element {
padding: 10px 20px 10px 20px;
/* top right bottom left */
}
```

```
.element {
padding: 10px;
/* For all sides equal */
}
```

# MARGIN IN BOX MODEL

Certainly! In the CSS (Cascading Style Sheets) box model, the "margin" refers to the space around an element. It is the outermost layer of the box model and adds space outside the border of an element.

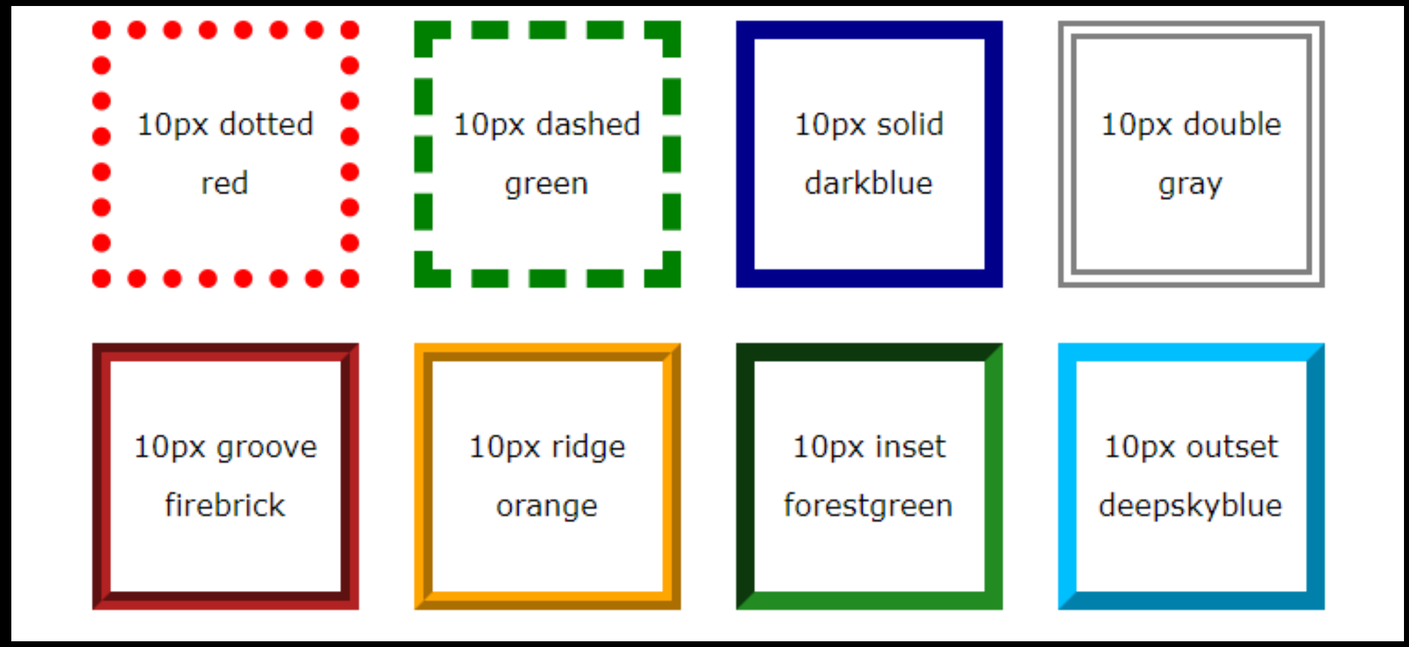


```
/* Shorthand notation for setting all margins */
selector {
margin: 10px;
/* Applies the same margin to all sides */
}

/* Individual margins */
selector {
margin-top: 10px;
margin-right: 20px;
margin-bottom: 15px;
margin-left: 5px;
}
```

# BORDER IN BOX MODEL

In the CSS (Cascading Style Sheets) box model, the term "border" refers to the area around the padding of an element. The box model consists of four main components: content, padding, border, and margin. The border property allows you to define the style, color, and width of an element's border.

```
div {
border: 2px solid #000000;
}
```



# BORDER-RADIUS PROPERTY

**Certainly! In CSS (Cascading Style Sheets), the border-radius property is used to create rounded corners for an element. It is a part of the box model, defining the curvature of the corners of the border area.**

```
.element {
border-radius: 10px;
/* Applies to all corners */
}
```

# BOX-SIZING PROPERTY

In CSS, the box-sizing property is used to control how the total width and height of an element is calculated. There are two main values for the box-sizing property: content-box and border-box.

content-box (default): This is the default value, and it calculates the width and height of the content area only, excluding padding and border. Any padding or border added to the element will increase its total size.

```
box-sizing: content-box;
```

border-box: With this value, the width and height of the element include the content, padding, and border. This means that if you set a specific width or height, the element's total size won't change when you add padding or border.

```
box-sizing: border-box;
```

```
/* Using content-box */
.element {
box-sizing: content-box;
width: 200px;
padding: 20px;
border: 2px solid #000;
}
```

```
/* Using border-box */
.element {
box-sizing: border-box;
width: 200px;
padding: 20px;
border: 2px solid #000;
}
```

In the first example, with box-sizing: content-box, the total width of the element will be 200px + 40px (20px padding on each side) + 4px (2px border on each side), resulting in a wider element.

In the second example, with box-sizing: border-box, the total width of the element will remain 200px, as the padding and border are included within that width.

# LEVEL 6

# DISPLAY (BLOCK) PROPERTY

In CSS (Cascading Style Sheets), the display property is used to define the rendering behavior of an HTML element. When you use display: block; on an element, it means that the element will be rendered as a block-level element.

Block-level elements typically start on a new line and take up the full width available. They also stack on top of each other vertically. Some common examples of block-level elements include , <p>, <h1> to <h6>, <ul>, and <li>.

```
.myBlockElement {
  display: block;
}
```

# DISPLAY (INLINE) PROPERTY

In CSS (Cascading Style Sheets), the display property is used to define how an HTML element should be displayed on a web page. The inline value for the display property is used to make an element behave like an inline element.

When you set display: inline; for an element, it will:

1. Not start on a new line.
2. Only take up as much width as necessary.
3. Allow other elements to be on the same line.

```
div {
display: inline;
}
```

# DISPLAY (INLINE-BLOCK) PROPERTY

In CSS, the display property is used to define how an HTML element should be displayed. The inline-block value is a combination of both inline and block display styles.

When an element is set to display: inline-block, it:

1. Allows the element to have inline behavior, meaning it flows within the content as part of a line.
2. Allows the element to have block behavior, meaning you can set a width and height, and it will start on a new line and stack with subsequent inline-block elements.

```
.inline-block-element {
display: inline-block;
width: 100px;
height: 50px;
border: 1px solid black;
}
```

# DISPLAY (NONE) PROPERTY

In CSS, the display property is used to define the rendering behavior of an element. If you want to hide an element, setting the display property to none is a common approach. Here's an example of how you can use it in CSS:

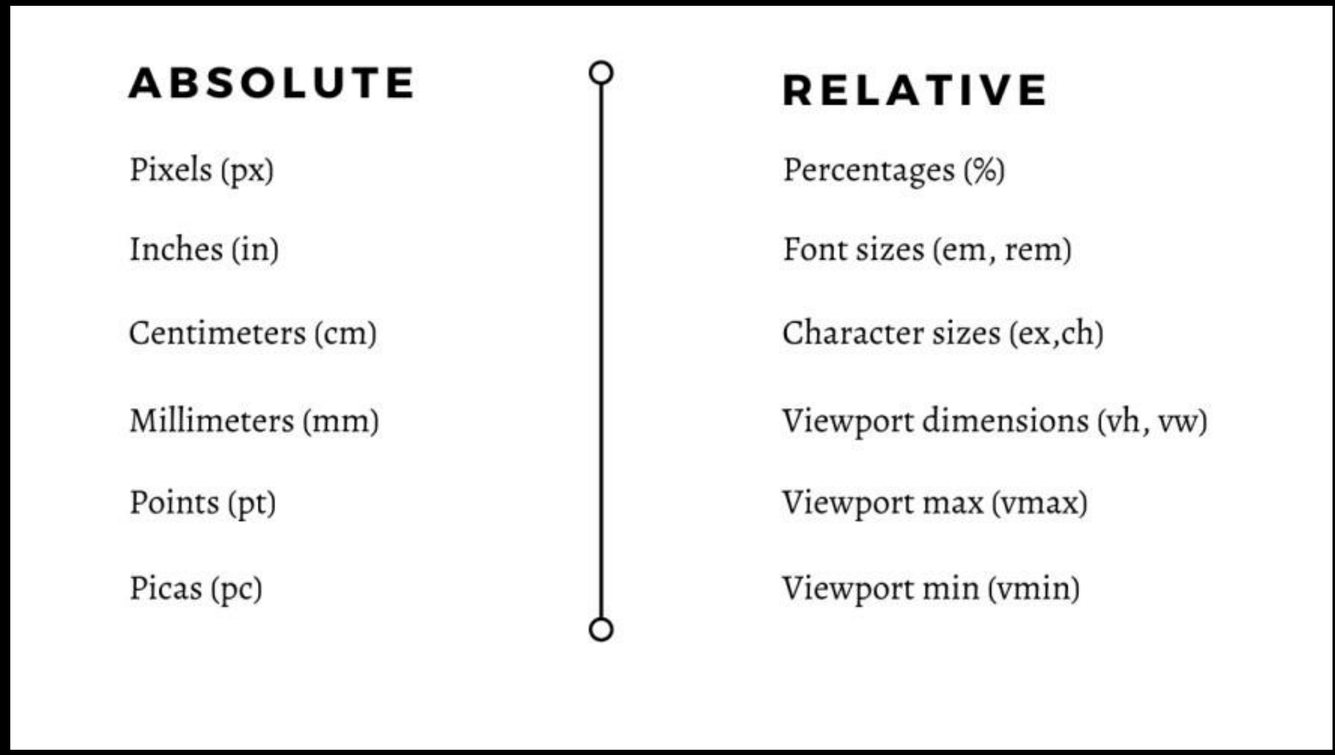
Keep in mind that setting display: none removes the element from the normal flow of the document, so it won't take up any space on the page. If you want to toggle visibility dynamically using JavaScript, you can change the display property accordingly using JavaScript code.

```
.hide-element {
display: none;
}
```

# LEVEL 7

# ABSOLUTE UNITS

**Pixels (written as px in CSS) are used most to define the absolute size of an element. If you define an attribute in pixels, this will not change no matter a user's screen size.**



# RELATIVE UNITS

## Css units

### %

Relative to the value of parent element. 100% is the width of the parent element

### em

Relative to the font-size of the parent element.

### vh

equal to 1% of the height of the browser window size.

### px

Pretty self explanatory .Absolute length in pixel

### rem

Relative to font-size of the root element.

### vw

equal to 1% of the width of the browser window size.

# RELATIVE (PERCENTAGE)

The percentage unit is relative to the parent element. For example, if you set a width of 50% to a child element, it will take up half of the width of its parent element.

```
div {
width: 50%;
}
```

# RELATIVE (EM)

The em unit is relative to the font-size of the element. For example, if the font-size of a parent element is 16 pixels, setting the font-size of a child element to 2em would make it 32 pixels.

```
p {
font-size: 1.2em;
}
```

# RELATIVE (REM)

The rem unit is similar to em but is relative to the root element's font-size. This can be useful for creating a more predictable and consistent layout.

```
body {
font-size: 16px;
}

p {
font-size: 1.5rem;
}
```

# RELATIVE (VW/VH)

**vw (Viewport Width):**

- **The vw unit is relative to 1% of the viewport's width. For example, setting a width of 50vw to an element would make it take up 50% of the viewport's width.**

**vh (Viewport Height):**

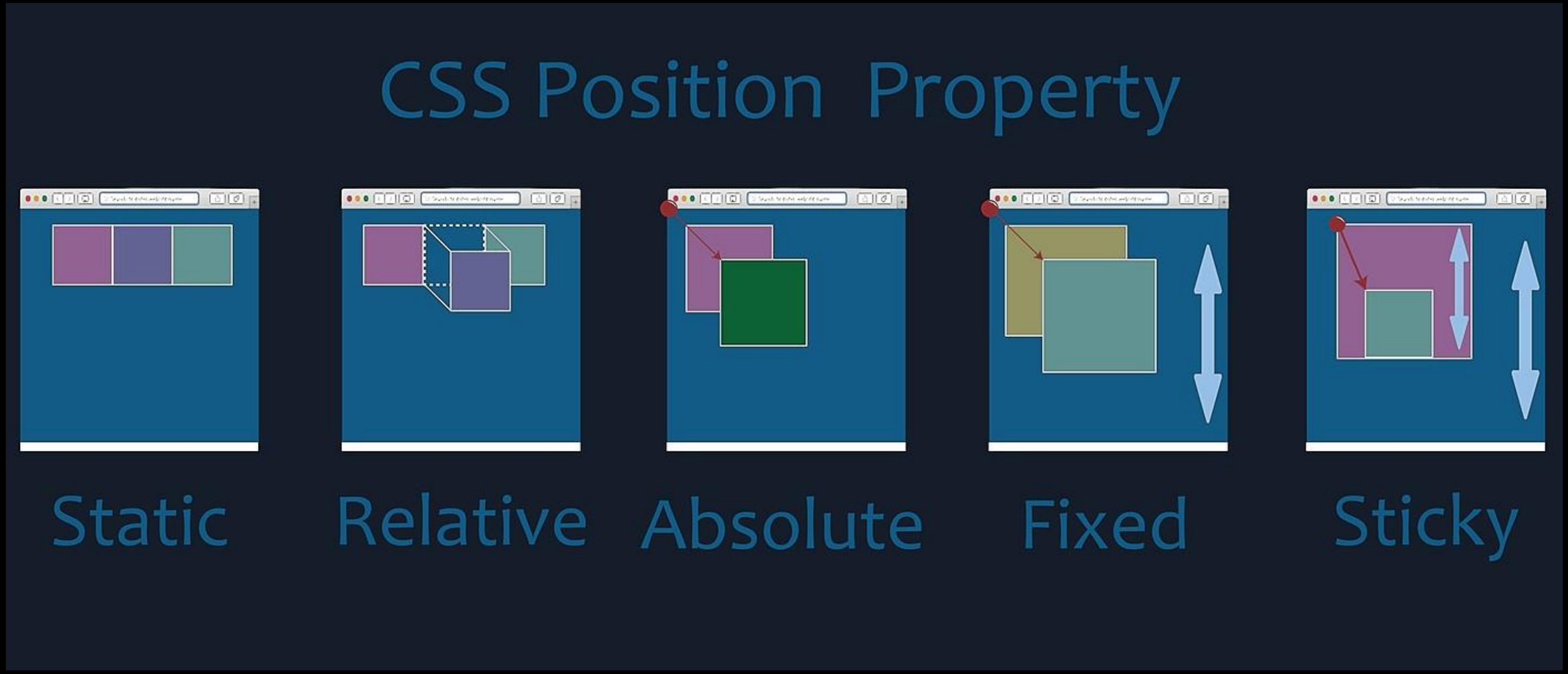
- **Similar to vw, the vh unit is relative to 1% of the viewport's height.**

```
div {
width: 50vw;
}
```

```
div {
height: 30vh;
}
```

# LEVEL 8

# POSITION PROPERTY



# POSITION (STATIC)

the position property is used to control the positioning of an element. However, the term "static" in this context usually refers to the default positioning, where elements are positioned according to the normal flow of the document.

```
.element {
position: static;
}
```

# POSITION (ABSOLUTE)

In CSS, the position property is used to specify the positioning method of an element. When you set the position property to absolute, the element is positioned relative to its nearest positioned ancestor, if any; otherwise, it's positioned relative to the initial containing block (usually the viewport).

## Using Top, Right, Bottom, Left:

- You can use the top, right, bottom, and left properties to explicitly set the distance of the element from the edges of its containing block.

```
.positioned-element {
position: absolute;
top: 20px;
 left: 50px;
}
```

This example positions an element with the class positioned-element 20 pixels from the top and 50 pixels from the left of its containing block.

Remember that the positioned ancestor is the nearest ancestor with a position value other than static. If there is none, the initial containing block (viewport) is used.

# POSITION (RELATIVE)

The position property can take several values, and one of them is relative. When you set an element's position to relative, it is positioned relative to its normal position in the document flow.

Here are the key points about the position: relative; property:

1. Normal Flow: The element is positioned according to the normal flow of the document, and then offset relative to its normal position.
2. Offsets: You can use the top, right, bottom, and left properties to specify the offset values for the element from its normal position.
3. No Impact on Surrounding Elements: Unlike some other positioning values, using position: relative; doesn't affect the layout of surrounding elements. It just shifts the element visually while still occupying the same space in the document flow.

```
.container {
position: relative;
}

.box {
position: relative;
top: 20px;
left: 30px;
}
```

In this example, the .box element will be positioned 20 pixels down and 30 pixels to the right from its normal position within the .container element.

Remember that using position: relative; may not always be necessary, and it's often used in combination with other positioning properties to achieve specific layouts on a webpage.

# POSITION (FIXED)

In CSS, the position: fixed; property is used to position an element relative to the viewport, which means it will stay in the same position even when the page is scrolled. When you apply position: fixed; to an element, you can use the top, right, bottom, and left properties to specify its position.

```
.fixed-element {
position: fixed;
top: 10px;
 left: 20px;
}
```

# POSITION (STICKY)

Certainly! In CSS, the position: sticky property is used to make an element behave as relatively positioned until it crosses a specified point, at which it becomes fixed. This is commonly used for creating navigation bars that stick to the top of the page when scrolling down.

```
.sticky {
position: -webkit-sticky;
position: sticky;
top: 0;
/* Specifies the point at which the element will become fixed */
background-color: #f1f1f1;
padding: 50px;
}
```

In this example, the .sticky class will become fixed at the top of its container when the user scrolls past it. The top property determines the point where it becomes fixed.

It's important to note that browser support may vary, and the -webkit-sticky is included for compatibility with some older versions of WebKit-based browsers.

# POSITION (Z-INDEX)

In CSS, the z-index property is used to control the stacking order of positioned elements on a web page. The stacking order determines which elements appear on top of others when they overlap. The z-index property accepts numerical values, and elements with higher values will appear on top of elements with lower values.

Here's a brief explanation of how it works:

- Elements with a higher z-index value are stacked above elements with a lower value.
- If two elements have the same z-index, the one that comes later in the HTML source order will be placed on top.
- Elements with a z-index value other than auto create a stacking context, which means their children are stacked within that context, independent of the rest of the page.

```
#element1 {
z-index: 2;
}

#element2 {
z-index: 1;
}
```

In this example, #element1 will appear above #element2 because it has a higher z-index value.

Keep in mind that the z-index property only works on positioned elements (i.e., elements with a position value of relative, absolute, or fixed). If an element is not positioned, the z-index property will have no effect.

# LEVEL 9

# FLOAT PROPERTY

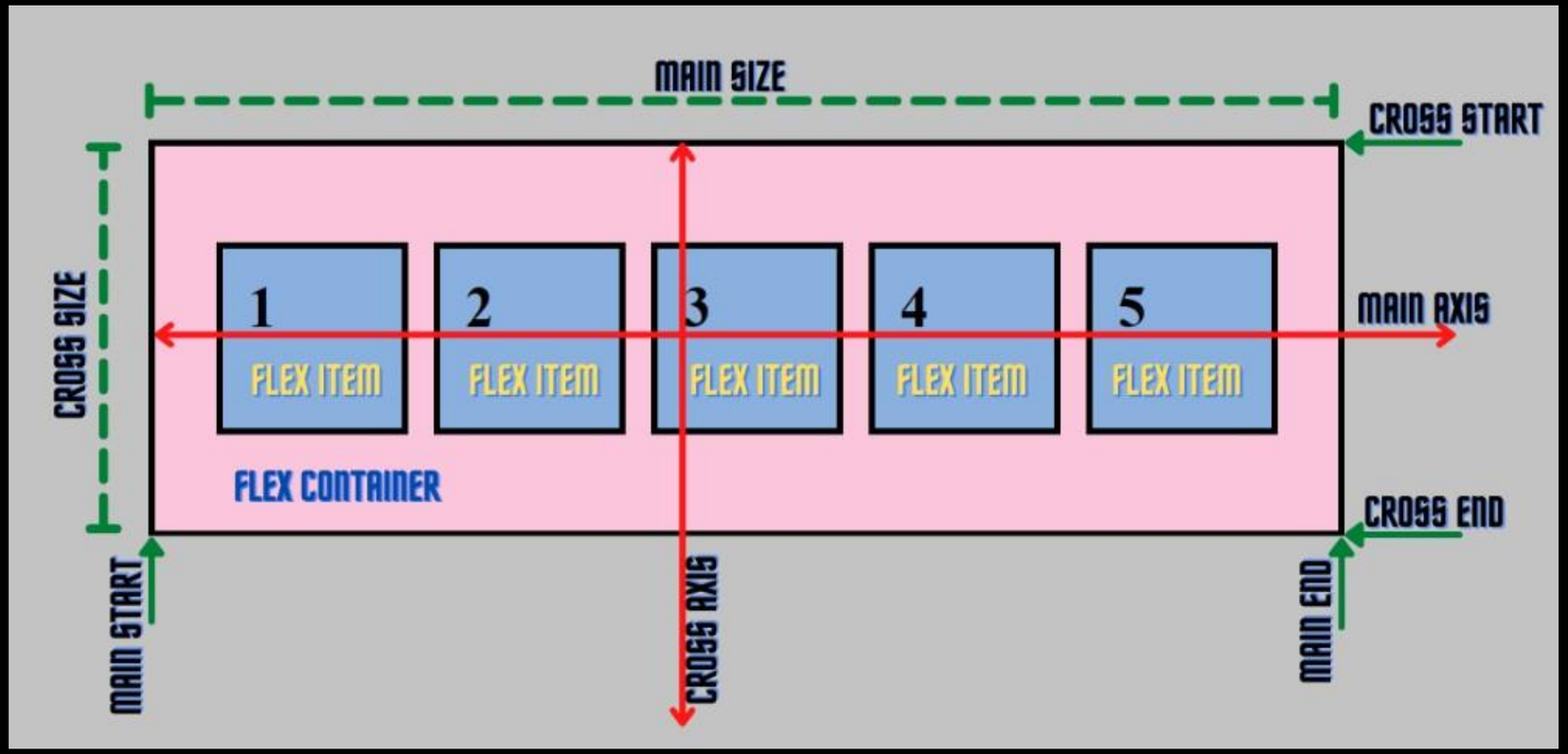
In CSS (Cascading Style Sheets), the float property is used to specify how an element should be positioned within its containing element. The float property can take one of the following values:

1. left: The element floats to the left of its containing element, allowing content to flow around it on the right side.
2. right: The element floats to the right of its containing element, allowing content to flow around it on the left side.
3. none (default): The element does not float, and content will not flow around it. This is the default value.

```
/* Floating an element to the left */
.float-left {
float: left;
}

/* Floating an element to the right */
.float-right {
float: right;
}

/* Clearing the float to prevent content from flowing around */
.clear-float {
clear: both;
}
```

# FLEXBOX MODEL

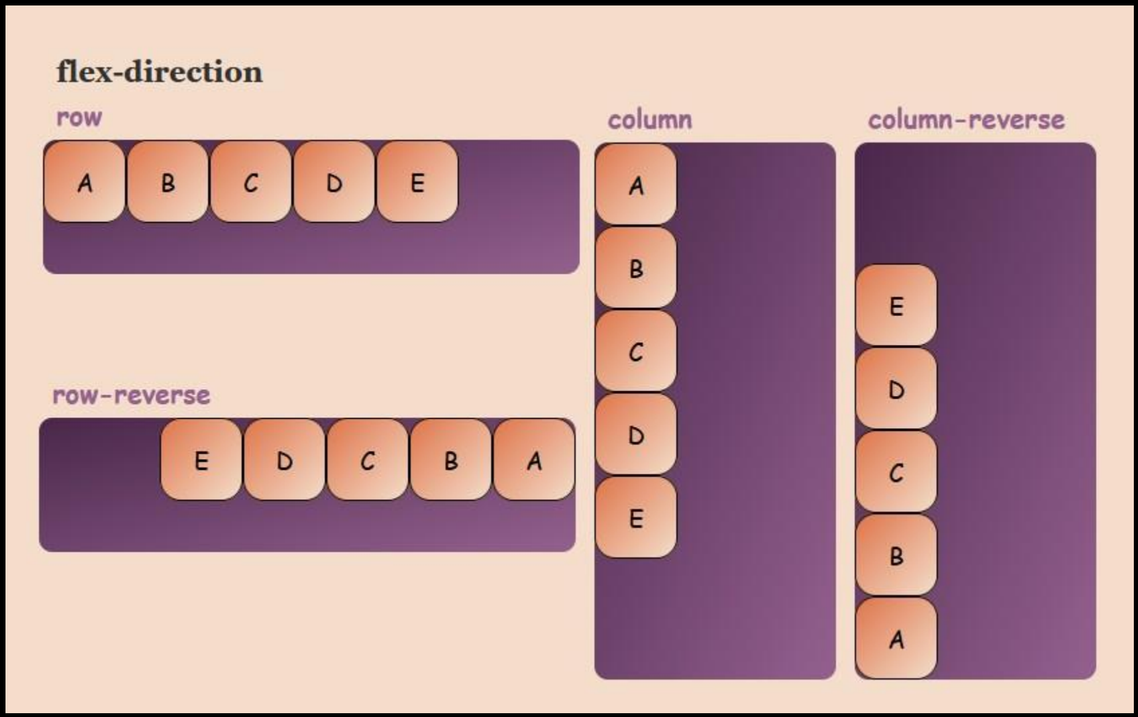
Flexbox, or the Flexible Box Layout, is a CSS (Cascading Style Sheets) layout model designed to provide a more efficient way to design and structure user interfaces. It allows you to create complex layouts with a more predictable and efficient arrangement of elements, especially when dealing with dynamic or responsive web designs.



# FLEX DIRECTION

The flex-direction property is used to set the direction of the flexible container's main axis. Here are the possible values for the flex-direction property:

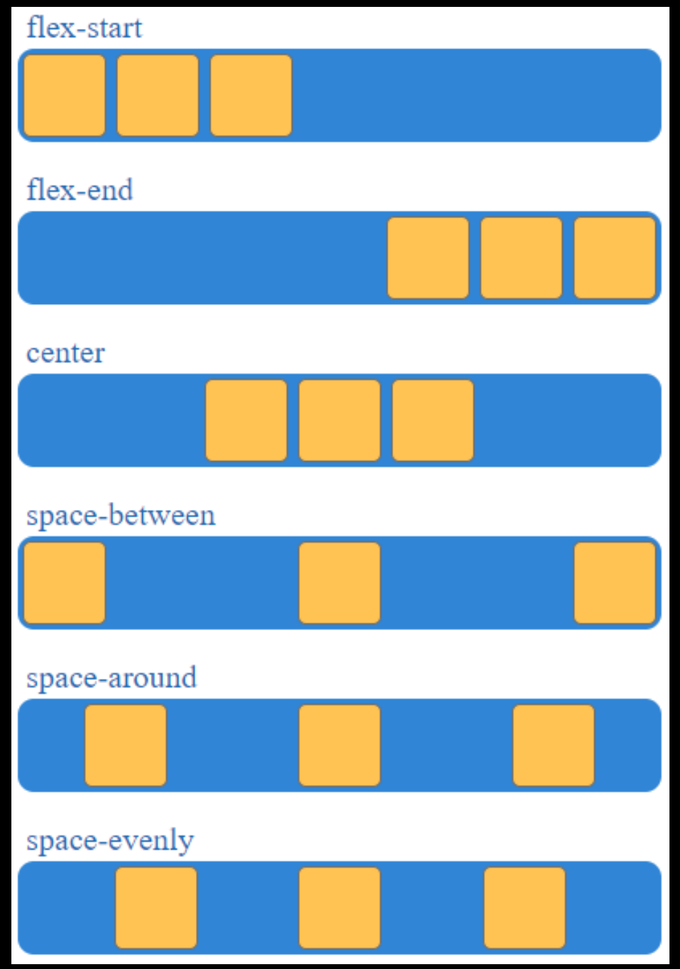
1. row: Items are placed along the horizontal axis, from left to right.
2. row-reverse: Items are placed along the horizontal axis, from right to left.
3. column: Items are placed along the vertical axis, from top to bottom.
4. column-reverse: Items are placed along the vertical axis, from bottom to top.




```
.container {
display: flex;
flex-direction:  row;
/* or row-reverse, column, column-reverse */
}
```

# FLEXBOX (JUSTIFY CONTENT)

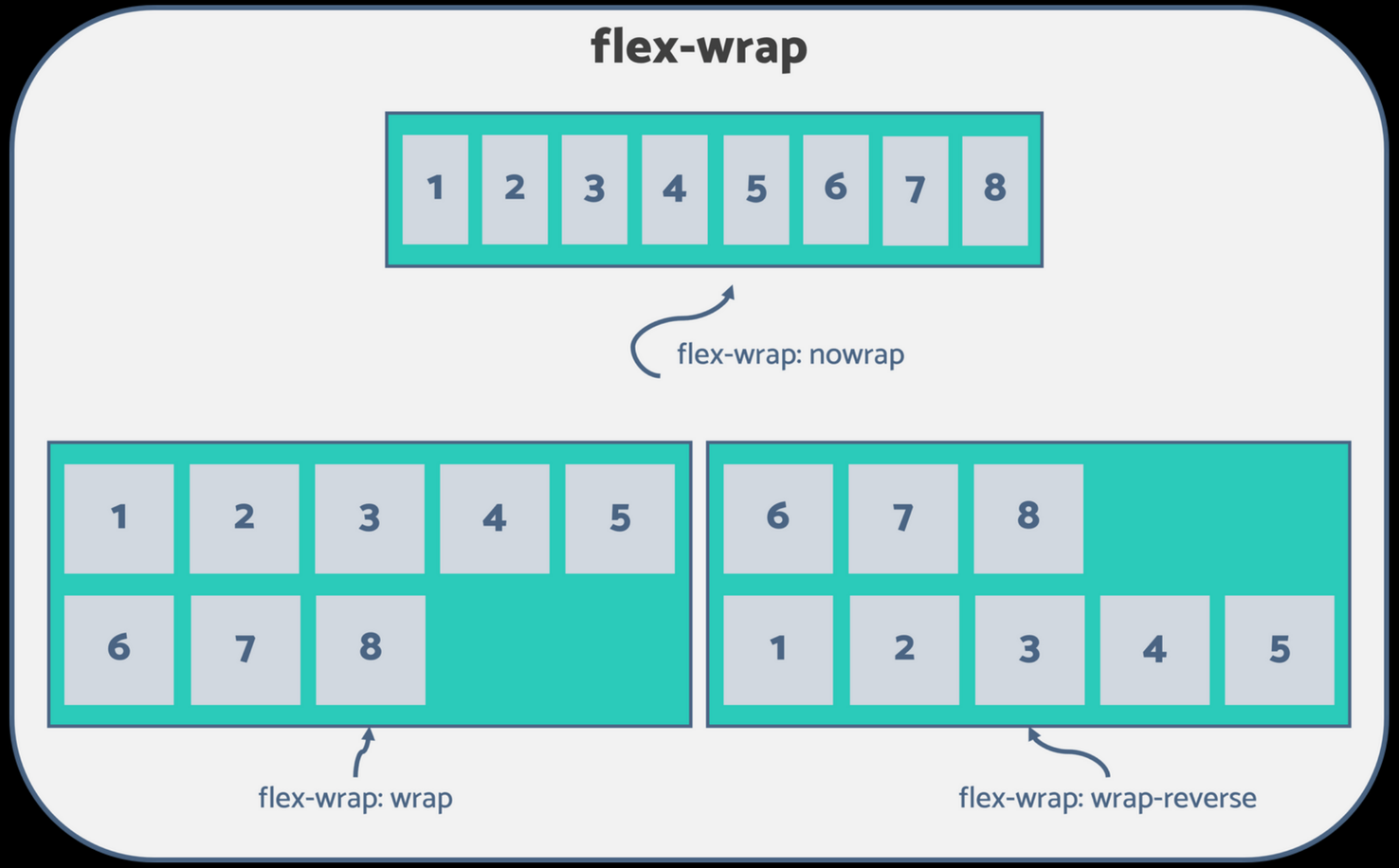
In CSS, the justify-content property is used to align and distribute space along the main axis of a flex container or a grid container. This property is particularly useful when working with flexbox or grid layouts. Here's a brief explanation of the justify-content property:




```
.container {
display: flex;
/* or display: grid; */
justify-content:  value;
}
```

# FLEX WRAP

Certainly! In CSS, the flex-wrap property is used in conjunction with the Flexible Box Layout (Flexbox) to control the wrapping behavior of flex containers. Here's some information about it:



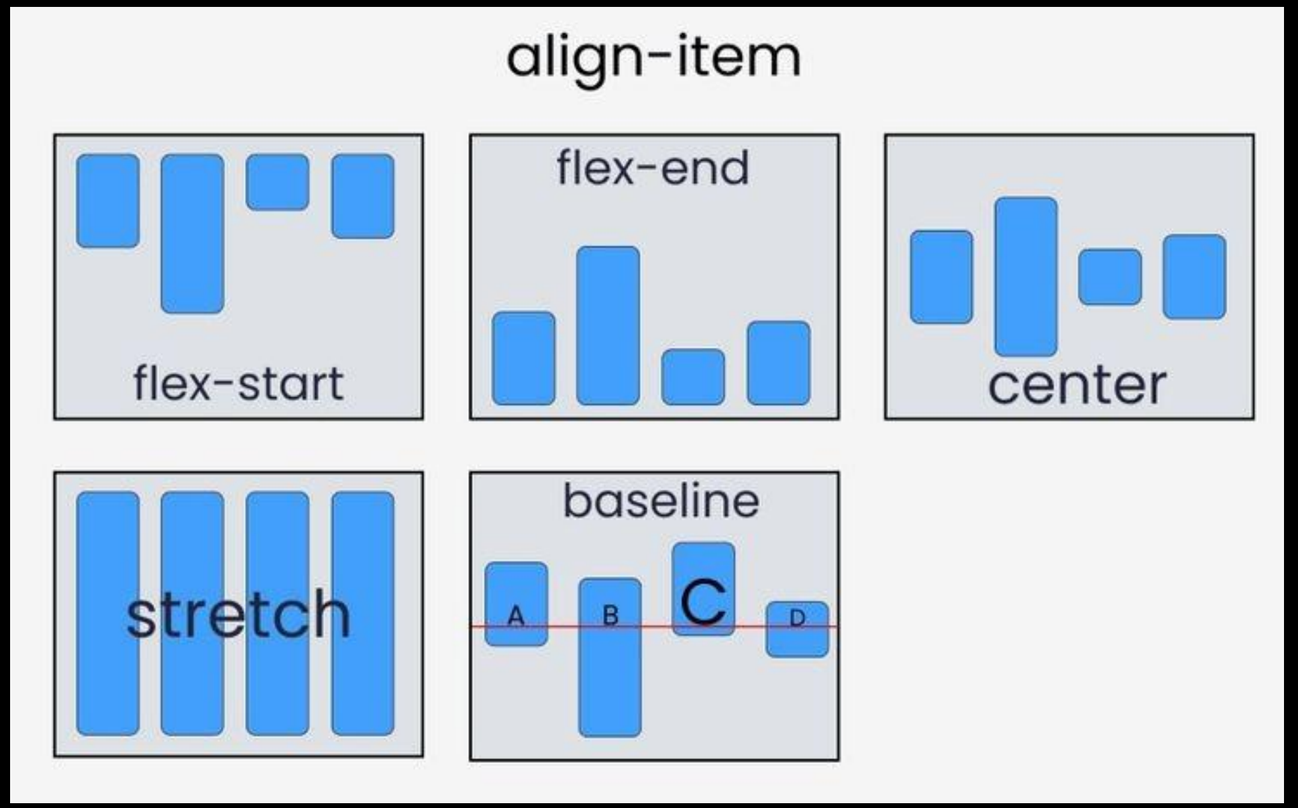

```
.container {
display: flex;
flex-wrap: wrap;
}
```

# FLEX (ALIGN-ITEMS)

**In Flexbox, the align-items property is used to define how flex items are aligned along the cross axis of the flex container.**

**Here are the possible values for the align-items property in Flexbox:**

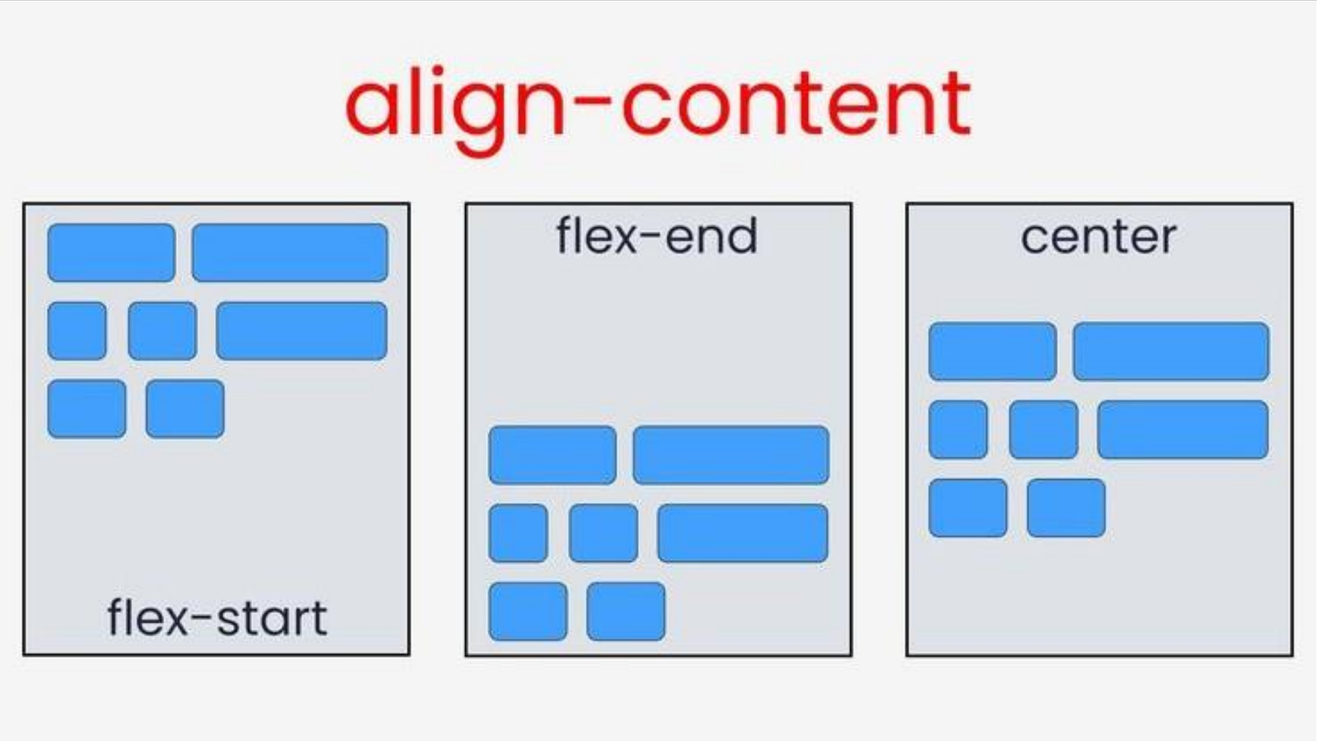
1. **flex-start: Items are aligned to the start of the cross axis.**
2. **flex-end: Items are aligned to the end of the cross axis.**
3. **center: Items are centered along the cross axis.**
4. **baseline: Items are aligned such as their baselines align.**
5. **stretch: Items are stretched to fill the container along the cross axis.**




```
.container {
display: flex;
align-items: center;
/* or any other value */
}
```

# FLEX (ALIGN-CONTENT)

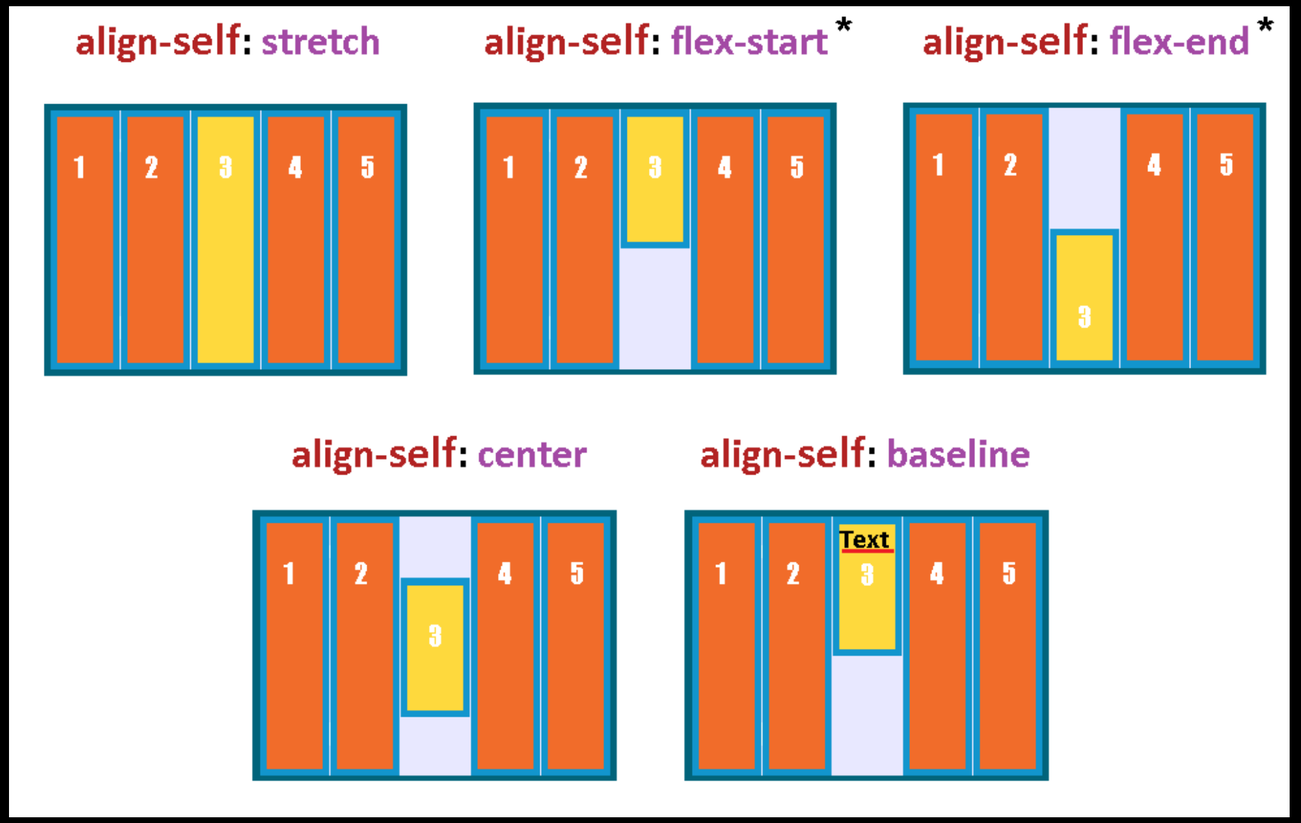
**The align-content property specifies how flex lines are distributed along the cross axis in a flexbox container.**




```
.flex-container {
display: flex;
align-content:  space-between;
/* or flex-start, flex-end, center,
space-around, space-evenly, stretch */
}
```

# FLEX (ALIGN-SELF)

**In CSS, the align-self property is used within a Flexbox or Grid layout to control the alignment of a specific item along the cross-axis (vertical axis for a row, horizontal axis for a column).**




```
.container {
display: flex; /* or display: grid; */
}

.item {
align-self: /* value */;
}
```

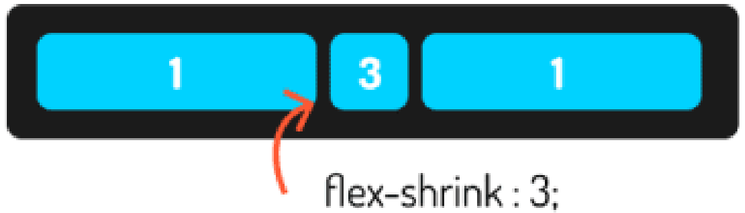
# FLEX SHRINK

Certainly! In CSS, the flex-shrink property is part of the Flexible Box Layout module (Flexbox). It is used to control the ability of a flex item to shrink if necessary. The flex-shrink property takes a unitless number as its value, which represents the factor by which the flex item will shrink compared to the other flex items in the flex container.

Here's a brief explanation:

- A flex-shrink value of 0 means the flex item will not shrink.
- A positive value for flex-shrink determines the relative shrink factor. For example, if one item has a flex-shrink of 2 and another has a flex-shrink of 1, the first item will shrink twice as much as the second one.





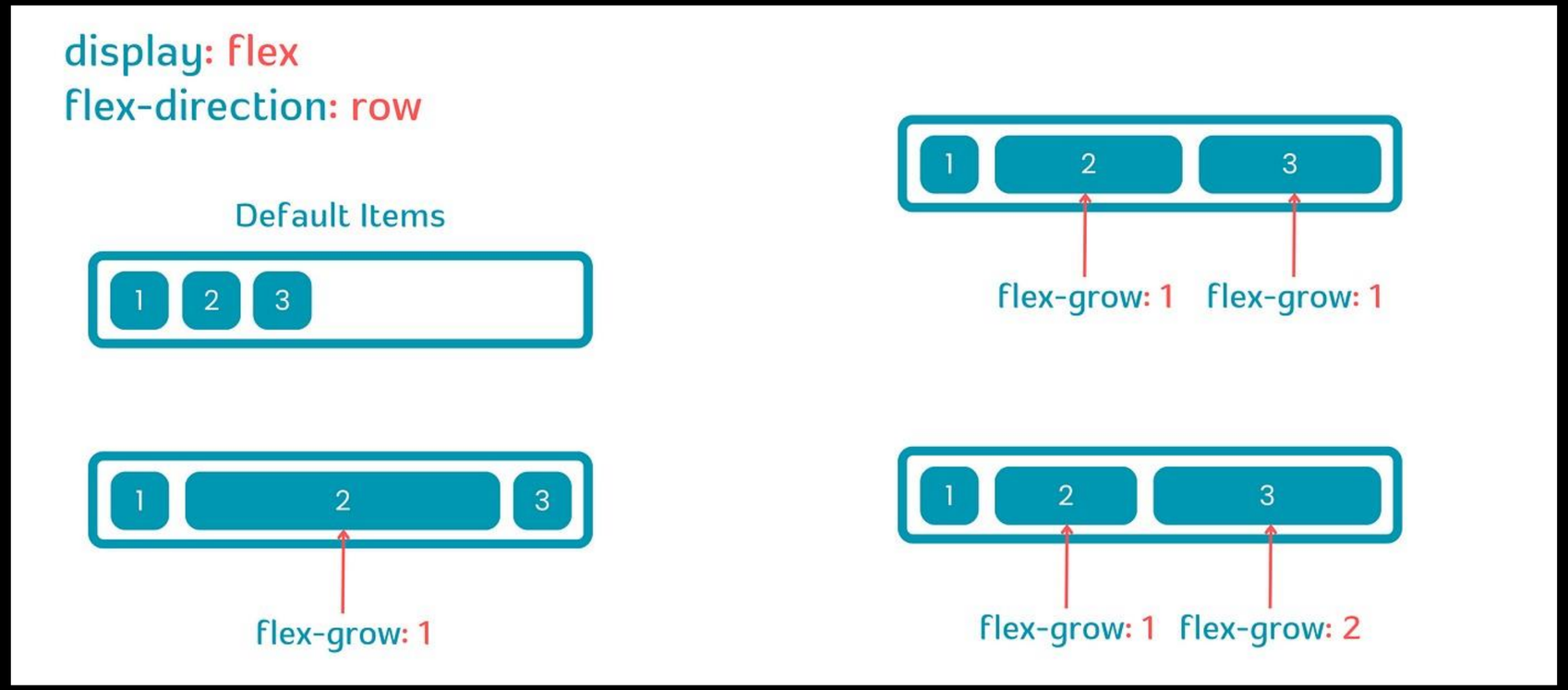

```
.flex-container {
display: flex;
}

.flex-item {
flex-shrink: 1;
/* You can set it to any positive value */
}
```

# FLEX GROW

Certainly! In CSS, the flex-grow property is used within a flex container to specify the ability of a flex item to grow if necessary. It determines the proportion of available space that the flex item should take up along the main axis.

Here's a brief explanation:

- flex-grow: This property accepts a unitless number as its value. It represents the factor by which the flex item should grow relative to the other flex items within the same flex container. A higher flex-grow value means the item will grow more compared to other items.




```
.flex-container {
display: flex;
}

.flex-item {
flex-grow: 1;
/* or any other numeric value */
}
```

# FLEX ORDER

Certainly! In CSS, the order property is used to define the order of flex items within a flex container. By default, all flex items have an order of 0, and they are displayed in the order they appear in the source code.

To change the order of flex items, you can use the order property and assign a numerical value to it. Items with a lower order value will appear before items with a higher order value. Negative values are also allowed.

```
.flex-container {
  display: flex;
}

.flex-item1 {
  order: 2;
}

.flex-item2 {
  order: 1;
}

.flex-item3 {
  order: 3;
}
```

In this example, flex-item2 will appear first, followed by flex-item1, and then flex-item3 because of the assigned order values.

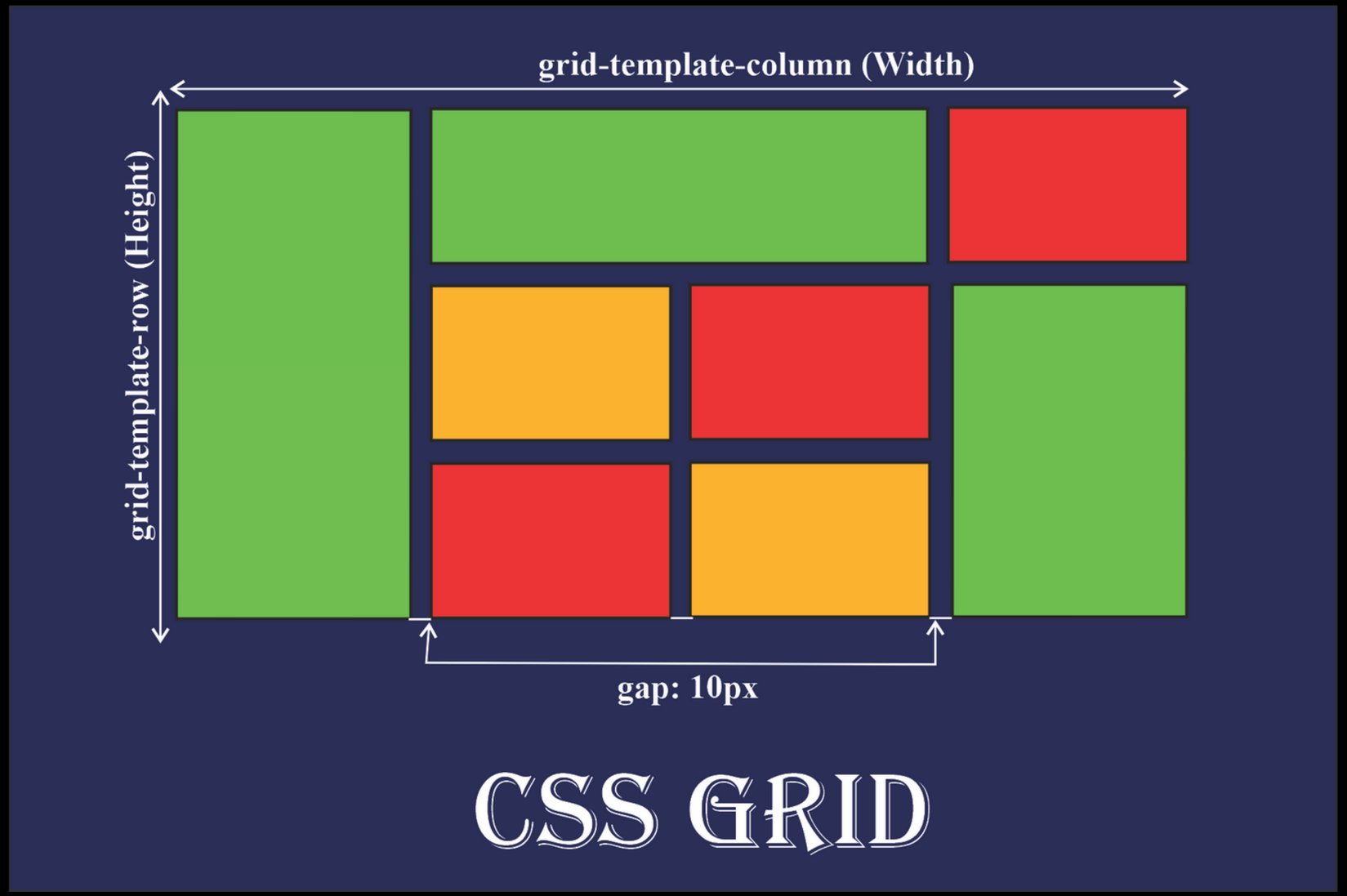
Remember that the order property only affects the visual order of the items, and it doesn't change their position in the source code.

# GRID MODEL

Certainly! In CSS (Cascading Style Sheets), the term "grid" usually refers to the CSS Grid Layout, which is a layout system that allows you to design complex web page layouts with rows and columns. Here's a brief overview:

## CSS Grid Layout:

- CSS Grid Layout is a two-dimensional layout system for the web.
- It lets you create layouts with rows and columns, providing a more flexible and powerful way to structure web pages compared to traditional methods.
- Grid Layout is supported by modern web browsers, making it a widely used feature in web design.




I am mentioning some of the most important properties of grid. Utilizing these properties gives you powerful control over the layout of your web page in a grid format.

**display: grid; : This property is used to define an element as a grid container, turning its direct children into grid items.**

**grid-template-rows and grid-template-columns: These properties define the number and size of rows and columns in the grid. You can use values like auto, percentages, or fixed lengths.**

```
grid-template-rows: 100px 200px;
/* Two rows with heights of 100px and 200px */
grid-template-columns: 1fr 2fr;
/* Two columns with a ratio of 1:2 */
```

grid-gap: This property sets the gap (space) between rows and columns. It is a shorthand for grid-row-gap and grid-column-gap.

```
grid-gap:  10px;
/* Sets a gap of 10px between rows and columns */
```

grid-template-areas: This property allows you to define named grid areas, making it easier to create complex layouts.

```
grid-template-areas:
"header header header"
"sidebar main main"
"footer footer footer";
```

**grid-row and grid-column: These properties can be used to place grid items into specific rows or columns.**

```
grid-row: 1 / 3; /* Places the item in rows 1 to 2 */
grid-column: 2 / 4; /* Places the item in columns 2 to 3 */
```

**justify-items and align-items: These properties control the alignment of grid items within their respective grid cells.**

```
justify-items: center;
/* Centers items horizontally in their grid cells */
align-items: end;
/* Aligns items to the bottom of their grid cells */
```

# LEVEL 10

# MEDIA QUERIES

In CSS (Cascading Style Sheets), a media query is a technique used to apply styles to a document based on certain conditions, such as the device's screen size, resolution, or other characteristics. Media queries allow developers to create responsive designs that adapt to different devices and screen sizes.

# The Syntax

declaration

Media Type

```
@media screen and (max-width: 768px){
  .container{
      // Write styles here
    }
}
```

styles to apply when all conditions are met

Specifying amout of screen to cover

Certainly! In CSS, media queries are used to apply different styles based on certain conditions, such as the width of the device or viewport. The width property in a media query is used to specify the maximum or minimum width at which the styles should be applied.

```
@media screen and (width: 600px) {
/* Styles to apply when the viewport width is exact 600 pixels */
body {
background-color:  lightblue;
}
```

```
@media screen and (min-width: 600px) {
/* Styles to apply when the viewport width is 600 pixels or wider */
body {
background-color: lightblue;
}
}

@media screen and (max-width: 800px) {
/* Styles to apply when the viewport width is 800 pixels or narrower */
body {
background-color: lightcoral;
}
}
```

# PSEUDO CLASSES

In CSS (Cascading Style Sheets), pseudo-classes are used to define styles for elements based on their state or position in the document tree. They allow you to select and style elements that are not represented by the HTML structure alone. Here are some commonly used pseudo-classes:

:hover: Selects and styles an element when the user hovers over it.

```
a:hover {
color: red;
}
```

:active: Selects and styles an element when it is being activated (clicked on) by the user.

```
button:active {
background-color:  #00ff00;
}
```

:focus: Selects and styles an element that currently has focus (e.g., when a form input is selected).

```
input:focus {
border: 2px solid blue;
}
```

:first-child: Selects and styles the first child element of a parent.

```
li:first-child {
font-weight: bold;
}
```

:last-child: Selects and styles the last child element of a parent.

```
p:last-child {
margin-bottom: 0;
}
```

:nth-child(): Selects and styles elements based on their position in the parent.

```
li:nth-child(2n) {
background-color:  #f0f0f0;
}
```

# LEVEL 11

# TRANSITIONS PROPERTY

In CSS (Cascading Style Sheets), a "transition" refers to the gradual change of a property's value over a specified duration. This effect is commonly used to create smooth animations and enhance user interfaces. Here's a basic explanation:

## SYNTAX :

```
element {
transition: property duration timing-function delay;
}
```

- property: Specifies the CSS property you want to apply the transition to (e.g., color, width, opacity).
- duration: Defines the duration of the transition in seconds (s) or milliseconds (ms).
- timing-function: Describes the acceleration curve of the transition (e.g., linear, ease, ease-in-out).
- delay: Optional parameter, specifying the delay before the transition starts.

```
.box {
width: 100px;
height: 100px;
background-color: blue;
transition: width 2s ease-in-out;
}

.box:hover {
width: 200px;
}
```

In this example, when you hover over the element with the class "box," the width will transition from 100px to 200px over a period of 2 seconds with an ease-in-out timing function.

# TRANSFORM PROPERTY

Certainly! In CSS (Cascading Style Sheets), the transform property is used to apply 2D and 3D transformations to an element. It allows you to rotate, scale, skew, or translate elements. Here are some common transformations you can apply using the transform property:

Translate: Moves an element from its current position on the page.

```
transform: translate(50px, 100px);
```

Rotate: Rotates an element by a specified angle.

```
transform: rotate(45deg);
```

Scale: Changes the size of an element.

```
transform: scale(1.5);
```

Skew: Skews an element along the X and Y axes.

```
transform: skew(30deg, 20deg);
```

# ANIMATION PROPERTY

The "animation" CSS property is used to apply animations to elements on a web page. It allows you to create dynamic and interactive effects, enhancing the user experience. Here's a brief overview of how the "animation" property works:

## SYNTAX

```
selector {
animation: name duration timing-function delay
iteration-count direction;
}
```

- **name: Specifies the name of the animation, defined using the @keyframes rule.**

- **duration: Specifies how long the animation takes to complete. It is measured in seconds (s) or milliseconds (ms).**

- **timing-function: Defines the timing of the animation, such as linear, ease-in, ease-out, ease-in-out, etc.**

- delay: Specifies a delay before the animation starts. It is measured in seconds (s) or milliseconds (ms).

- iteration-count: Defines the number of times the animation should repeat. It can take a numerical value or "infinite" for continuous looping.

- direction: Sets the direction of the animation, either "normal" (default), "reverse," "alternate," or "alternate-reverse."

```
@keyframes slide-in {
from {
transform: translateX(-100%);
}
to {
transform: translateX(0);
}
}

.element {
animation: slide-in 1s ease-in-out 0.5s infinite alternate;
}
```

**This example defines a keyframe animation called "slide-in" and applies it to an element, making it slide in from the left with a duration of 1 second, an ease-in-out timing function, a 0.5-second delay, and infinite alternate iterations.**

TThhaannkk *you!*