**Selectors in CSS**

A CSS selector is a pattern used to select the HTML elements you want to style. CSS selectors can range from simple to complex, allowing you to target

elements in various ways.

Selectors

Attribute selectors

Combinator

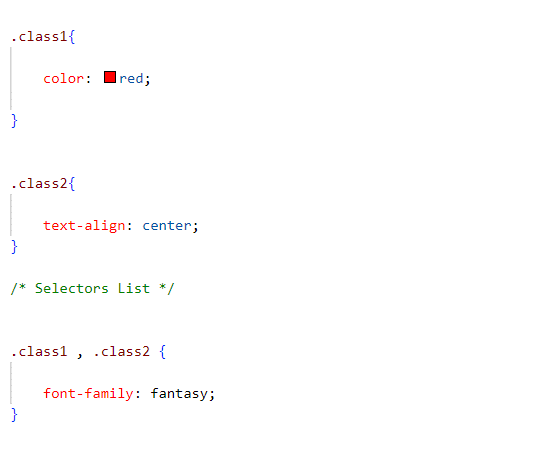
Pseudo class Selector

Pseudo Elements

Simple Selectors

1. Descendant selector
2. Child selector
3. Adjacent sibling selector
4. General sibling selector
5. Descendant selector
6. Child selector
7. Adjacent sibling selector
8. General sibling selector
9. Universal Selectors
10. Element Selectors
11. Class Selectors
12. ID Selectors
13. Selectors list
14. **Simple Selectors**

Simple selectors in CSS are used to select elements based on their name, id, class, or attribute.

1. **Universal Selectors (\*)**:
   * Selects all elements on a page.
   * Example: \* { margin: 0; } applies zero margin to all elements.
2. **Element Selectors**:
   * Selects all elements of a given type.
   * Example: p { font-size: 16px; } selects all <p> elements and sets their font size to 16px.
3. **Class Selectors (.)**:
   * Selects all elements with a specific class attribute.
   * Example: .highlight { background-color: yellow; } selects all elements with the class highlight and sets their background color to yellow.
4. **ID Selectors (#)**:
   * Selects a single element with a specific ID attribute.
   * Example: #header { color: blue; } selects the element with the ID header and sets its text color to blue.
5. **Selectors List**:
   * Combines multiple selectors separated by commas, applying the same styles to all selected elements.
   * Example: h1, h2, h3 { margin-bottom: 10px; } selects all <h1>, <h2>, and <h3> elements and sets their bottom margin to 10px.
6. **Combinator**

Combinators in CSS are tools used to select elements based on their relationships with other elements.

1. **Descendant Combinator ( )**:

* Selects elements that are inside another element.
* Example: div p { color: red; } changes the text color of all <p> tags inside <div> tags to red.

Valid HTML Structure:

* Do not nest block-level elements inside <p> tags.
* Ensure CSS selectors reflect the valid HTML structure.

**Rules**:

<p> (paragraph) tags cannot contain block-level elements (e.g., <h1>, <div>, <ul>, etc.).

<p>

<h1>Invalid HTML</h1>

</p>

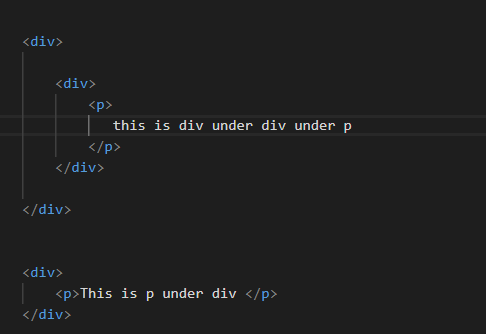
**Correct HTML Structure**

<div>

<p>This is a paragraph.</p>

<h1>This is a heading.</h1>

</div>



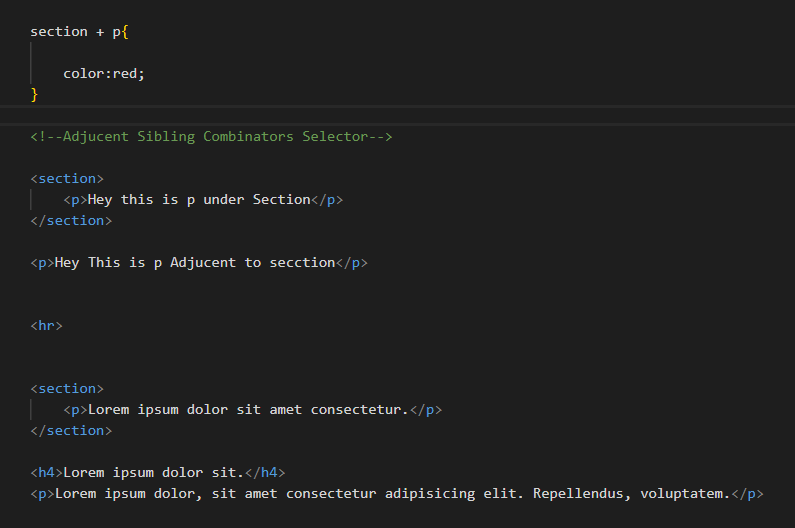
या मध्ये Syntax: ancestor descendant हे pair सगळे कडे सेम राहणार like **div p हे** जिथे असेल तिथ त्यची property लागते

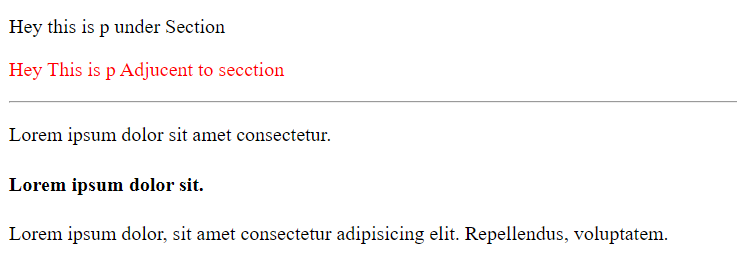
1. **Child Combinator (>)**:

* Selects elements that are directly inside another element.
* Example: ul > li { margin-bottom: 5px; } changes the bottom margin of <li> tags that are directly inside <ul> tags to 5px.

1. **Adjacent Sibling Combinator (+)**:

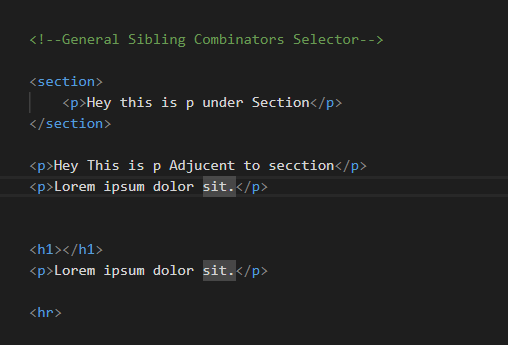
* Selects an element that comes immediately after another element.
* Example: h1 + p { margin-top: 0; } removes the top margin of a <p> tag that comes right after an <h1> tag.

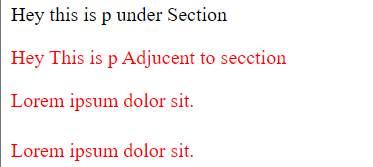




1. **General Sibling Combinator (~)**:

* Selects all elements that come after another element, even if there are other elements in between.
* Example: h1 ~ p { color: blue; } changes the text color of all <p> tags that come after an <h1> tag to blue, no matter how many tags are in between.





**3.Attribute selectors**

**Attribute Selectors**: Used to select elements based on the presence or value of their attributes.

1. **[attr]**:

* Selects elements with the specified attribute.
* Example: [type] { color: red; } selects all elements with a type attribute.

2. **[attr="value"]**:

* Selects elements with the specified attribute and value.
* Example: [type="text"] { background-color: yellow; } selects all elements with type="text".

3. **[attr~="value"]**:

* Selects elements with the specified attribute whose value is a space-separated list containing at least one specified value.
* Example: [class~="btn"] { font-weight: bold; } selects elements with a class attribute containing the word btn.

4. **[attr|="value"]**:

* Selects elements with the specified attribute whose value is either exactly "value" or starts with "value-" (used for language subcode).
* Example: [lang|="en"] { color: blue; } selects elements with a lang attribute value of "en" or starting with "en-".

5. **[attr^="value"]**:

* Selects elements with the specified attribute whose value begins with the specified value.
* Example: [href^="https"] { color: green; } selects elements with href attributes starting with "https".

6. **[attr$="value"]**:

* Selects elements with the specified attribute whose value ends with the specified value.
* Example: [src$=".jpg"] { border: 1px solid black; } selects elements with src attributes ending in ".jpg".

7. \**[attr*="value"]\*\*:

* Selects elements with the specified attribute whose value contains the specified value.
* Example: [title\*="flower"] { font-style: italic; } selects elements with title attributes containing the word "flower".

**BOX Model & Background**

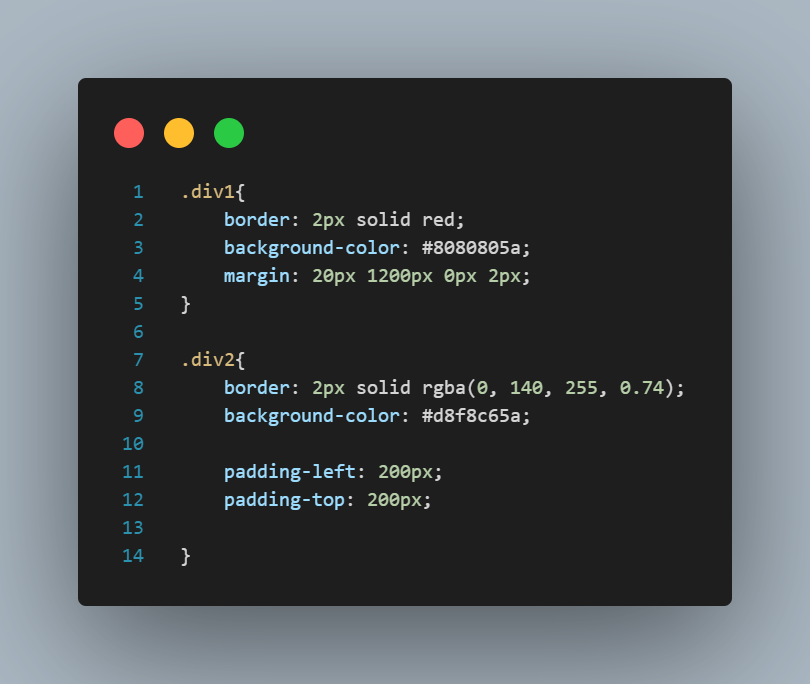
**Margine-Padding**

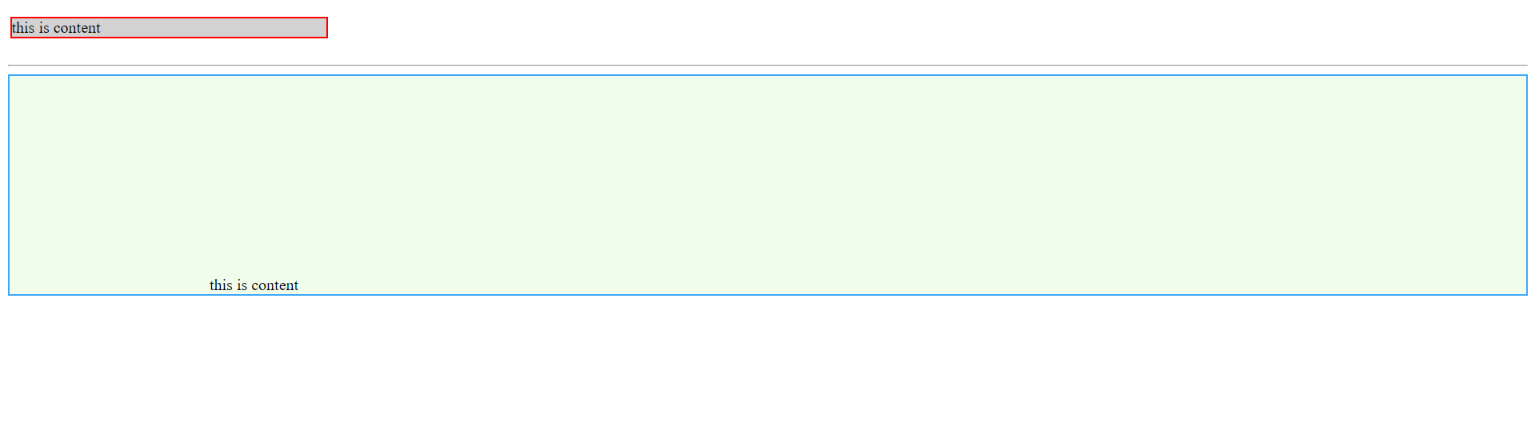
**Margin**

* **Margin**: Space outside the border of an element.
* **Syntax**:
  + **Single value**: margin: 10px; (applies to all sides)
  + **Two values**: margin: 10px 20px; (first value for top & bottom, second for left & right)
  + **Three values**: margin: 10px 20px 30px; (top, horizontal, bottom)
  + **Four values**: margin: 10px 20px 30px 40px; (top, right, bottom, left)
* **Individual sides**:
  + margin-top: 10px;
  + margin-right: 20px;
  + margin-bottom: 30px;
  + margin-left: 40px;

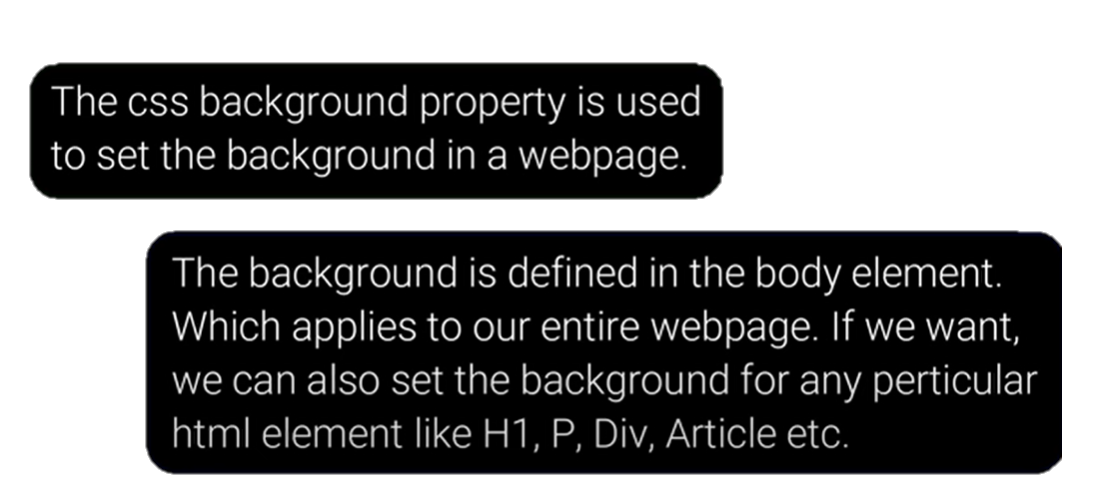
**Padding**

* **Padding**: Space inside the border of an element.
* **Syntax**:
  + **Single value**: padding: 10px; (applies to all sides)
  + **Two values**: padding: 10px 20px; (first value for top & bottom, second for left & right)
  + **Three values**: padding: 10px 20px 30px; (top, horizontal, bottom)
  + **Four values**: padding: 10px 20px 30px 40px; (top, right, bottom, left)
* **Individual sides**:
  + padding-top: 10px;
  + padding-right: 20px;
  + padding-bottom: 30px;
  + padding-left: 40px;





**Background Properties**





**CSS Background Properties: Sorted and Important Notes**

**1. background-color**

* **Sets the background color** of an element.
* **Syntax**: background-color: color;
* **Example**: background-color: #ff0000;

**2. background-image**

* **Sets the background image** of an element.
* **Syntax**: background-image: url('image.jpg');
* **Example**: background-image: url('pattern.png');

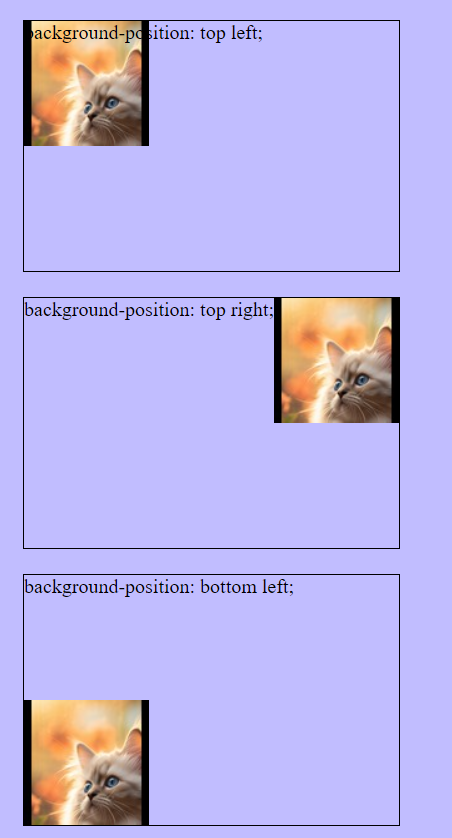
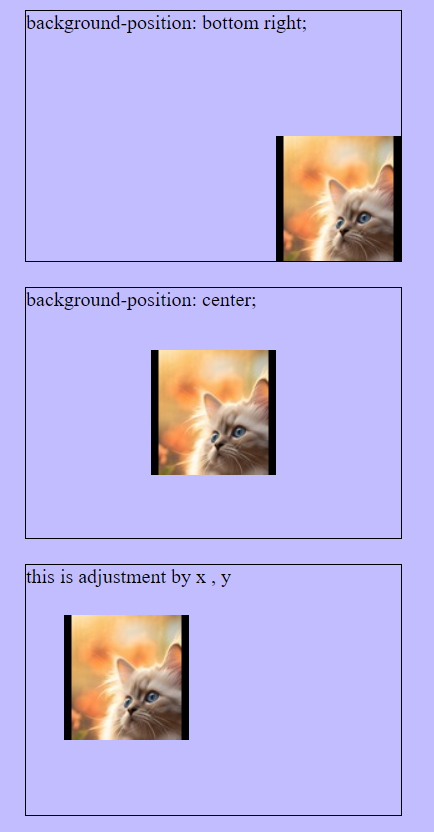
**3. background-repeat**

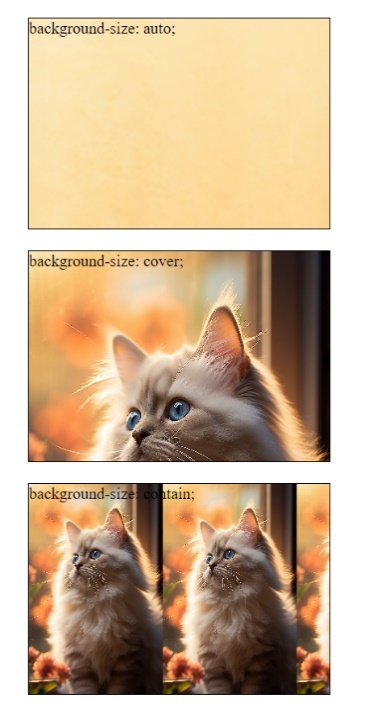
* **Defines how the background image is repeated**.
  + repeat: Repeats both horizontally and vertically (default).
  + repeat-x: Repeats horizontally.
  + repeat-y: Repeats vertically.
  + no-repeat: No repetition.
* **Syntax**: background-repeat: value;
* **Example**: background-repeat: no-repeat;



**4. background-position**

* **Sets the initial position** of the background image.
* **Syntax**: background-position: x-pos y-pos;
* **Example**: background-position: center center;



**5. background-size**

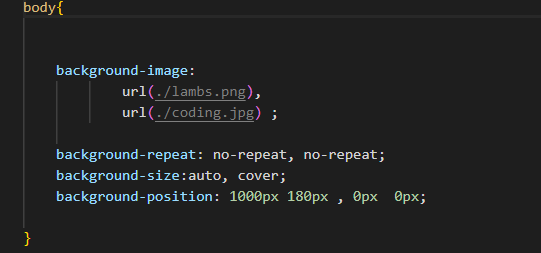
* **Specifies the size** of the background image.
  + cover: Scales the image to cover the entire element.
  + contain: Scales the image to be fully visible within the element.
* **Syntax**: background-size: value;
* **Example**: background-size: cover;

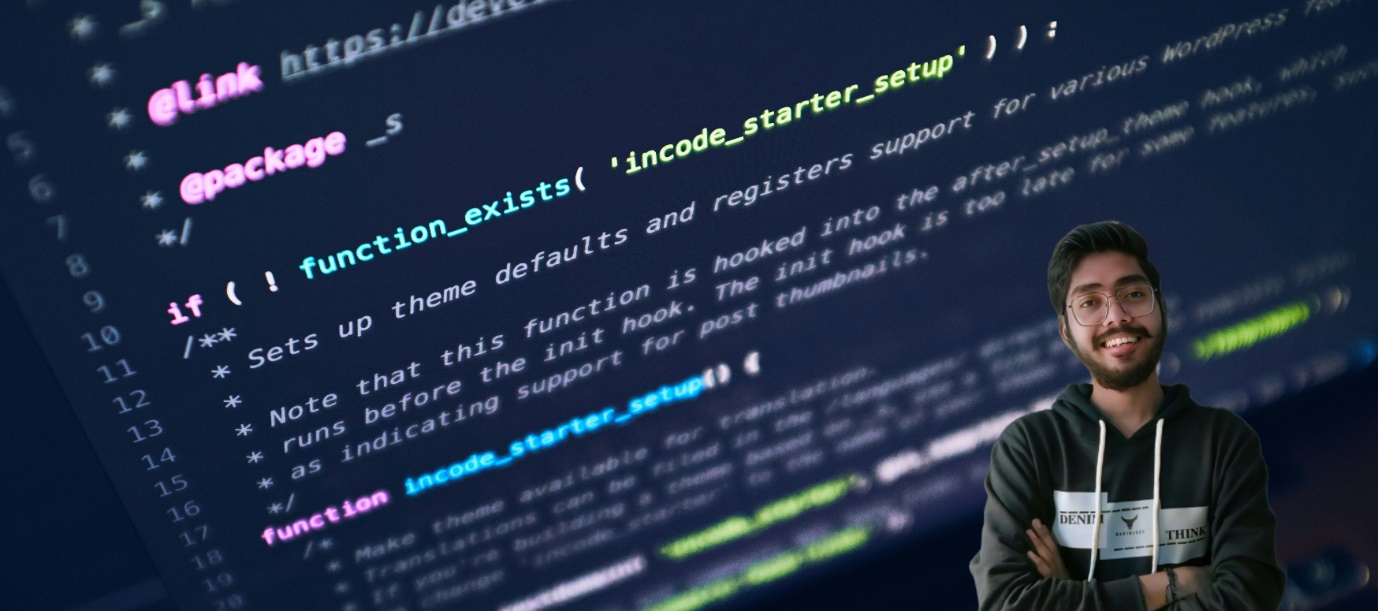
 **auto**: The background image retains its original dimensions.

 **cover**: The background image covers the entire element, maintaining aspect ratio but possibly cropping the image.

 **contain**: The background image is fully visible within the element, maintaining aspect ratio but possibly leaving empty space.

**Multiple Background:**





**6. background-attachments**

* **Sets whether a background image is fixed** or scrolls with the rest of the page.
  + **scroll**: Background scrolls with the element (default).
  + **fixed**: Background is fixed with regard to the viewport.
  + **local**: Background scrolls with the element's contents.
* **Syntax**: background-attachment: value;
* **Example**: background-attachment: fixed;

About the local:

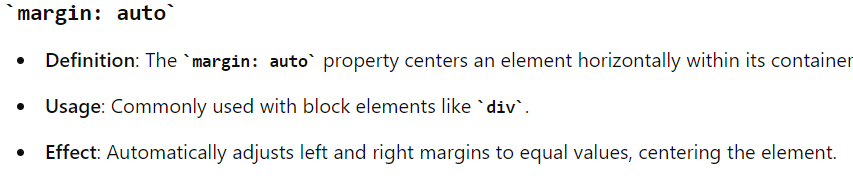
When you scroll the content inside the element, the background image moves along with it.

या मध्ये जेवा आपण block वर block ठवतो तेचा जर आपल्याला scroll property use करता येत नाहीत सो we use local

Module 19

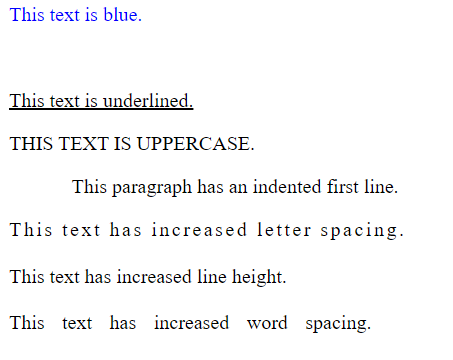
* **Margin And Padding**
* Text In CSS
* Font In CSS
* Display
* Height And Width In CSS

**Margin-Padding**

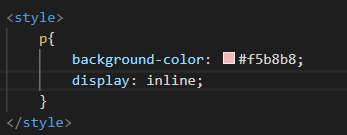


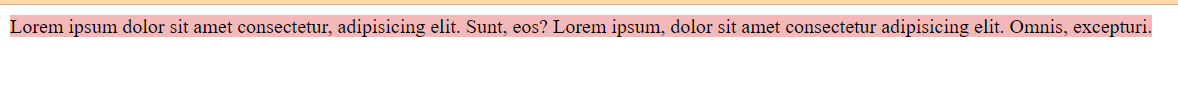
**Text properties And Font Property**





* **Display Properties**
  1. Display-inline
* The element is displayed as an inline element (takes up only as much width as necessary and does not start on a new line).
* This makes block element in inline



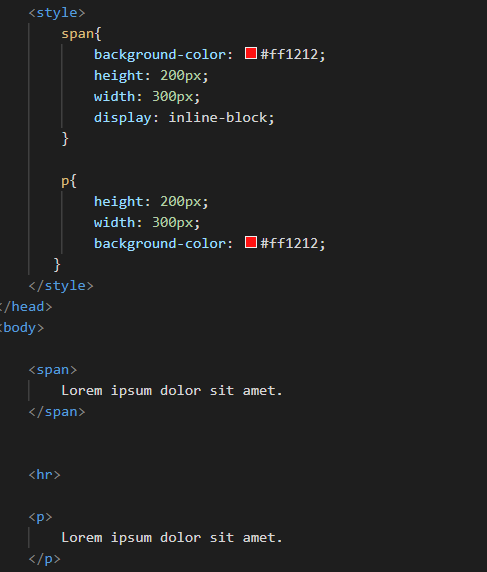


* 1. Display-Block

**Description**: The element is displayed as a block element (takes up the full width available and starts on a new line).

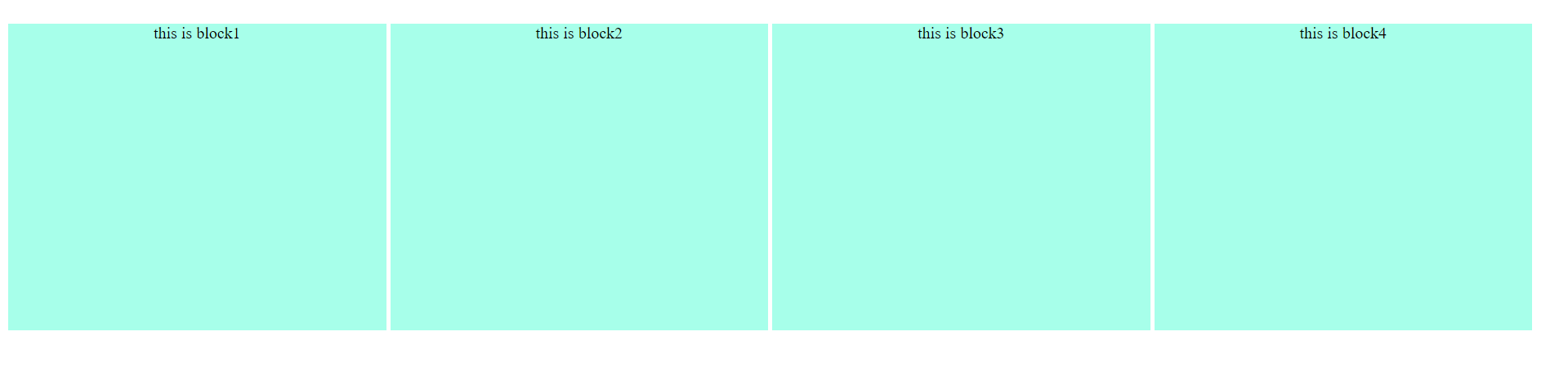
* 1. Inline-block

display: inline-block is a versatile CSS property that allows elements to behave like inline elements in terms of layout (sitting next to each other) while also having the block-like ability to accept width, height, padding, and margin settings. This makes it ideal for creating complex and responsive layouts that require precise control over element dimensions and positioning.



**key Points:**

* **Inline Layout with Block Capabilities**: Elements can sit next to each other while having dimensions set.



* **Alignment and Spacing**: Provides better control over alignment and spacing compared to pure inline or block elements.
* **Responsive Design**: Useful in responsive design for creating adaptable and flexible layouts.

END OF SECTION

**Height and Width**

**List of Contents:**

1. Width

2. Min-width

3. Max-width

4. Height

5. Min-height

6. Max-height

7. Box-sizing

Read the notebook notes

**1. Pseudo Class :**

Module 20

* **Pseudo Classes**
* Gradients
* Transitions
* Animation

**Definition**: Pseudo-class selectors in CSS are used to define the special states of elements. They allow you to apply styles to elements based on their state or position within the document structure, without the need to add classes or IDs.

**Key Points:**

* **State-Based Styling**: Apply styles based on the state of an element (e.g., :hover, :focus, :active).
* **Structural Styling**: Target elements based on their position in the document (e.g., :first-child, :nth-child()).
* **User Interaction**: Change the appearance of elements based on user actions (e.g.,: hover when a user hovers over an element).

Most frequently used CSS Pseudo class selectors.

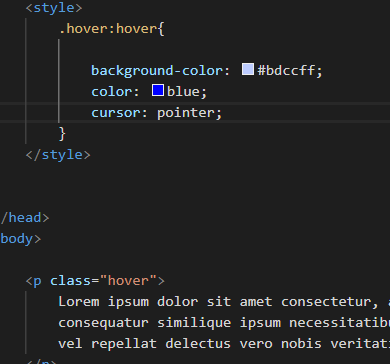
1. **: hover**: Interaction on hover.
2. **: focus**: Interaction when focused.
3. **:** **link**: Unvisited links.

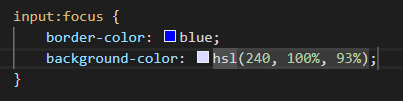
**: visited**: Visited links.

**: active**: Interaction on click.

1. **: first-child**: First child element.
2. **: lang**: Language-specific styling.
3. **: nth-child ()**: Styling elements based on position
4. Hover

***Selector/class/id:hover****{  
 }*

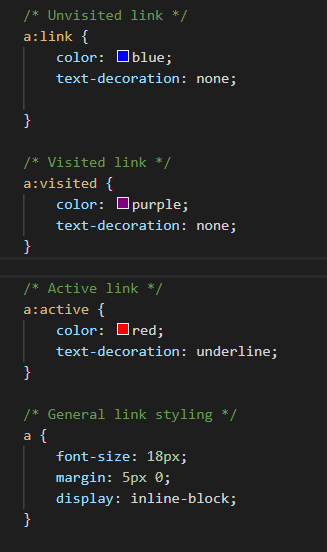


  
2.Focus**:**

  <input type="text">

3.Link:

* **link**: Unvisited links.
* **: visited**: Visited links.
* **: active**: Interaction on click.



* 1. first-child:
  2. nth child ():

<https://github.com/Lambodar2001/Web-Development-/blob/2cf7ddb48565dad1dd5a1d62d170e1356abfb7f5/2.CSS/CSS%209/index1.html>

 Descendant **Combinator**: Targets all levels of descendants.

 Child **Combinator**: Targets only direct children.

 First**-Child**: Targets the first child of a parent. – pseudo

* **Simplest Description**

**First Child (:first-child)**

* **What it does**: Selects the first element inside a parent.
* **Imagine**: You have a list of items. The :first-child selector will pick the very first item in that list.

**nth-child (:nth-child(n))**

* **What it does**: Selects the nth element inside a parent, where n is a number you specify.
* **Imagine**: You have a list of items. The :nth-child(2) selector will pick the second item in that list, :nth-child(3) will pick the third, and so on.

**Examples**

* **:first-child**: Think of a group of boxes. If you apply a style with :first-child, only the first box gets styled.
* **:nth-child(3)**: Think of a group of boxes. If you apply a style with :nth-child(3), only the third box gets styled. You can replace 3 with any number to style the corresponding box.

* here's a list of frequently used nth-child formulas:

1. **Every Even Child (nth-child(even) or nth-child(2n))**
2. **Every Odd Child (nth-child(odd) or nth-child(2n+1))**
3. **Every 3rd Child (nth-child(3n))**
4. **Every 3rd Child Starting from 1 (nth-child(3n+1))**
5. **First Child (nth-child(1))**
6. **Last Child (nth-last-child(1))**
7. **First 5 Children (nth-child(-n+5))**
8. **Children Starting from the 3rd (nth-child(n+3))**
9. **Every 4th Child Starting from 2 (nth-child(4n+2))**
10. **Every 5th Child (nth-child(5n))**

* **Understanding n in the :nth-child Formula**
* **n**: This is a variable that starts at 0 and increments by 1. It can be used to create patterns for selecting elements.

**Common Patterns**

1. **nth-child(n)**:
   * Selects every element (acts like a wildcard, since every element fits the formula).
2. **nth-child(2n)**:
   * Selects every even-numbered element.
   * Pattern: 2, 4, 6, 8, ...
3. **nth-child(2n+1)**:
   * Selects every odd-numbered element.
   * Pattern: 1, 3, 5, 7, ...
4. **nth-child(3n)**:
   * Selects every third element.
   * Pattern: 3, 6, 9, 12, ...
5. **nth-child(3n+1)**:
   * Selects every third element starting from the first.
   * Pattern: 1, 4, 7, 10, ...
6. **nth-child(-n+3)**:
   * Selects the first three elements.
   * Pattern: 1, 2, 3
7. **nth-child(n+3)**:
   * Selects all elements starting from the third.
   * Pattern: 3, 4, 5, 6, ...

**Example Breakdown**

* **nth-child(2n)**:
  + n = 0: 2\*0 = 0 (not selected, as nth-child starts from 1)
  + n = 1: 2\*1 = 2 (selects 2nd child)
  + n = 2: 2\*2 = 4 (selects 4th child)
  + n = 3: 2\*3 = 6 (selects 6th child)
  + ...
* **nth-child(3n+1)**:
  + n = 0: 3\*0 + 1 = 1 (selects 1st child)
  + n = 1: 3\*1 + 1 = 4 (selects 4th child)
  + n = 2: 3\*2 + 1 = 7 (selects 7th child)
  + n = 3: 3\*3 + 1 = 10 (selects 10th child)
  + ...

Using n in the formula allows you to create flexible and powerful patterns for selecting elements based on their order in the document.

1. **Pseudo Elements ::**

Pseudo-elements are like virtual elements you can style in CSS to enhance how content looks or behaves without adding extra HTML. Here's what each does:

1. **::first-line**: Styles the first line of text in a block.
2. **::first-letter**: Styles the first letter of a block of text, like making it bigger or a different color.
3. **::after**: Adds content after an element's content, like inserting icons or text after a paragraph.
4. **::before**: Adds content before an element's content, useful for adding icons or labels before text.
5. **::marker**: Styles list markers, like bullets or numbers in lists.
6. **::selection**: Styles the text that's selected by the user on a webpage, changing its color or background.

<https://github.com/Lambodar2001/Web-Development-/blob/8dbe51f0db322f5cf9495f461abc3ed3a986c1ef/2.CSS/CSS%209/index2.html>

1. **Gradient**

List of Content

1. Gradient

2. Linear gradient

3. Radial gradient

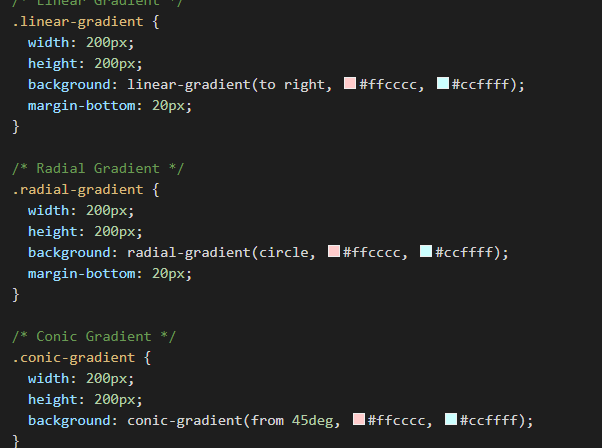
4. Conic gradient

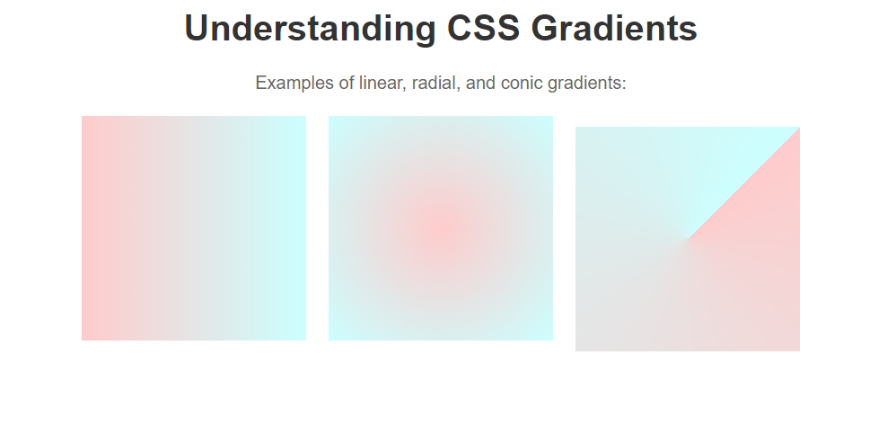
It lets you display smooth transitions between two or more specified colors.

**CSS defines three types of gradients:**

**● Linear Gradients (goes down/up/left/right/diagonally)**

**● Radial Gradients (defined by their center)**

**● Conic Gradients (rotated around a center point)**



1. **Transition**

**1. Transitions**

**2. Tooltips**

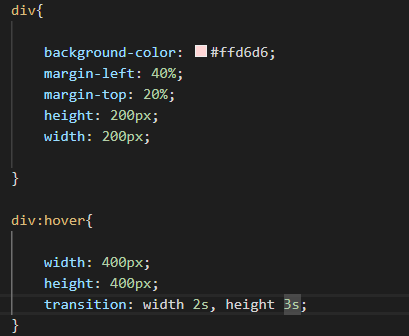
1. **Transition**:

References: [**https://github.com/Lambodar2001/Web-Development-/blob/179208d38401e9d37f7812a089e4b7e30efc7524/2.CSS/CSS%209/index4.html**](https://github.com/Lambodar2001/Web-Development-/blob/179208d38401e9d37f7812a089e4b7e30efc7524/2.CSS/CSS%209/index4.html)

Height and width

Color

Syntax:  
  
transition: propertyManipulating sec ;



**2. CSS Transition Timing Function:**

References: [**https://github.com/Lambodar2001/Web-Development-/blob/792bbc5c775f58db835d634368441bc6e504d9ff/2.CSS/CSS%209/index5.html**](https://github.com/Lambodar2001/Web-Development-/blob/792bbc5c775f58db835d634368441bc6e504d9ff/2.CSS/CSS%209/index5.html)

The transition-timing-function property in CSS controls the speed of an animation. It defines how the intermediate states of the transition are calculated.

**Values:**

1. **linear:** The animation has the same speed from start to finish.
2. **ease:** The default. The animation starts slow, then speeds up, and slows down at the end.
3. **ease-in:** The animation starts slowly and then speeds up.
4. **ease-out:** The animation starts quickly and then slows down.
5. **ease-in-out:** The animation starts slowly, speeds up in the middle, and slows down at the end.
6. **steps(n, direction):** The animation changes in steps. n is the number of steps, and direction can be start or end.
7. **cubic-bezier(x1, y1, x2, y2):** Allows you to define your own timing function. It uses cubic-bezier curves.

**ubic-bezier(x1, y1, x2, y2)**

**Definition:** The cubic-bezier(x1, y1, x2, y2) function in CSS allows you to create a custom timing function for transitions and animations. It defines the speed of the animation using a cubic Bezier curve.

**Parameters:**

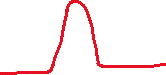
* **x1, y1, x2, y2:** These are four control points that define the shape of the cubic Bezier curve.
  + x1 and x2 must be in the range [0, 1] because they represent time (0% to 100% of the duration).
  + y1 and y2 can be any value, as they represent the progress of the animation.

**How It Works:**

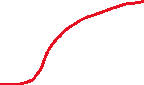
* The cubic Bezier curve is a mathematical formula used to create smooth curves.
* The curve starts at (0,0) and ends at (1,1).
* The points (x1, y1) and (x2, y2) control the shape of the curve.



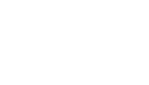
1. **Ease :** **The default. The animation starts slow, then speeds up, and slows down at the end.**



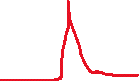
1. **Ease-in ease-in:** The animation starts slowly and then speeds up.



1. **ease-out:** The animation starts quickly and then slows down.



1. **ease-in-out:** The animation starts slowly, speeds up in the middle, and slows down at the end.



**cubic-bezier(x1, y1, x2, y2)**

**Definition:** The cubic-bezier(x1, y1, x2, y2) function in CSS allows you to create a custom timing function for transitions and animations. It defines the speed of the animation using a cubic Bezier curve.

**Parameters:**

* **x1, y1, x2, y2:** These are four control points that define the shape of the cubic Bezier curve.
  + x1 and x2 must be in the range [0, 1] because they represent time (0% to 100% of the duration).
  + y1 and y2 can be any value, as they represent the progress of the animation.

**How It Works:**

* The cubic Bezier curve is a mathematical formula used to create smooth curves.
* The curve starts at (0,0) and ends at (1,1).
* The points (x1, y1) and (x2, y2) control the shape of the curve.

**Example:**

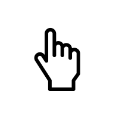
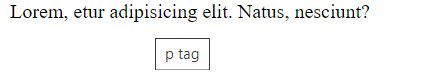
Css

***transition-timing-function: cubic-bezier(0.25, 0.1, 0.25, 1);***

**tooltip:**

A tooltip in CSS is a small pop-up box that appears when you hover over an element. It typically provides additional information about the element or context. You can create tooltips using CSS by *title*



****

1. **Animamtion** 
   * 1. Introduction:

 Animation: Making an element change its style smoothly over time.

 @**keyframes Rules**: Define what an animation looks like at each stage (0% to 100%).

 Delaying **an Animation**: Start the animation after a specified time.

 Setting **how many times an animation can run**: Define how many times the animation should repeat.

 Run **Animation in Reverse Direction or Alternate Cycles**: Make the animation play backward or switch directions each cycle.

 Specify **the speed Curve of the Animation**: Control the pace of the animation (e.g., fast at start, slow at end).

 Specify **the fill-mode For an Animation**: Define how styles are applied before and after the animation runs.

 Animation **Shorthand Property**: Combine multiple animation properties into one line for simplicity.

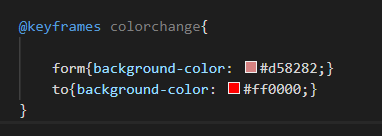
**What is animation**

● An animation lets an element gradually change from one style to another.

● You can change as many CSS properties you want, as many times as you want.

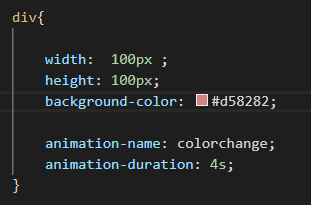
● To use CSS animation, you must first specify some keyframes for the animation.

* + 1. How to define and use animation

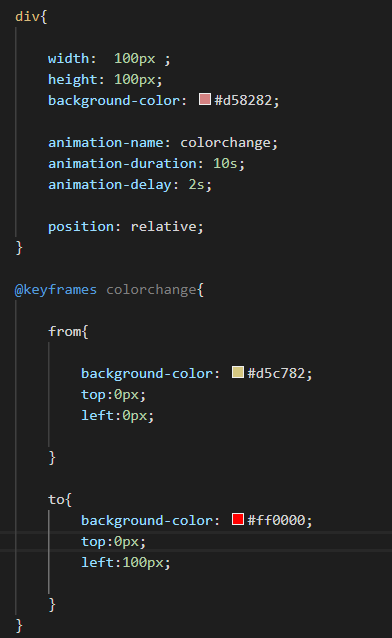
**Step 1**: Define keyframe function with proper name using keyframe key word

**Step 2**: call function where element you want using *animation-name: colorchange*;

**Step 3:**  give animation duration using animation*-duration: 4sec*



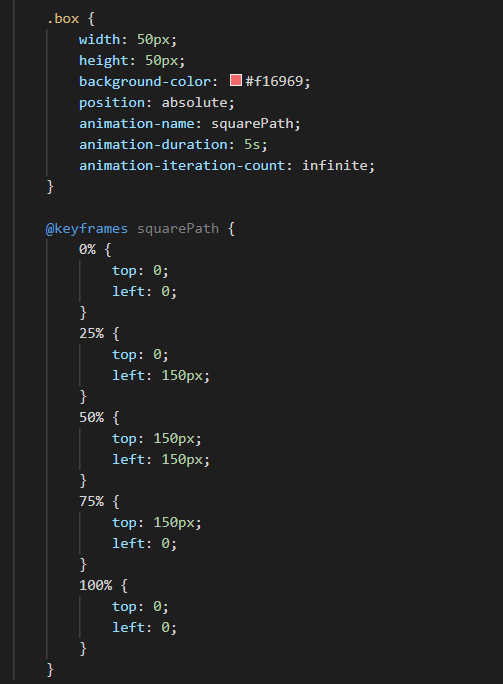
***Project****: combination of all above properties: POS AND COLOR CHANGE ANIMATION*

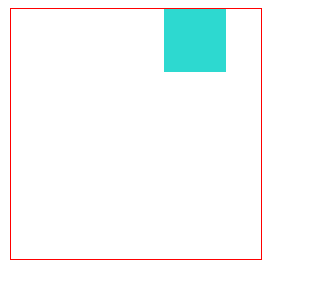


* + 1. Keyframes Using percentage value

**Keyframes:**

* **0%**: Start at the top-left corner (0, 0).
* **25%**: Move to the top-right corner (0, 150px).
* **50%**: Move to the bottom-right corner (150px, 150px).
* **75%**: Move to the bottom-left corner (150px, 0).
* **100%**: Return to the top-left corner (0, 0).





* + 1. Keyframes Using percentage value

1. give animation Delay to wait after load page using: animation*-delay: 4sec*
2. Repeat animation using: animation-iteration-count: infinite/3/number;
3. Animation motion using transition time function value: *animation-timing-function: linear*;

**Specify the Speed Curve of the Animation**

The animation-timing-function property specifies the speed curve of the animation. This property can have the following values:

1. **ease**: Specifies an animation with a slow start, then fast, then slow end (this is the default).
2. **linear**: Specifies an animation with the same speed from start to end.
3. **ease-in**: Specifies an animation with a slow start.
4. **ease-out**: Specifies an animation with a slow end.
5. **ease-in-out**: Specifies an animation with a slow start and end.
6. **cubic-bezier(n,n,n,n)**: Lets you define your own values in a cubic-bezier function.
7. Animation direction:
   * + 1. **normal**: The animation plays forward, from start to end.
       2. **reverse**: The animation plays backward, from end to start.
       3. **alternate**: The animation plays forward first, then backward, and keeps repeating this cycle.
       4. **alternate-reverse**: The animation plays backward first, then forward, and keeps repeating this cycle.

1. Specify the Fill Mode for an Animation:

In CSS animations, elements usually don't have any styles before the animation starts or after it ends. The animation-fill-mode property changes this behaviour.

1. **none**: This is the default. It means the animation doesn't change any styles before or after it plays.
2. **forwards**: After the animation ends, the element keeps the styles from the last keyframe.
3. **backwards**: Before the animation starts, the element gets the styles from the first keyframe and keeps them until the animation begins.
4. **both**: The animation applies styles both before it starts (like backwards) and after it ends (like forwards).

Module 21

**Mobile Responsive and positions**

* + 1. Lecture 01: Media Queries

**Mobile Responsive**

**Mobile Responsive** means designing websites that look good on all devices (desktops, tablets, smartphones). The layout, images, and other elements adjust automatically to fit the screen size.

**Media Queries**

**Media Queries** are CSS rules that apply styles based on the device's characteristics, like screen size. They help make a website responsive.

**Example of Media Queries**

/\* Styles for small screens \*/

@media (max-width: 600px) {

body {

background-color: lightblue;

}

}

/\* Styles for medium screens \*/

@media (min-width: 601px) and (max-width: 1200px) {

body {

background-color: lightgreen;

}

}

/\* Styles for large screens \*/

@media (min-width: 1201px) {

body {

background-color: lightcoral;

}

}

This code changes the background color of the webpage based on the screen size:

* **Light blue** for screens up to 600px wide.
* **Light green** for screens between 601px and 1200px wide.
* **Light coral** for screens larger than 1201px wide.
  1. Ways of write MQ

View Port:  
A viewport is the visible area of a webpage on your screen. When you open a website on your computer, tablet, or phone, the viewport is the part of the screen where the webpage is displayed. It changes size depending on the device you're using.

In CSS, you can control how your webpage looks in different viewports using units like:

* vw (viewport width): 1vw is 1% of the viewport's width.
* vh (viewport height): 1vh is 1% of the viewport's height.
* vmin and vmax: Based on the smaller or larger value between vw and vh.

This helps make your website look good on all devices, from big desktop monitors to small phone screens.

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

How to write Media Queries:

**/\* Styles for screens wider than 400px \*/**

**@media (min-width: 400px) {**

**body {**

**background-color: lightblue;**

**}**

**}**

**/\* Styles for screens narrower than 767px \*/**

**@media (max-width: 767px) {**

**body {**

**background-color: lightgreen;**

**}**

**}**

**/\* Styles for screens between 400px and 767px \*/**

**@media (min-width: 400px) and (max-width: 767px) {**

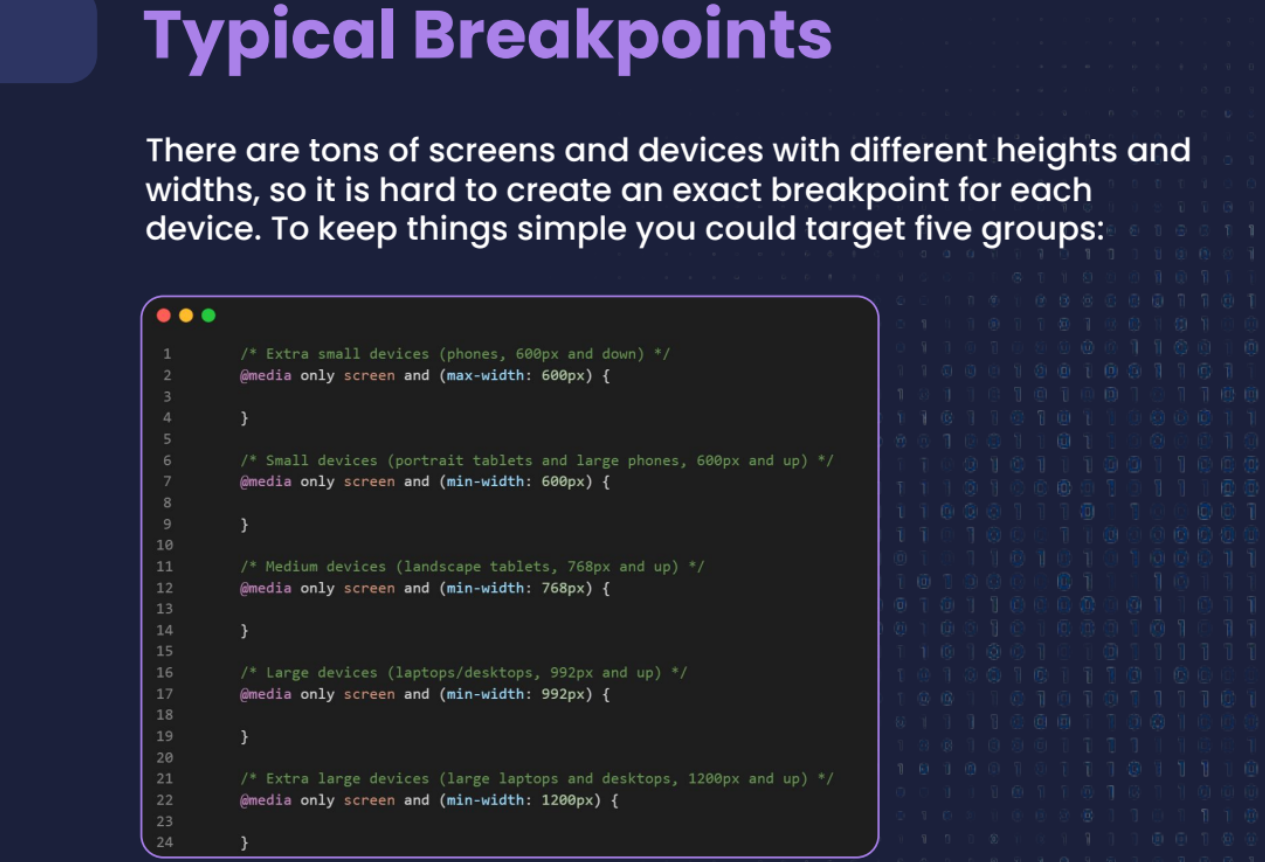
**body {**

**background-color: lightcoral;**

**}**

**}  
  
\* Only screen- see on next topic**

* @media (min-width: 400px) applies styles when the viewport width is 400px or wider.
* @media (max-width: 767px) applies styles when the viewport width is 767px or narrower.
* @media (min-width: 400px) and (max-width: 767px) applies styles when the viewport width is between 400px and 767px (inclusive).
  1. Standard media queries

****

**/\* Extra Small Devices (phones, less than 576px) \*/**

@media (max-width: 575.98px) {

body {

font-size: 14px;

}

}

**/\* Small Devices (tablets, 576px and up) \*/**

@media (min-width: 576px) {

body {

font-size: 16px;

}

}

**/\* Medium Devices (desktops, 768px and up) \*/**

@media (min-width: 768px) {

body {

font-size: 18px;

}

}

**/\* Large Devices (large desktops, 992px and up) \*/**

@media (min-width: 992px) {

body {

font-size: 20px;

}

}

**/\* Extra Large Devices (extra-large desktops, 1200px and up) \*/**

@media (min-width: 1200px) {

body {

font-size: 22px;

}

}

In this example:

* **Extra Small Devices**: Targets screens smaller than 576px (phones).
* **Small Devices**: Targets screens 576px and wider (small tablets).
* **Medium Devices**: Targets screens 768px and wider (tablets and smaller desktops).
* **Large Devices**: Targets screens 992px and wider (larger desktops).
* **Extra Large Devices**: Targets screens 1200px and wider (extra large desktops).

Adjust the font-size or any other styles according to your design requirements for each device size category. These breakpoints (576px, 768px, 992px, 1200px) are common in many responsive design frameworks like Bootstrap, but you can adjust them based on your specific design needs. Always test your styles across different devices to ensure they look and function as intended.

* “Only screen” Explanation:

@media only screen and (max-width: 600px) {

}   
In CSS media queries, "only screen" is a way to specify that the styles inside the media query should only apply when the query matches a screen device.

! "only screen" in a CSS @media query is a way to ensure that the styles you're defining inside that query only apply to devices with screens, like desktop computers, laptops, tablets, and smartphones.

Here’s a practical breakdown:

* **Screen Devices**: Devices with screens include things like your computer monitor, laptop screen, tablet display, and smartphone screen.
* **Not Print or Speech**: By using "only screen", you're telling the browser to ignore devices that are meant for printing (like printers) or for speech synthesis (like screen readers).
* **Why It's Used**: It's a way to make sure the styles you're applying are relevant and optimized for how things appear on a screen. This is important because screen-based styles often need to adjust layout, font sizes, or other aspects to fit different screen sizes and resolutions.

So, when you see "only screen" in a media query like @media only screen and (max-width: 600px), it means the styles inside { ... } will only affect how things look on screens, specifically when the screen width is 600 pixels or less. This helps in creating designs that are responsive and adapt well to various screen sizes.

* 1. Lecture 02: CSS Layout

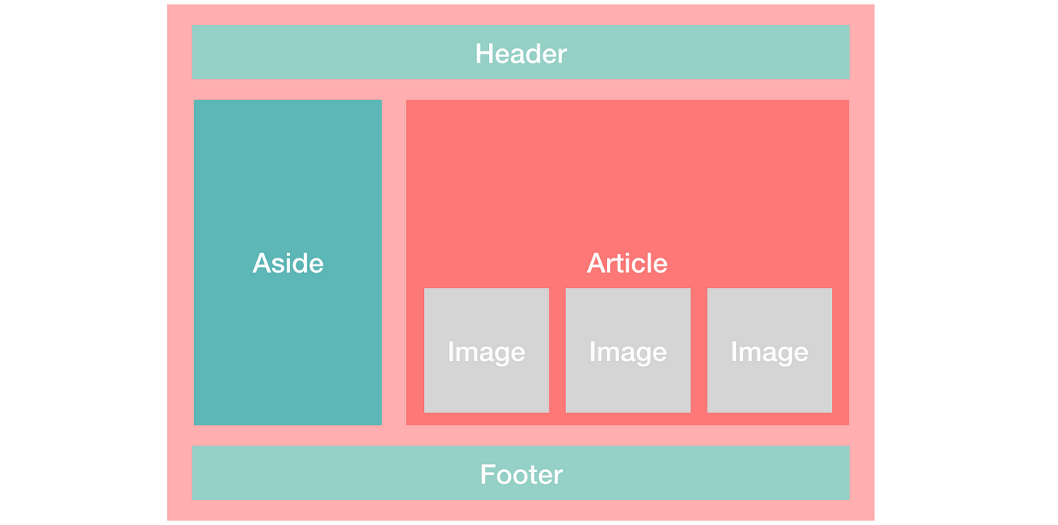
Topics to be covered

1. Introduction to CSS Layout

2. Benefits of CSS Layout

3. Types of CSS Layout

1. Normal Flow
2. Float
3. Position
4. Flex
5. Grid



It is a very important part of web design which allows the developer to position the elements

on a web page so that the web page can be visually appealing and user friendly.

In other words, we can say that it is a process of using CSS to control the position and size of

elements on a web page.

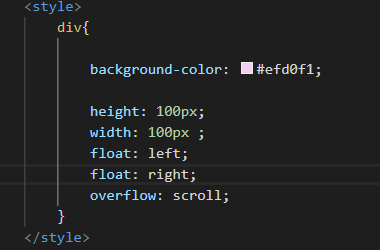
**1. CSS Layout: Normal Flow**

In normal flow, elements are displayed one after the other in the order they appear in the HTML. Block elements (like paragraphs) take up the full width and start on a new line, while inline elements (like links or span) sit side by side.

**2. CSS Layout Float**

The float property allows you to push an element to the left or right, allowing other content to wrap around it. It's often used for creating layouts with images and text side by side.

* **Left**
* **Right**
* **None**



* 1. Position, 4. flex and 5. grid will explain in Deep in next module

**3. CSS Layout : Position**

**1.1 Topics to be covered**

1. Introduction to CSS Position

2. Why to use CSS Position?

3. CSS Position properties

4. Top, Bottom, Left and Right properties

**1.2 Introduction to CSS Position**

CSS Position is a very powerful technique used to control the layout of elements

on a web page. We can easily specify the position of an element within the

document flow and can also control the behaviour of elements when they are

rendered on the screen. (हे developer ला freedom देतो कि element screen वर कुठे ही position करू शकतो )

**1.3 Why to use CSS Position?**

1. Control over element position

2. Positioning relative to other elements

3. Removing elements from document flow

4. Overlapping elements

5. Positioning relative to viewport

6. Create scroll effect

7. Accessibility

**1.4 CSS position property**

**There are majorly 5 position properties named:**

**1. Static**

**2. Relative**

**3. Absolute**

**4. Fixed**

**5. Sticky**

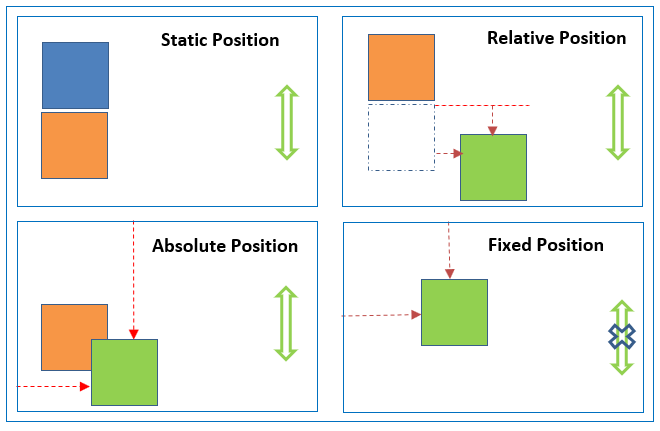
**And this are controlled by:**

**1.top**

**2.bottom**

**3.left**

**4. right**



१.static ज्याच्या वर top, right सारख्या properties चा फरक पडत नही

२. Fixed is that it will remain fixed like a chat option is an example of it

**1. 5** In CSS, the position property, combined with the properties top, bottom, left, and right, helps you control the placement of elements on a page. Here's how they relate to each other with different position values:

**Position Types and Related Properties**

margin-top affects both the element and the elements around it by adding space.

position: relative; top moves the element without affecting others, keeping its original space reserved.

1. **Static**:

* This is the default position for all elements. They follow the normal document flow, and the top, bottom, left, and right properties have no effect.

**. static-element {**

**position: static;**

**top: 10px; /\* No effect \*/**

**left: 20px; /\* No effect \*/**

**}**

1. Fixed:

* It stays fixed even when the page is scrolled. The top, bottom, left, and right properties determine its position.

**. fixed-element {**

**position: fixed;**

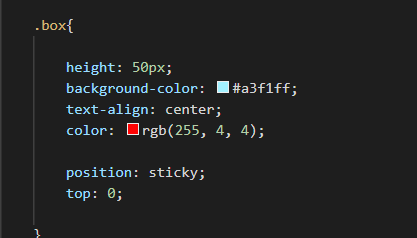
**bottom: 10px; /\* Sticks to the bottom of the viewport, 10px away \*/**

**right: 20px; /\* Sticks to the right of the viewport, 20px away \*/**

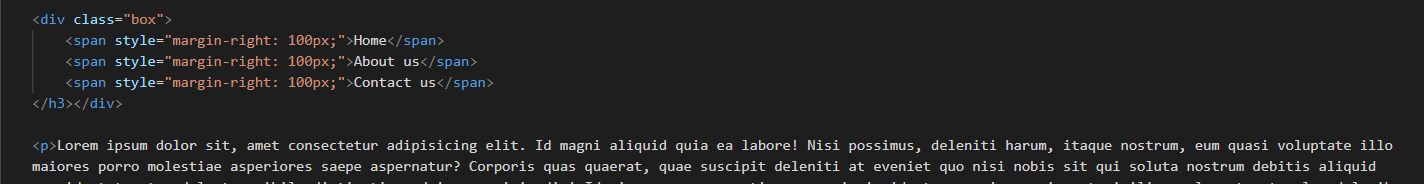
**}**  
  
ex: Chatbot option in webapp

1. Sticky:

Acts like relative until you scroll to a certain point, then sticks in place.

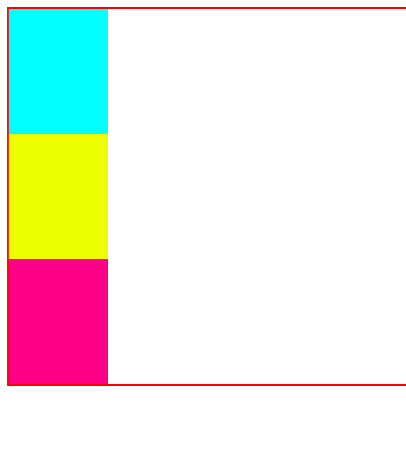
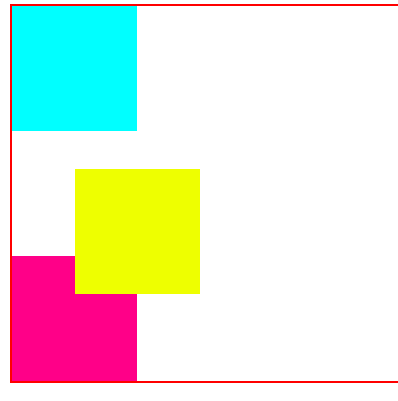
Ex. headers that stay at top:



**या मध्ये sticky चा Relative म्हणजेच खालचे parent मधले सर्व tag, content संपे पर्यंत तो stick राहतो top ला**

**या मध्ये जो पर्यंत p tag चा content संपत नाही तो पर्यंत box stick राहणार**

1. Relative
2. Absolute   
     
   1.Relative मध्ये element ज्या current Position ला आहे तेथून ते position , top, right etc ला change करते

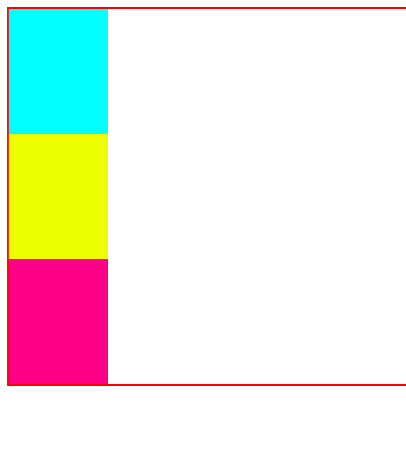
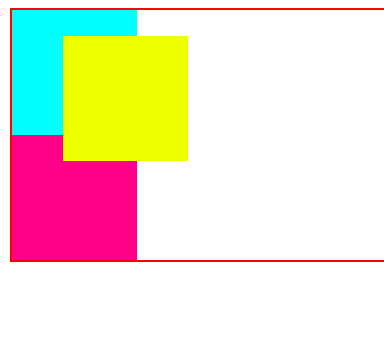


position: relative;

            left: 50px;

            top : 30px;

2.Absolute मध्ये element हा parents अनुसार त्याची position change करतो



position: Absolute;

            left: 50px;

            top : 30px;

**1.6. Z-Index:**

The z-index property in CSS controls the stacking order of elements that overlap on a webpage. It only works on positioned elements (those with position: relative, absolute, fixed, or sticky). The higher the z-index value, the closer the element is to the viewer, meaning it will appear on top of elements with lower z-index values.

**. box1 {**

**position: absolute;**

**z-index: 1; /\* Lower z-index, behind \*/**

**}**

**. box2 {**

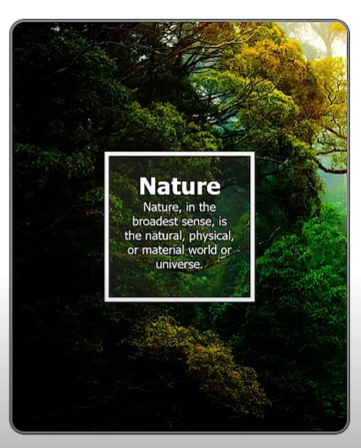
**position: absolute;**

**z-index: 2; /\* Higher z-index, in front \*/**

**}**

In this example, .box2 will appear on top of .box1 because it has a higher z-index value.

या मध्ये पहिला position define करावे लागते आणि नंतर z-index अनुसर overlap आपण ठरवू शकतो

  
ex:

Project: Transparent div

**4. CSS Layout: Flexbox**

Flexbox (Flexible Box Layout) is a CSS layout model designed to help create responsive and flexible layouts with ease. It allows you to arrange items in a container, control their size, spacing, and alignment, both horizontally and vertically (1D or row/Column).

Parent-Child/items Relation = flexbox

**Key Concepts:**

1. **Flex Container**: The parent element with

display: flex

display: inline-flex.

1. **Flex Items**: The direct children of the flex container.
2. **Main axis**
3. **Cross Axis**

**Main Properties:**

1. **justify-content**: Aligns items along the main axis (left to right in a row, top to bottom in a column).
   * Values: flex-start, flex-end, center, space-between, space-around, space-evenly
2. **align-items**: Aligns items along the cross-axis (perpendicular to the main axis).
   * Values: flex-start, flex-end, center, baseline, stretch
3. **flex-direction**: Sets the direction of the main axis (the flow of items).
   * Values: row, row-reverse, column, column-reverse
4. **flex-wrap**: Determines whether items should wrap onto multiple lines.
   * Values: nowrap, wrap, wrap-reverse
5. **align-content**: Aligns rows of items when there's extra space in the cross-axis.
   * Values: flex-start, flex-end, center, space-between, space-around, stretch

**Basic Usage Example:**

.container {

display: flex;

justify-content: center;

align-items: center;

flex-wrap: wrap;

}

* In this example, the items in the .container are centered both horizontally and vertically, and they will wrap to the next line if needed.
* Flexbox is great for creating responsive layouts and aligning elements dynamically without needing complex CSS rules or extra markup.

**Understand properties through code:**

**1. Display**

**1. Container:**

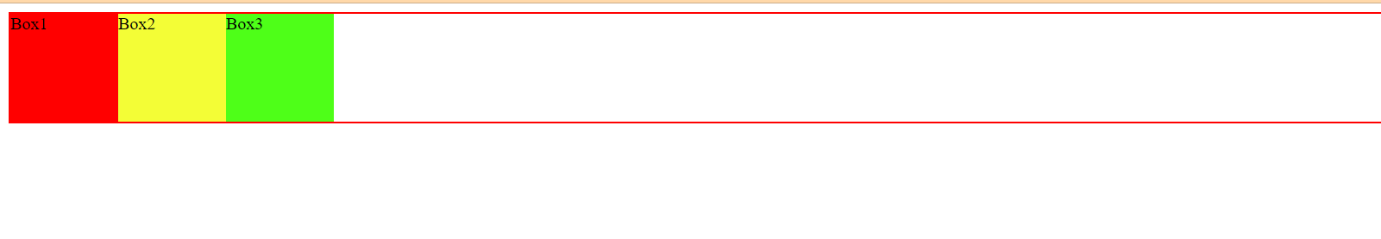
**a. Display: Flex;**

        .container {

            display: flex;

            border: 2px solid red ;

        }



**a. Display: inline-Flex;**

  .container {

            display: inline-flex;

            border: 2px solid red;

        }



display: inline block and

display: flex /inline-flex

या मध्ये असा difference आहे की display: inline block हे elements मध्ये लावतो जो को height, width allow krto

but

display: flex /inline-flex

हे flexbox चे property आहेत so हे container ला देतो असं flex box work as parent-iteam

**Understand properties through code:**

**2. Flex-Direction:**

The flex-direction property in CSS defines the direction in which the flex container's items are placed along the main axis. Here are the four main values you can use:

1. **row** (default):
   * Items are placed in a horizontal row, from left to right.
2. **row-reverse**:
   * Items are placed in a horizontal row, but in reverse order, from right to left.
3. **column**:
   * Items are placed in a vertical column, from top to bottom.
4. **column-reverse**:
   * Items are placed in a vertical column, but in reverse order, from bottom to top.

**Example:**

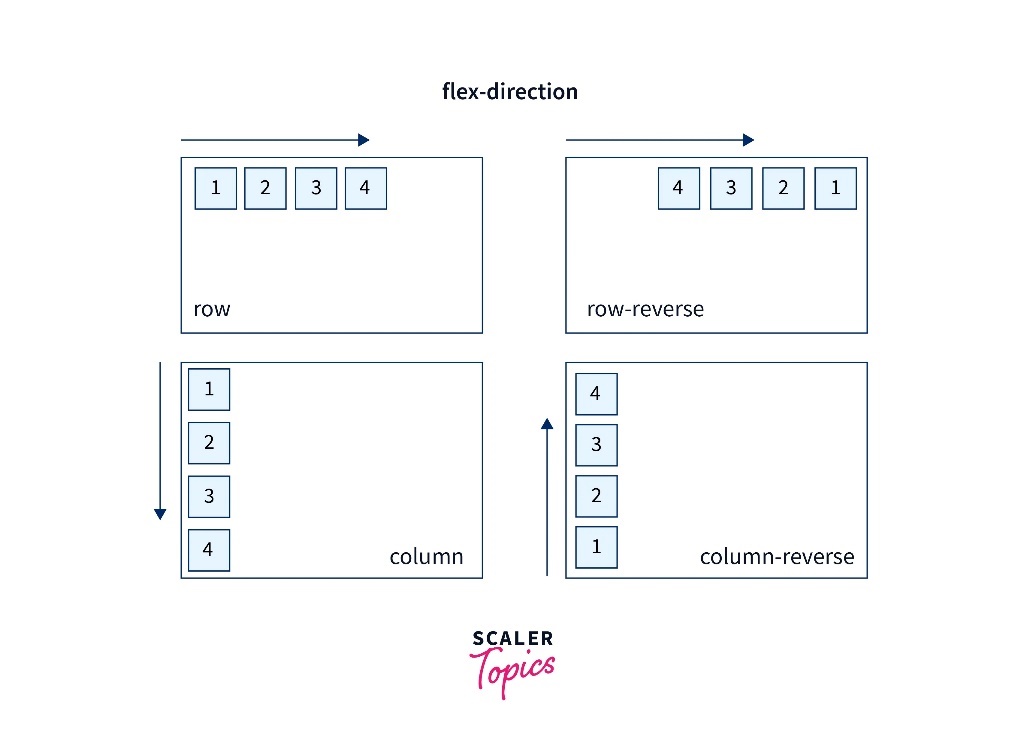
.container {

display: flex;

flex-direction: row-reverse;

}

In this example, the items inside the .container will be arranged in a row from right to left.

****

**Understand properties through code:**

**3. Flex-wrap:**

The flex-wrap property in CSS controls whether the flex container's items should wrap onto multiple lines or stay on a single line.

Here are the three main values:

1. **nowrap** (default):
   * Items stay on a single line, even if they overflow the container. They will not wrap.
2. **wrap**:
   * Items wrap onto multiple lines, from top to bottom. They will move to the next line when there's not enough space.
3. **wrap-reverse**:
   * Items wrap onto multiple lines, but from bottom to top, in reverse order.

**Example:**

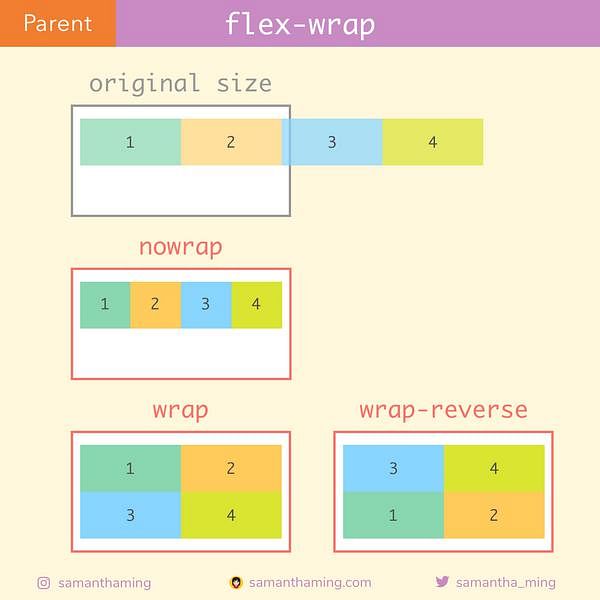
.container {

display: flex;

flex-wrap: wrap;

}

In this example, the items inside the .container will wrap to the next line if there isn't enough space on the first line.



**Understand properties through code:**

**3. justify-content:**

The justify-content property in CSS is used to align flex items along the main axis of a flex container (either horizontally in a row or vertically in a column).

Here are the main values:

1. **flex-start** (default):
   * Items are aligned at the start of the container.
2. **flex-end**:
   * Items are aligned at the end of the container.
3. **center**:
   * Items are centered along the main axis.
4. **space-between**:
   * Items are evenly distributed, with the first item at the start and the last item at the end of the container.
5. **space-around**:
   * Items are evenly distributed with equal space around them, meaning the spaces between the items and the edges of the container are equal.
6. **space-evenly**:
   * Items are evenly distributed with equal space between them and at the edges of the container.

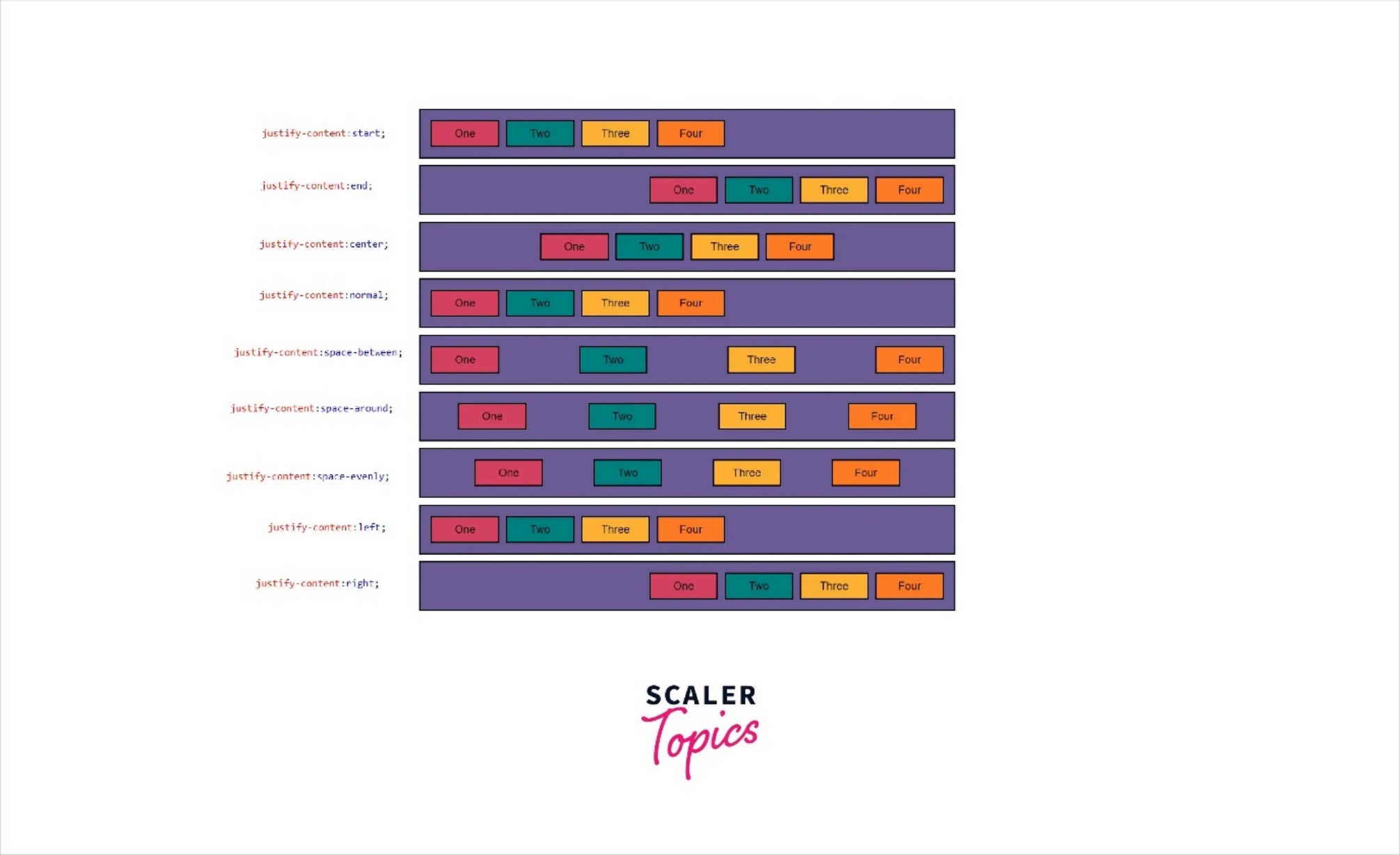
**Example:**

.container {

display: flex;

justify-content: space-between;

}

****In this example, the items inside the .container are evenly spaced, with the first item at the start and the last item at the end of the container, and equal space between all items.