




CSS

{ CASCADING STYLE SHEET }

Presented By:
SAGAR SHRESTHA



CSS Introduction

- CSS is the language we use to style a Web page.
- CSS stands for Cascading Style Sheets.
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media.
- CSS saves a lot of work. It can control the layout of multiple web pages all at once.
- External stylesheets are stored in CSS files.
- CSS is a style sheet language used for specifying the presentation and styling of a document written in a markup language such as HTML or XML.
- CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

Types of CSS

1. **Inline CSS:** Inline CSS is applied directly within the HTML element using the **style attribute**.

`<p style="color: red;">This is a paragraph with inline CSS.</p>`

2. **Internal CSS:** Internal CSS is defined within the **<style>** tag in the **<head>** section of an HTML document.

```
<head>
  <style>
    p {
      color: blue;
    }
  </style>
</head>
```

Types of CSS

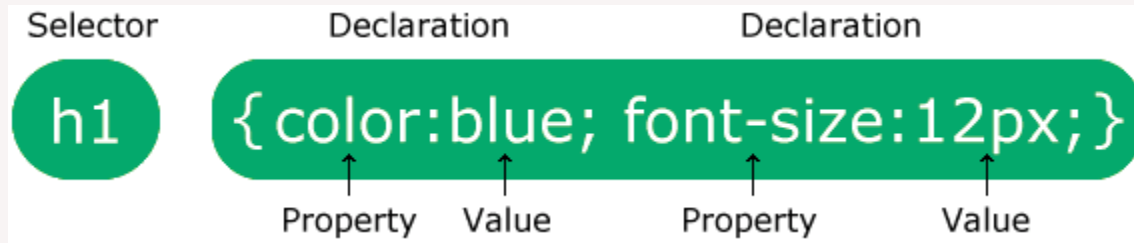
3. **External CSS:** External CSS is defined in a separate .css file, and the file is linked to the HTML document using the `<link>` tag in the `<head>` section.

```
<head>  
  <link rel="stylesheet" type="text/css" href="styles.css">  
</head>
```

```
p {  
  color: green;  
}
```

CSS Syntax

- A CSS rule consists of a selector and a declaration block.



- The selector points to the HTML element you want to style.
- The declaration block contains one or more declarations separated by semicolons.
- Each declaration includes a CSS property name and a value, separated by a colon.
- Multiple CSS declarations are separated with semicolons, and declaration blocks are surrounded by curly braces.

CSS Selectors

- A CSS selector selects the HTML element(s) you want to style.
- CSS selectors are used to "find" (or select) the HTML elements you want to style.
- We can divide CSS selectors into five categories:
 1. **Simple selectors** (select elements based on name, id, class)
 2. **Combinator selectors** (select elements based on a specific relationship between them)
 3. **Pseudo-class selectors** (select elements based on a certain state)
 4. **Pseudo-elements selectors** (select and style a part of an element)
 5. **Attribute selectors** (select elements based on an attribute or attribute value)

CSS Simple Selectors

1. **The CSS element Selector:** The element selector selects HTML elements based on the element name.

Example

Here, all `<p>` elements on the page will be center-aligned, with a red text color:

```
p {  
  text-align: center;  
  color: red;  
}
```

CSS Simple Selectors

2. The CSS id Selector:

- The id selector uses the id attribute of an HTML element to select a specific element.
- The id of an element is unique within a page, so the id selector is used to select one unique element!
- To select an element with a specific id, write a hash (#) character, followed by the id of the element.

Example

The CSS rule below will be applied to the HTML element with id="para1":

```
#para1 {  
  text-align: center;  
  color: red;  
}
```


CSS Simple Selectors

3. The CSS class Selector:

- You can also specify that only specific HTML elements should be affected by a class.

Example

Select and style all elements with class="intro":

```
.intro {  
  background-color: yellow;  
}
```

CSS Simple Selectors

4. The CSS Universal Selector:

- The universal selector (*) selects all HTML elements on the page.

Example

The CSS rule below will affect every HTML element on the page:

```
* {  
  text-align: center;  
  color: blue;  
}
```

CSS Simple Selectors

Selector	Example	Example description
<u>#id</u>	#firstname	Selects the element with id="firstname"
<u>.class</u>	.intro	Selects all elements with class="intro"
<u>element.class</u>	p.intro	Selects only <p> elements with class="intro"
<u>*</u>	*	Selects all elements
<u>element</u>	p	Selects all <p> elements
<u>element,element,...</u>	div, p	Selects all <div> elements and all <p> elements

CSS Comments

- A CSS comments are not displayed in the browser, but they can help document your source code.
- Comments are used to explain the code, and may help when you edit the source code at a later date.
- A CSS comment is placed inside the <style> element, and starts with /* and ends with */
- Examples:

```
/* This is a single-line comment */  
p {  
  color: red;  
}
```

```
p {  
  color: red; /* Set text color to red */  
}
```

```
p {  
  color: /*red*/blue;  
}
```

```
/* This is  
a multi-line  
comment */  
  
p {  
  color: red;  
}
```

CSS Colors

- CSS colors allow web developers to specify colors for HTML elements using various formats including named colors, hexadecimal, RGB, RGBA, HSL, and HSLA values, offering flexibility in design and accessibility.
- In CSS, a color can be specified by using a predefined color name:

Tomato	Orange	DodgerBlue	MediumSeaGreen
Gray	SlateBlue	Violet	LightGray

CSS RGB Colors

- An RGB color value represents RED, GREEN, and BLUE light sources.
- In CSS, a color can be specified as an RGB value, using this formula:

rgb(red, green, blue)

- Each parameter (red, green, and blue) defines the intensity of the color between 0 and 255.
- For example, rgb(255, 0, 0) is displayed as red, because red is set to its highest value (255) and the others are set to 0.
- To display black, set all color parameters to 0, like this: rgb(0, 0, 0).
- To display white, set all color parameters to 255, like this: rgb(255, 255, 255).

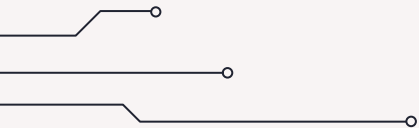
rgb(255, 0, 0)

rgb(0, 0, 255)

CSS RGB Colors

RGBA Colors:

- Similar to RGB but includes an additional value for alpha (transparency) channel, which ranges from 0 (fully transparent) to 1 (fully opaque).
- Example: **rgba(255, 0, 0, 0.5)** for semi-transparent red (50% opacity).



CSS HEX Colors

- A hexadecimal color is specified with: #RRGGBB, where the RR (red), GG (green) and BB (blue) hexadecimal integers specify the components of the color.
- In CSS, a color can be specified using a hexadecimal value in the form:

#rrggbb

- Where rr (red), gg (green) and bb (blue) are hexadecimal values between 00 and ff (same as decimal 0-255).
- For example, #ff0000 is displayed as red, because red is set to its highest value (ff) and the others are set to the lowest value (00).
- To display black, set all values to 00, like this: #000000.
- To display white, set all values to ff, like this: #ffffff.

A solid red rectangular color swatch.

#ff0000

A solid blue rectangular color swatch.

#0000ff

CSS HSL Colors

- HSL stands for hue, saturation, and lightness.
- In CSS, a color can be specified using hue, saturation, and lightness (HSL) in the form:

hsl(hue, saturation, lightness)

- Hue is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue.
- Saturation is a percentage value. 0% means a shade of gray, and 100% is the full color.
- Lightness is also a percentage. 0% is black, 50% is neither light or dark, 100% is white

`hsl(0, 100%, 50%)`

`hsl(240, 100%, 50%)`

CSS Backgrounds

- The CSS background properties are used to add background effects for elements.
- CSS background properties:
 - background-color
 - background-image
 - background-repeat
 - background-position
 - background (shorthand property)

CSS Backgrounds

background-color:

- The background-color property specifies the background color of an element.

Example

The background color of a page is set like this:

```
body {  
  background-color: lightblue;  
}
```

Here, the <h1>, <p>, and <div> elements will have different background colors:

```
h1 {  
  background-color: green;  
}  
  
div {  
  background-color: lightblue;  
}  
  
p {  
  background-color: yellow;  
}
```

CSS Backgrounds

Background Opacity / Transparency:

- The opacity property specifies the opacity/transparency of an element. It can take a value from 0.0 - 1.0. The lower value, the more transparent:



Example

```
div {  
  background-color: green;  
  opacity: 0.3;  
}
```

CSS Backgrounds

background-image:

- The background-image property specifies an image to use as the background of an element.
- By default, the image is repeated so it covers the entire element.
- When using a background image, use an image that does not disturb the text.

Example

Set the background image for a page:

```
body {  
  background-image: url("paper.gif");  
}
```

CSS Backgrounds

CSS background – shorthand property:

- To shorten the code, it is also possible to specify all the background properties in one single property. This is called a shorthand property.

```
body {  
  background-color: #ffffff;  
  background-image: url("img_tree.png");  
  background-repeat: no-repeat;  
  background-position: right top;  
}
```

```
body {  
  background: #ffffff url("img_tree.png") no-repeat right top;  
}
```

CSS Borders

- The CSS border properties allow you to specify the style, width, and color of an element's border.

I have borders on all sides.

I have a red bottom border.

I have rounded borders.

I have a blue left border.

CSS Borders

CSS Border Styles:

A dotted border.

A dashed border.

A solid border.

A double border.

A groove border. The effect depends on the border-color value.

A ridge border. The effect depends on the border-color value.

An inset border. The effect depends on the border-color value.

An outset border. The effect depends on the border-color value.

No border.

A hidden border.

A mixed border.

```
p.dotted {border-style: dotted;}
```

```
p.dashed {border-style: dashed;}
```

```
p.solid {border-style: solid;}
```

```
p.double {border-style: double;}
```

```
p.groove {border-style: groove;}
```

```
p.ridge {border-style: ridge;}
```

```
p.inset {border-style: inset;}
```

```
p.outset {border-style: outset;}
```

```
p.none {border-style: none;}
```

```
p.hidden {border-style: hidden;}
```

```
p.mix {border-style: dotted dashed solid double;}
```


CSS Borders

CSS Border Width:

5px border-width

medium border-width

2px border-width

thick border-width

```
p.one {  
  border-style: solid;  
  border-width: 5px;  
}  
  
p.two {  
  border-style: solid;  
  border-width: medium;  
}  
  
p.three {  
  border-style: dotted;  
  border-width: 2px;  
}  
  
p.four {  
  border-style: dotted;  
  border-width: thick;  
}
```

CSS Borders

CSS Border Sides:

Different Border Styles

```
p {  
  border-top-style: dotted;  
  border-right-style: solid;  
  border-bottom-style: dotted;  
  border-left-style: solid;  
}
```

CSS Borders

CSS Border Shorthand:

Some text

```
p {  
  border: 5px solid red;  
}
```

CSS Borders

CSS Rounded Border:

Normal border

Round border

Rounder border

Roudest border

```
normal {  
  border: 2px solid red;  
  padding: 5px;  
}  
  
round1 {  
  border: 2px solid red;  
  border-radius: 5px;  
  padding: 5px;  
}  
  
round2 {  
  border: 2px solid red;  
  border-radius: 8px;  
  padding: 5px;  
}  
  
round3 {  
  border: 2px solid red;  
  border-radius: 12px;  
  padding: 5px;  
}
```

CSS Borders

CSS Border Color:

Red border

Green border

Blue border

```
p.one {  
  border-style: solid;  
  border-color: red;  
}  
  
p.two {  
  border-style: solid;  
  border-color: green;  
}  
  
p.three {  
  border-style: dotted;  
  border-color: blue;  
}
```

CSS Margins

- Margins are used to create space around elements, outside of any defined borders.
- CSS has properties for specifying the margin for each side of an element:

- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`

- All the margin properties can have the following values:

- `auto` - the browser calculates the margin
- `length` - specifies a margin in px, pt, cm, etc.
- `%` - specifies a margin in % of the width of the containing element
- `inherit` - specifies that the margin should be inherited from the parent element

CSS Margins

Example

Set different margins for all four sides of a <p> element:

```
p {  
  margin-top: 100px;  
  margin-bottom: 100px;  
  margin-right: 150px;  
  margin-left: 80px;  
}
```

CSS Margins

Margin - Shorthand Property:

- To shorten the code, it is possible to specify all the margin properties in one property.
- If the **margin** property has four values:

- **margin: 25px 50px 75px 100px;**

- top margin is 25px
- right margin is 50px
- bottom margin is 75px
- left margin is 100px

```
p {  
    margin: 25px 50px 75px 100px;  
}
```

- If the **margin** property has three values:

- **margin: 25px 50px 75px;**

- top margin is 25px
- right and left margins are 50px
- bottom margin is 75px

```
p {  
    margin: 25px 50px 75px;  
}
```


CSS Margins

Margin - Shorthand Property:

- If the **margin** property has two values:

- **margin: 25px 50px;**
 - top and bottom margins are 25px
 - right and left margins are 50px

```
p {  
  margin: 25px 50px;  
}
```

- If the **margin** property has one value:

- **margin: 25px;**
 - all four margins are 25px

```
p {  
  margin: 25px;  
}
```

CSS Margins

The auto Value:

- You can set the margin property to auto to horizontally center the element within its container.
- The element will then take up the specified width, and the remaining space will be split equally between the left and right margins.

Example

Use margin: auto:

```
div {  
  width: 300px;  
  margin: auto;  
  border: 1px solid red;  
}
```

CSS Margins

The inherit Value:

- This example lets the left margin of the `<p class="ex1">` element be inherited from the parent element (`<div>`):

Example

Use of the inherit value:

```
div {  
  border: 1px solid red;  
  margin-left: 100px;  
}  
  
p.ex1 {  
  margin-left: inherit;  
}
```

CSS Margins

Margin Collapse:

- Top and bottom margins of elements are sometimes collapsed into a single margin that is equal to the largest of the two margins.
- This does not happen on left and right margins! Only top and bottom margins!

```
h1 {  
  margin: 0 0 50px 0;  
}  
  
h2 {  
  margin: 20px 0 0 0;  
}
```

- In the example above, the <h1> element has a bottom margin of 50px and the <h2> element has a top margin set to 20px.
- Common sense would seem to suggest that the vertical margin between the <h1> and the <h2> would be a total of 70px. But due to margin collapse, the actual margin ends up being 50px.

CSS Padding

- The CSS padding properties are used to generate space around an element's content, inside of any defined borders.
- With CSS, you have full control over the padding. There are properties for setting the padding for each side of an element (top, right, bottom, and left).
- CSS has properties for specifying the padding for each side of an element:
 - `padding-top`
 - `padding-right`
 - `padding-bottom`
 - `padding-left`
- All the padding properties can have the following values:
 - *length* - specifies a padding in px, pt, cm, etc.
 - *%* - specifies a padding in % of the width of the containing element
 - *inherit* - specifies that the padding should be inherited from the parent element

CSS Padding

➤ Example:

```
div {  
  border: 1px solid black;  
  background-color: lightblue;  
  padding-top: 50px;  
  padding-right: 30px;  
  padding-bottom: 50px;  
  padding-left: 80px;  
}
```

This div element has a top padding of 50px, a right padding of 30px, a bottom padding of 50px, and a left padding of 80px.

CSS Padding

Padding - Shorthand Property:

➤ If the **padding** property has four values:

- **padding: 25px 50px 75px 100px;**

- top padding is 25px
- right padding is 50px
- bottom padding is 75px
- left padding is 100px

```
div {  
  padding: 25px 50px 75px 100px;  
}
```

➤ If the **padding** property has three values:

- **padding: 25px 50px 75px;**

- top padding is 25px
- right and left paddings are 50px
- bottom padding is 75px

```
div {  
  padding: 25px 50px 75px;  
}
```

CSS Padding

Padding - Shorthand Property:

➤ If the **padding** property has two values:

- **padding: 25px 50px;**
 - top and bottom paddings are 25px
 - right and left paddings are 50px

```
div {  
  padding: 25px 50px;  
}
```

➤ If the **padding** property has one value:

- **padding: 25px;**
 - all four paddings are 25px

```
div {  
  padding: 25px;  
}
```


CSS Padding

Padding and Element Width:

- The CSS width property specifies the width of the element's content area. The content area is the portion inside the padding, border, and margin of an element.
- So, if an element has a specified width, the padding added to that element will be added to the total width of the element. This is often an undesirable result.

```
div.ex1 {  
  width: 300px;  
  background-color: yellow;  
}  
  
div.ex2 {  
  width: 300px;  
  padding: 25px;  
  background-color: lightblue;  
}
```

This div is 300px wide.

The width of this div is 350px, even though it is defined as 300px in the CSS.

CSS Padding

Padding and Element Width:

- To keep the width at 300px, no matter the amount of padding, you can use the **box-sizing** property.
- This causes the element to maintain its actual width; if you increase the padding, the available content space will decrease.

```
div.ex1 {  
  width: 300px;  
  background-color: yellow;  
}  
  
div.ex2 {  
  width: 300px;  
  padding: 25px;  
  background-color: lightblue;  
  box-sizing: border-box  
}
```

This div is 300px wide.

The width of this div is 350px, even though it is defined as 300px in the CSS.

CSS Height, Width and Max-width

- The CSS **height** and **width** properties are used to set the height and width of an element.
- The CSS **max-width** property is used to set the maximum width of an element.

```
div {  
  height: 50px;  
  width: 100%;  
  border: 1px solid #4CAF50;  
}
```

This div element has a height of 50 pixels and a width of 100%.

CSS Height, Width and Max-width

Setting max-width:

- The **max-width** property is used to set the maximum width of an element.
- The **max-width** can be specified in length values, like px, cm, etc., or in percent (%) of the containing block, or set to none (this is default. Means that there is no maximum width).
- The problem with the **<div>** above occurs when the browser window is smaller than the width of the element (500px). The browser then adds a horizontal scrollbar to the page.
- Using **max-width** instead, in this situation, will improve the browser's handling of small windows.

```
div {  
  max-width: 500px;  
  height: 100px;  
  background-color: powderblue;  
}
```

This element has a height of 100 pixels and a max-width of 500 pixels.

CSS Height, Width and Max-width

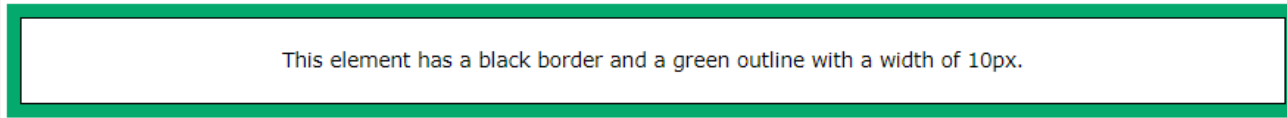
Note: If you for some reason use both the **width** property and the **max-width** property on the same element, and the value of the **width** property is larger than the **max-width** property; the **max-width** property will be used (and the **width** property will be ignored).

All CSS Dimension Properties

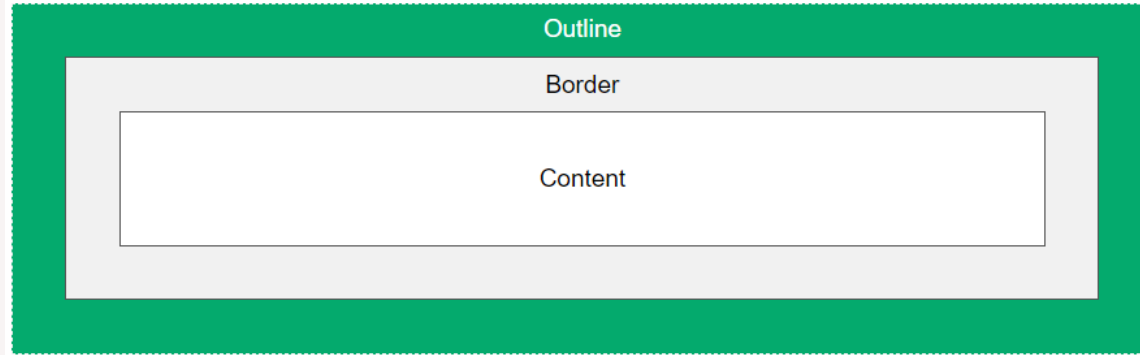
Property	Description
<u>height</u>	Sets the height of an element
<u>max-height</u>	Sets the maximum height of an element
<u>max-width</u>	Sets the maximum width of an element
<u>min-height</u>	Sets the minimum height of an element
<u>min-width</u>	Sets the minimum width of an element
<u>width</u>	Sets the width of an element

CSS Outline

- An outline is a line drawn outside the element's border.



- An outline is a line that is drawn around elements, OUTSIDE the borders, to make the element "stand out".

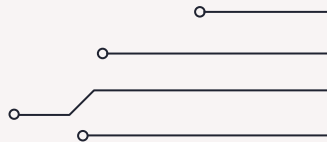


CSS Outline

- Outline differs from borders! Unlike border, the outline is drawn outside the element's border, and may overlap other content.
- Also, the outline is NOT a part of the element's dimensions; the element's total width and height is not affected by the width of the outline.

CSS has the following outline properties:

- `outline-style`
- `outline-color`
- `outline-width`
- `outline-offset`
- `outline`



CSS Outline

CSS Outline Style:

The `outline-style` property specifies the style of the outline, and can have one of the following values:

- `dotted` - Defines a dotted outline
- `dashed` - Defines a dashed outline
- `solid` - Defines a solid outline
- `double` - Defines a double outline
- `groove` - Defines a 3D grooved outline
- `ridge` - Defines a 3D ridged outline
- `inset` - Defines a 3D inset outline
- `outset` - Defines a 3D outset outline
- `none` - Defines no outline
- `hidden` - Defines a hidden outline

CSS Outline

Example:

```
p.dotted {outline-style: dotted;}  
p.dashed {outline-style: dashed;}  
p.solid {outline-style: solid;}  
p.double {outline-style: double;}  
p.groove {outline-style: groove;}  
p.ridge {outline-style: ridge;}  
p.inset {outline-style: inset;}  
p.outset {outline-style: outset;}
```

A dotted outline

A dashed outline

A solid outline

A double outline

A groove outline. The effect depends on the outline-color value.

A ridge outline. The effect depends on the outline-color value.

An inset outline. The effect depends on the outline-color value.

An outset outline. The effect depends on the outline-color value.

CSS Outline

CSS Outline Width:

- The outline-width property specifies the width of the outline, and can have one of the following values:

- thin (typically 1px)
- medium (typically 3px)
- thick (typically 5px)
- A specific size (in px, pt, cm, em, etc)

A thin outline.

A medium outline.

A thick outline.

A 4px thick outline.

CSS Outline

CSS Outline Width Example:

```
p.ex1 {  
  border: 1px solid black;  
  outline-style: solid;  
  outline-color: red;  
  outline-width: thin;  
}  
p.ex2 {  
  border: 1px solid black;  
  outline-style: solid;  
  outline-color: red;  
  outline-width: medium;  
}  
p.ex3 {  
  border: 1px solid black;  
  outline-style: solid;  
  outline-color: red;  
  outline-width: thick;  
}  
p.ex4 {  
  border: 1px solid black;  
  outline-style: solid;  
  outline-color: red;  
  outline-width: 4px;  
}
```

A thin outline.

A medium outline.

A thick outline.

A 4px thick outline.

CSS Outline

CSS Outline – Shorthand property:

- The **outline** property is a shorthand property for setting the following individual outline properties:

- `outline-width`
- `outline-style` (required)
- `outline-color`

Example

```
p.ex1 {outline: dashed;}  
p.ex2 {outline: dotted red;}  
p.ex3 {outline: 5px solid yellow;}  
p.ex4 {outline: thick ridge pink;}
```

A dashed outline.

A dotted red outline.

A 5px solid yellow outline.

A thick ridge pink outline.

CSS Outline

CSS Outline Offset:

- The **outline-offset** property adds space between an outline and the edge/border of an element. The space between an element and its outline is transparent.

Example

```
p {  
  margin: 30px;  
  border: 1px solid black;  
  outline: 1px solid red;  
  outline-offset: 15px;  
}
```

This paragraph has an outline 15px outside the border edge.

CSS Text Color

➤ The color property is used to set the color of the text. The color is specified by:

- a color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

```
h1 {  
  color: green;  
}
```

This is heading 1

Text Alignment

- The text-align property is used to set the horizontal alignment of a text.
- A text can be left or right aligned, centered, or justified.

```
h1 {  
  text-align: center;  
}  
  
h2 {  
  text-align: left;  
}  
  
h3 {  
  text-align: right;  
}
```

Heading 1 (center)

Heading 2 (left)

Heading 3 (right)

Text Alignment

- When the text-align property is set to "justify", each line is stretched so that every line has equal width, and the left and right margins are straight (like in magazines and newspapers):

```
div {  
  border: 1px solid black;  
  padding: 10px;  
  width: 200px;  
  height: 200px;  
  text-align: justify;  
}
```

In my younger and more vulnerable years my father gave me some advice that I've been turning over in my mind ever since. 'Whenever you feel like criticizing anyone,' he told me, 'just remember that all the people in this world haven't had the advantages that you've had.'

Vertical Alignment

- The vertical-align property sets the vertical alignment of an element.

```
span.a {  
  vertical-align: baseline;  
  font-size: 12px;  
}  
  
span.b {  
  vertical-align: text-top;  
  font-size: 12px;  
}  
  
span.c {  
  vertical-align: text-bottom;  
  font-size: 12px;  
}  
  
span.d {  
  vertical-align: sub;  
  font-size: 12px;  
}  
  
span.e {  
  vertical-align: super;  
  font-size: 12px;  
}
```

The vertical-align baseline

The vertical-align text-top

The vertical-align text-bottom

The vertical-align sub

The vertical-align super

Text Decoration

- The text-decoration-line property is used to add a decoration line to text.
- You can combine more than one value, like overline and underline to display lines both over and under a text.

- text-decoration-line
- text-decoration-color
- text-decoration-style
- text-decoration-thickness
- text-decoration

- **Add a Decoration Line to Text:** The **text-decoration-line** property is used to add a decoration line to text.

```
h1 {  
  text-decoration: overline;  
}  
h2 {  
  text-decoration: line-through;  
}  
h3 {  
  text-decoration: underline;  
}  
p.ex {  
  text-decoration: overline underline;  
}
```

Overline text decoration

~~Line-through text decoration~~

Underline text decoration

Overline and underline text decoration.

Text Decoration

- **Specify a Color for the Decoration Line:** The **text-decoration-color** property is used to set the color of the decoration line.

```
h1 {  
  text-decoration-line: overline;  
  text-decoration-color: red;  
}  
  
h2 {  
  text-decoration-line: line-through;  
  text-decoration-color: blue;  
}  
  
h3 {  
  text-decoration-line: underline;  
  text-decoration-color: green;  
}  
  
p {  
  text-decoration-line: overline underline;  
  text-decoration-color: purple;  
}
```

Overline text decoration

~~Line through text decoration~~

Underline text decoration

Overline and underline text decoration.

Text Decoration

- **Specify a Style for the Decoration Line:** The **text-decoration-style** property is used to set the style of the decoration line.

```
h1 {  
  text-decoration-line: underline;  
  text-decoration-style: solid; /* this is default */  
}  
  
h2 {  
  text-decoration-line: underline;  
  text-decoration-style: double;  
}  
  
h3 {  
  text-decoration-line: underline;  
  text-decoration-style: dotted;  
}  
  
p.ex1 {  
  text-decoration-line: underline;  
  text-decoration-style: dashed;  
}  
  
p.ex2 {  
  text-decoration-line: underline;  
  text-decoration-style: wavy;  
}  
  
p.ex3 {  
  text-decoration-line: underline;  
  text-decoration-color: red;  
  text-decoration-style: wavy;  
}
```

Heading 1

Heading 2

Heading 3

A paragraph.

Another paragraph.

Another paragraph.

Text Decoration

- **Specify the Thickness for the Decoration Line:** The **text-decoration-thickness** property is used to set the thickness of the decoration line.

```
h1 {  
  text-decoration-line: underline;  
  text-decoration-thickness: auto; /* this is default */  
}  
  
h2 {  
  text-decoration-line: underline;  
  text-decoration-thickness: 5px;  
}  
  
h3 {  
  text-decoration-line: underline;  
  text-decoration-thickness: 25%;  
}  
  
p {  
  text-decoration-line: underline;  
  text-decoration-color: red;  
  text-decoration-style: double;  
  text-decoration-thickness: 5px;  
}
```

Heading 1

Heading 2

Heading 3

A paragraph.

Text Decoration

➤ **The Shorthand Property:** The text-decoration property is a shorthand property for:

- `text-decoration-line` (required)
- `text-decoration-color` (optional)
- `text-decoration-style` (optional)
- `text-decoration-thickness` (optional)

```
h1 {  
  text-decoration: underline;  
}  
  
h2 {  
  text-decoration: underline red;  
}  
  
h3 {  
  text-decoration: underline red double;  
}  
  
p {  
  text-decoration: underline red double 5px;  
}
```

Heading 1

Heading 2

Heading 3

A paragraph.

Text Transformation

- The text-transform property is used to specify uppercase and lowercase letters in a text.
- It can be used to turn everything into uppercase or lowercase letters, or capitalize the first letter of each word.

```
p.uppercase {  
  text-transform: uppercase;  
}  
  
p.lowercase {  
  text-transform: lowercase;  
}  
  
p.capitalize {  
  text-transform: capitalize;  
}
```

THIS TEXT IS TRANSFORMED TO UPPERCASE.

this text is transformed to lowercase.

This Text Is Capitalized.

CSS Text Spacing

- The **text-indent** property is used to specify the indentation of the first line of a text:

```
p {  
  text-indent: 100px;  
}
```

In my younger and more vulnerable years my father gave me some advice that I've been turning over in my mind ever since.
'Whenever you feel like criticizing anyone,' he told me, 'just remember that all the people in this world haven't had the advantages that you've had.'

- The **letter-spacing** property is used to specify the space between the characters in a text:

```
h2 {  
  letter-spacing: 5px;  
}  
  
h3 {  
  letter-spacing: -2px;  
}
```

This is heading 1

This is heading 2

- The **line-height** property is used to specify the space between lines:

```
p.small {  
  line-height: 0.7;  
}  
  
p.big {  
  line-height: 1.8;  
}
```

This is a paragraph with a smaller line-height.
This is a paragraph with a smaller line-height.

This is a paragraph with a bigger line-height.
This is a paragraph with a bigger line-height.

CSS Text Spacing

- The **word-spacing** property is used to specify the space between the words in a text:

```
p.one {  
  word-spacing: 10px;  
}  
  
p.two {  
  word-spacing: -2px;  
}
```

This is a paragraph with larger word spacing.

This is a paragraph with smaller word spacing.

- The **white-space** property specifies how white-space inside an element is handled:

```
p {  
  white-space: nowrap;  
}
```

This is some text that will not wrap. This is some text that will not wrap. This is some text that will not wrap. This is some text that will not wrap. This is

Text Shadow

- The **text-shadow** property adds shadow to text.
- In its simplest use, you only specify the horizontal shadow (2px) and the vertical shadow (2px):

```
h1.a {  
  text-shadow: 2px 2px;  
}  
h1.b {  
  text-shadow: 2px 2px red;  
}  
h1.blur {  
  text-shadow: 2px 2px 5px red;  
}
```

Text-shadow effect!

Text-shadow effect with color

Text-shadow effect with blur and color

CSS Fonts

- Choosing the right font has a huge impact on how the readers experience a website.
- The right font can create a strong identity for your brand.
- Using a font that is easy to read is important. The font adds value to your text. It is also important to choose the correct color and text size for the font.

Difference Between Serif and Sans-serif Fonts



Sans-serif



Serif

CSS Font Families

- In CSS, we use the **font-family** property to specify the font of a text.

```
.p1 {  
  font-family: "Times New Roman", Times, serif;  
}  
  
.p2 {  
  font-family: Arial, Helvetica, sans-serif;  
}  
  
.p3 {  
  font-family: "Lucida Console", "Courier New", monospace;  
}
```

This is a paragraph, shown in the Times New Roman font.

This is a paragraph, shown in the Arial font.

This is a paragraph, shown in the Lucida Console font.

CSS Font Style

- The **font-style** property is mostly used to specify italic text.
- This property has three values:
 - normal - The text is shown normally
 - italic - The text is shown in italics
 - oblique - The text is "leaning" (oblique is very similar to italic, but less supported)

```
p.normal {  
  font-style: normal;  
}  
  
p.italic {  
  font-style: italic;  
}  
  
p.oblique {  
  font-style: oblique;  
}
```

This is a paragraph in normal style.

This is a paragraph in italic style.

This is a paragraph in oblique style.

Google Fonts

- If you do not want to use any of the standard fonts in HTML, you can use **Google Fonts**.
- **Google Fonts** are free to use, and have more than 1000 fonts to choose from.

```
<head>
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Sofia">
<style>
body {
  font-family: "Sofia", sans-serif;
}
</style>
</head>
```

Sofia Font

Lorem ipsum dolor sit amet.

123456790

Trirong Font

Lorem ipsum dolor sit amet.

123456790

Audiowide Font

Lorem ipsum dolor sit amet.

123456790

CSS Icons

- Icons can easily be added to your HTML page, by using an icon library.
- The simplest way to add an icon to your HTML page, is with an icon library, such as Font Awesome.
- Add the name of the specified icon class to any inline HTML element (like `<i>` or ``).



- Some popular icon libraries are:
 1. Font Awesome Icons
 2. Bootstrap Icons
 3. Google Icons

Font Awesome Icons

- To use the Font Awesome icons, go to fontawesome.com, sign in, and get a code to add in the **<head>** section of your HTML page:

`<script src="https://kit.fontawesome.com/yourcode.js" crossorigin="anonymous"></script>`

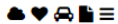
Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://kit.fontawesome.com/a076d05399.js" crossorigin="anonymous"></script>
</head>
<body>

<i class="fas fa-cloud"></i>
<i class="fas fa-heart"></i>
<i class="fas fa-car"></i>
<i class="fas fa-file"></i>
<i class="fas fa-bars"></i>

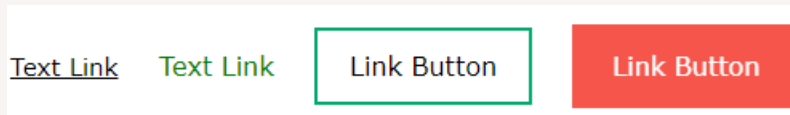
</body>
</html>
```

Result:



CSS Links

- With CSS, links can be styled in many different ways.



- Links can be styled differently depending on what state they are in.

- The four links states are:

- `a:link` - a normal, unvisited link
- `a:visited` - a link the user has visited
- `a:hover` - a link when the user mouses over it
- `a:active` - a link the moment it is clicked

CSS Links

```
/* unvisited link */
a:link {
  color: red;
}

/* visited link */
a:visited {
  color: green;
}

/* mouse over link */
a:hover {
  color: hotpink;
}

/* selected link */
a:active {
  color: blue;
}
```

This is a link

This is a link

Note: a:hover MUST come after a:link and a:visited in the CSS definition in order to be effective.

Note: a:active MUST come after a:hover in the CSS definition in order to be effective.

CSS List

- In HTML, there are two main types of lists unordered lists (****) and ordered lists (****) :
- **unordered lists ()** - the list items are marked with bullets
- **ordered lists ()** - the list items are marked with numbers or letters

Unordered Lists:

- Coffee
- Tea
- Coca Cola
- Coffee
- Tea
- Coca Cola

Ordered Lists:

1. Coffee
2. Tea
3. Coca Cola
- I. Coffee
- II. Tea
- III. Coca Cola

- The CSS list properties allow you to:
 - Set different list item markers for ordered lists
 - Set different list item markers for unordered lists
 - Set an image as the list item marker
 - Add background colors to lists and list items

CSS List

Different List Item Markers:

- The **list-style-type** property specifies the type of list item marker.

```
ul.a {  
  list-style-type: circle;  
}  
  
ul.b {  
  list-style-type: square;  
}  
  
ol.c {  
  list-style-type: upper-roman;  
}  
  
ol.d {  
  list-style-type: lower-alpha;  
}
```

Example of unordered lists:

- Coffee
- Tea
- Coca Cola

- Coffee
- Tea
- Coca Cola

Example of ordered lists:

- I. Coffee
- II. Tea
- III. Coca Cola

- a. Coffee
- b. Tea
- c. Coca Cola

CSS List

An Image as The List Item Marker:

- The **list-style-image** property specifies an image as the list item marker:

```
ul {  
  list-style-image: url('sqpurple.gif');  
}
```

- Coffee
- Tea
- Coca Cola

Position The List Item Markers:

- The **list-style-position** property specifies the position of the list-item markers (bullet points).

```
li{  
  border: 1px solid red;  
}  
ul.a {  
  list-style-position: outside;  
}  
ul.b {  
  list-style-position: inside;  
}
```

list-style-position: outside (default):

- Coffee - A brewed drink prepared from roasted coffee beans, which are the seeds of berries from the Coffea plant
- Tea - An aromatic beverage commonly prepared by pouring hot or boiling water over cured leaves of the Camellia sinensis, an evergreen shrub (bush) native to Asia
- Coca Cola - A carbonated soft drink produced by The Coca-Cola Company. The drink's name refers to two of its original ingredients, which were kola nuts (a source of caffeine) and coca leaves

list-style-position: inside:

- Coffee - A brewed drink prepared from roasted coffee beans, which are the seeds of berries from the Coffea plant
- Tea - An aromatic beverage commonly prepared by pouring hot or boiling water over cured leaves of the Camellia sinensis, an evergreen shrub (bush) native to Asia
- Coca Cola - A carbonated soft drink produced by The Coca-Cola Company. The drink's name refers to two of its original ingredients, which were kola nuts (a source of caffeine) and coca leaves

CSS List

Remove Default Settings:

- The **list-style-type:none** property can also be used to remove the markers/bullets.
- Note that the list also has default margin and padding. To remove this, add **margin:0** and **padding:0** to `` or ``:

```
ul.demo {  
  list-style-type: none;  
  margin: 0;  
  padding: 0;  
}
```

Default list:

- Coffee
- Tea
- Coca Cola

Remove bullets, margin and padding from list:

Coffee
Tea
Coca Cola

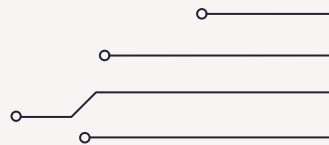
- The **list-style** property is a shorthand property. It is used to set all the list properties in one declaration:

```
ul {  
  list-style: square inside url("sqpurple.gif");  
}
```

- Coffee
- Tea
- Coca Cola

CSS Display

- The display property is the most important CSS property for controlling layout.
- The display property is used to specify how an element is shown on a web page.
- Every HTML element has a default display value, depending on what type of element it is.
- The default display value for most elements is block or inline.
- The display property is used to change the default display behavior of HTML elements.
- A block-level element ALWAYS starts on a new line and takes up the full width available (stretches out to the left and right as far as it can).
- An inline element DOES NOT start on a new line and only takes up as much width as necessary.
- **display: none;** is commonly used with JavaScript to hide and show elements without deleting and recreating them.
- Changing an inline element to a block element, or vice versa, can be useful for making the page look a specific way, and still follow the web standards.



CSS Display

```
p {  
  display: inline;  
  border: 1px solid red;  
}  
span{  
  display: block;  
  border: 1px solid red;  
}
```

This is a block element but made inline

This is inline element but made block

- Hiding an element can be done by setting the **display** property to **none**. The element will be hidden, and the page will be displayed as if the element is not there.
- **visibility:hidden**; also hides an element. However, the element will still take up the same space as before. The element will be hidden, but still affect the layout.

CSS Layout – The position Property

- The position property specifies the type of positioning method used for an element.
- There are five different position values:

- `static`
- `relative`
- `fixed`
- `absolute`
- `sticky`

- Elements are then positioned using the **top**, **bottom**, **left**, and **right** properties.
- However, these properties will not work unless the position property is set first.
- They also work differently depending on the position value.

CSS Layout – The position Property

position: static;

- HTML elements are positioned static by default.
- Static positioned elements are not affected by the top, bottom, left, and right properties.
- An element with position: static; is not positioned in any special way; it is always positioned according to the normal flow of the page.

```
div.static {  
  position: static;  
  border: 3px solid #73AD21;  
}
```

This div element has position: static;

CSS Layout – The position Property

position: relative;

- An element with position: relative; is positioned relative to its normal position.
- Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position.
- Other content will not be adjusted to fit into any gap left by the element.

```
div.relative {  
  position: relative;  
  left: 30px;  
  border: 3px solid #73AD21;  
}
```

This div element has position: relative;

CSS Layout – The position Property

position: fixed;

- An element with `position: fixed;` is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled.
- The `top`, `right`, `bottom`, and `left` properties are used to position the element.

```
div.fixed {  
  position: fixed;  
  top: 50px;  
  right: 0;  
  width: 300px;  
  border: 3px solid #73AD21;  
}
```

position: fixed;

An element with `position: fixed;` is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled:

This div element has `position: fixed;`

CSS Layout – The position Property

position: absolute;

- An element with **position: absolute;** is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).
- However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.
- Absolute positioned elements are removed from the normal flow, and can overlap elements.

```
div.relative {  
  position: relative;  
  width: 400px;  
  height: 200px;  
  border: 3px solid #73AD21;  
}  
  
div.absolute {  
  position: absolute;  
  top: 80px;  
  right: 0;  
  width: 200px;  
  height: 100px;  
  border: 3px solid #73AD21;  
}
```

This <div> element has position: relative;

This <div> element has
position: absolute;

CSS Layout – The position Property

position: sticky;

- An element with **position: sticky;** is positioned based on the user's scroll position.
- A sticky element toggles between relative and fixed, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).
- Internet Explorer does not support sticky positioning. Safari requires a **-webkit- prefix**. You must also specify at least one of top, right, bottom or left for sticky positioning to work.

```
div.sticky {  
  position: -webkit-sticky; /* Safari */  
  position: sticky;  
  top: 0;  
  background-color: green;  
  border: 2px solid #4CAF50;  
}
```

Try to scroll inside this frame to understand how sticky positioning works.

I am sticky!

Lorem ipsum dolor sit amet, illum definitiones no quo, malisset concladaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.

I am sticky!

Lorem ipsum dolor sit amet, illum definitiones no quo, malisset concladaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.

Lorem ipsum dolor sit amet, illum definitiones no quo, malisset concladaturque et eum, altera fabulas ut quo. Atqui causae gloriatur ius te, id agam omnis evertitur eum. Affert laboramus repudiandae nec et. Inciderint efficiantur his ad. Eum no molestiae voluptatibus.

CSS Layout – The position Property

Positioning Text In an Image

```
<div class="container">  
    
  <div class="text">Top Left</div>  
</div>
```

```
.container {  
  position: relative;  
}  
  
.text {  
  position: absolute;  
  top: 8px;  
  left: 16px;  
  font-size: 18px;  
}  
  
img {  
  width: 100%;  
  height: auto;  
  opacity: 0.3;  
}
```

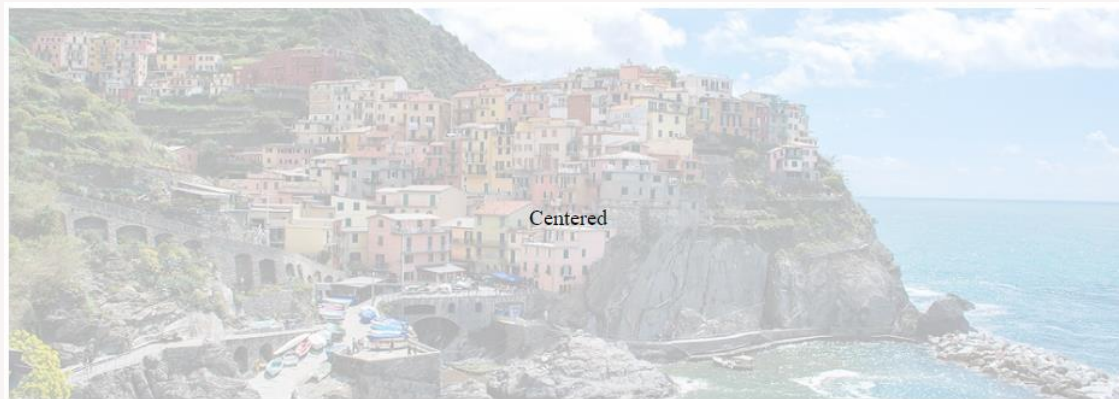


CSS Layout – The position Property

Positioning Text In an Image

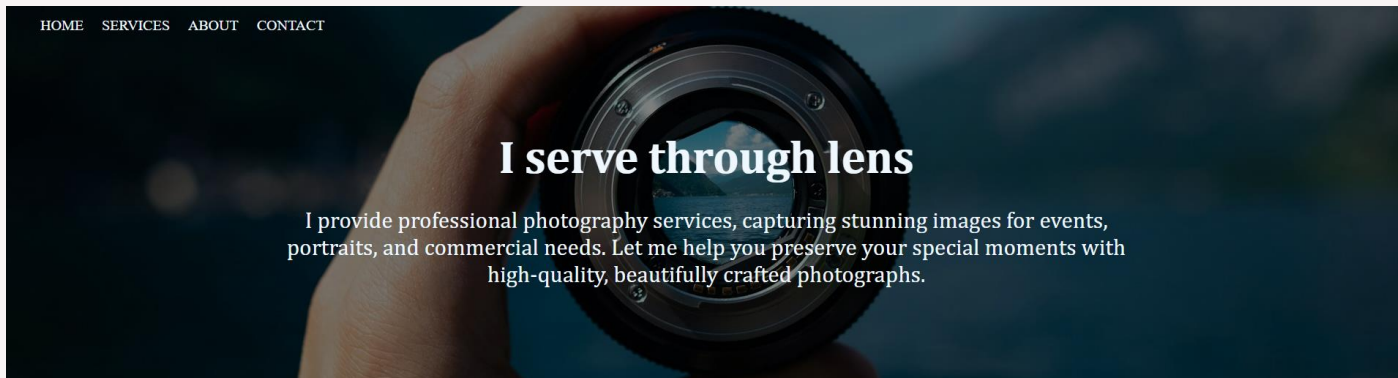
```
<div class="container">  
    
  <div class="text">Top Left</div>  
</div>
```

```
.container {  
  position: relative;  
}  
  
.text {  
  position: absolute;  
  top: 50%;  
  width: 100%;  
  text-align: center;  
  font-size: 18px;  
}  
  
img {  
  width: 100%;  
  height: auto;  
  opacity: 0.3;  
}
```



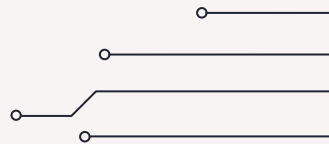
CSS Layout – The z-index Property

- The **z-index** property specifies the stack order of an element.
- When elements are positioned, they can overlap other elements.
- The **z-index** property specifies the stack order of an element (which element should be placed in front of, or behind, the others).
- An element can have a positive or negative stack order.



CSS Layout – Overflow

- The CSS overflow property controls what happens to content that is too big to fit into an area.
- The overflow property specifies whether to clip the content or to add scrollbars when the content of an element is too big to fit in the specified area.
- The overflow property has the following values:
 - `visible` - Default. The overflow is not clipped. The content renders outside the element's box
 - `hidden` - The overflow is clipped, and the rest of the content will be invisible
 - `scroll` - The overflow is clipped, and a scrollbar is added to see the rest of the content
 - `auto` - Similar to `scroll`, but it adds scrollbars only when necessary
- The overflow property only works for block elements with a specified height.
- In OS X Lion (on Mac), scrollbars are hidden by default and only shown when being used (even though "overflow:scroll" is set).



CSS Layout – Overflow

❑ overflow: visible

- By default, the overflow is visible, meaning that it is not clipped and it renders outside the element's box:

```
div {  
  width: 200px;  
  height: 65px;  
  background-color: coral;  
  overflow: visible;  
}
```

You can use the overflow property when you want to have better control of the layout. The overflow property specifies what happens if content overflows an element's box.

❑ overflow: hidden

- With the hidden value, the overflow is clipped, and the rest of the content is hidden:

```
div {  
  overflow: hidden;  
}
```

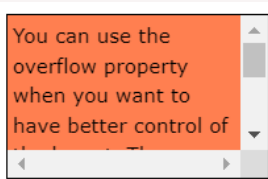
You can use the overflow property when you want to have better control of the layout. The overflow property specifies what

CSS Layout – Overflow

❑ overflow: scroll

- Setting the value to scroll, the overflow is clipped and a scrollbar is added to scroll inside the box. Note that this will add a scrollbar both horizontally and vertically (even if you do not need it):

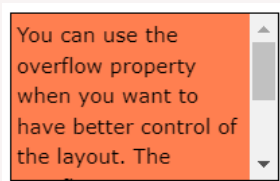
```
div {  
  overflow: scroll;  
}
```



❑ overflow: auto

- The auto value is similar to scroll, but it adds scrollbars only when necessary:

```
div {  
  overflow: auto;  
}
```

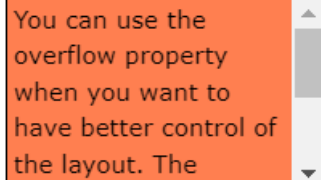


CSS Layout – Overflow

❑ overflow-x and overflow-y

- The overflow-x and overflow-y properties specifies whether to change the overflow of content just horizontally or vertically (or both):
- overflow-x specifies what to do with the left/right edges of the content.
- overflow-y specifies what to do with the top/bottom edges of the content.

```
div {  
  overflow-x: hidden; /* Hide horizontal scrollbar */  
  overflow-y: scroll; /* Add vertical scrollbar */  
}
```



You can use the
overflow property
when you want to
have better control of
the layout. The

CSS Layout – Float

- The **float** property is used for positioning and formatting content e.g. let an image float left to the text in a container.
- The float property can have one of the following values:
 - **left** - The element floats to the left of its container
 - **right** - The element floats to the right of its container
 - **none** - The element does not float (will be displayed just where it occurs in the text). This is default
 - **inherit** - The element inherits the float value of its parent

Float Left

Float Right

CSS Layout – Float and Clear

➤ Examples:

```
<div>
  
  <p>Lorem, ipsum dolor sit amet consectetur adipisicing elit. Deserunt odit earum veniam fugit quod similique placeat commodi autem provident dicta, quisquam veritatis omnis quos tempore necessitatibus consequatur amet voluptates suscipit corrupti pariatur exercitationem eaque optio sunt! Alias illum aut reiciendis ad ullam adipisci repellat earum, repellendus voluptas, tempora quaeat eligendi repudiandae amet consequuntur vero dolor fugiat ea non aperiam voluptates accusantium!</p>
</div>
```

```
img{
  float: right;
}
img{
  float: left;
}
img{
  float: none;
}
```

Lorem, ipsum dolor sit amet consectetur adipisicing elit. Deserunt odit earum veniam fugit quod similique placeat commodi autem provident dicta, quisquam veritatis omnis quos tempore necessitatibus consequatur amet voluptates suscipit corrupti pariatur exercitationem eaque optio sunt! Alias illum aut reiciendis ad ullam adipisci repellat earum, repellendus voluptas, tempora quaeat eligendi repudiandae amet consequuntur vero dolor fugiat ea non aperiam voluptates accusantium!



Lorem, ipsum dolor sit amet consectetur adipisicing elit. Deserunt odit earum veniam fugit quod similique placeat commodi autem provident dicta, quisquam veritatis omnis quos tempore necessitatibus consequatur amet voluptates suscipit corrupti pariatur exercitationem eaque optio sunt! Alias illum aut reiciendis ad ullam adipisci repellat earum, repellendus voluptas, tempora quaeat eligendi repudiandae amet consequuntur vero dolor fugiat ea non aperiam voluptates accusantium!

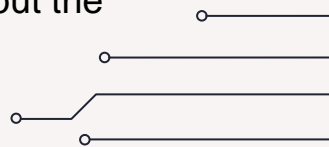


Lorem, ipsum dolor sit amet consectetur adipisicing elit. Deserunt odit earum veniam fugit quod similique placeat commodi autem provident dicta, quisquam veritatis omnis quos tempore necessitatibus consequatur amet voluptates suscipit corrupti pariatur exercitationem eaque optio sunt! Alias illum aut reiciendis ad ullam adipisci repellat earum, repellendus voluptas, tempora quaeat eligendi repudiandae amet consequuntur vero dolor fugiat ea non aperiam voluptates accusantium!

CSS Layout – Clear and Clearfix

❑ The clear Property

- When we use the float property, and we want the next element below (not on right or left), we will have to use the clear property.
- The clear property specifies what should happen with the element that is next to a floating element.
- The clear property can have one of the following values:
 - **none** - The element is not pushed below left or right floated elements. This is default
 - **left** - The element is pushed below left floated elements
 - **right** - The element is pushed below right floated elements
 - **both** - The element is pushed below both left and right floated elements
 - **inherit** - The element inherits the clear value from its parent
- **When clearing floats, you should match the clear to the float:** If an element is floated to the left, then you should clear to the left. Your floated element will continue to float, but the cleared element will appear below it on the web page.



CSS Layout – Clear and Clearfix

➤ Examples:

```
.div1 {  
  float: left;  
  padding: 10px;  
  border: 3px solid green;  
}  
  
.div2 {  
  padding: 10px;  
  border: 3px solid red;  
}  
  
.div3 {  
  float: left;  
  padding: 10px;  
  border: 3px solid green;  
}  
  
.div4 {  
  padding: 10px;  
  border: 3px solid red;  
  clear: left;  
}
```

Without clear

div1 div2 - Notice that div2 is after div1 in the HTML code. However, since div1 floats to the left, the text in div2 flows around div1.

With clear

div3
div4 - Here, clear: left; moves div4 down below the floating div3. The value "left" clears elements floated to the left. You can also clear "right" and "both".

CSS Layout – Clear and Clearfix

❑ The clearfix hack

- If a floated element is taller than the containing element, it will "overflow" outside of its container.
- We can then add a clearfix hack to solve this problem:

```
.clearfix {  
  overflow: auto;  
}
```

Without Clearfix

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum...



With Clearfix

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum...



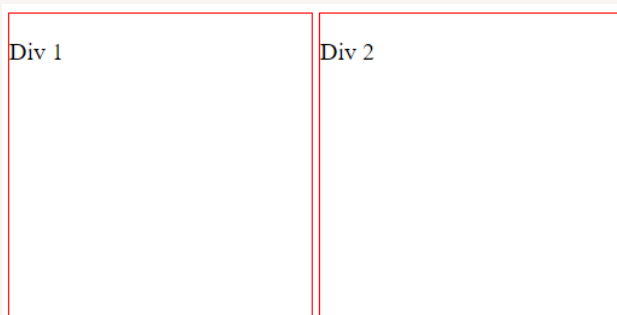
- The **overflow: auto** clearfix works well as long as you are able to keep control of your margins and padding (else you might see scrollbars). The new, modern clearfix hack however, is safer to use, and the following code is used for most webpages:

```
.clearfix::after {  
  content: "";  
  clear: both;  
  display: table;  
}
```

CSS Layout – display: inline-block

- Compared to **display: inline**, the major difference is that **display: inline-block** allows to set a width and height on the element.
- Also, with **display: inline-block**, the top and bottom margins/paddings are respected, but with **display: inline** they are not.
- Compared to **display: block**, the major difference is that **display: inline-block** does not add a line-break after the element, so the element can sit next to other elements.

```
div{  
  border: 1px solid red;  
  display: inline-block;  
  width: 200px;  
  height: 200px;  
}
```



Center Vertically – Using position & transform

```
.center {  
  height: 200px;  
  position: relative;  
  border: 3px solid green;  
}  
  
.center p {  
  margin: 0;  
  position: absolute;  
  top: 50%;  
  left: 50%;  
  transform: translate(-50%, -50%);  
}
```

I am vertically and horizontally centered.

CSS Combinators

- A combinator is something that explains the relationship between the selectors.
- A CSS selector can contain more than one simple selector. Between the simple selectors, we can include a combinator.
- There are four different combinators in CSS:
 - descendant selector (space)
 - child selector (>)
 - adjacent sibling selector (+)
 - general sibling selector (~)

1. Descendant Selector (space)

- The descendant selector matches all elements that are descendants of a specified element.

```
div p {  
  background-color: yellow;  
}
```

Paragraph 1 in the div.

Paragraph 2 in the div.

Paragraph 3 in the div.

CSS Combinators

2. Child Selector (>)

- The child selector selects all elements that are the children of a specified element.

```
div > p {  
  background-color: yellow;  
}
```

Paragraph 1 in the div.

Paragraph 2 in the div.

Paragraph 3 in the div (inside a section element).

Paragraph 4 in the div.

3. Adjacent Sibling Selector (+)

- The adjacent sibling selector is used to select an element that is directly after another specific element.
- Sibling elements must have the same parent element, and "adjacent" means "immediately following".

```
div + p {  
  background-color: yellow;  
}
```

Paragraph 1 in the div.

Paragraph 2 in the div.

Paragraph 3. After a div.

Paragraph 4. After a div.

Paragraph 5 in the div.

Paragraph 6 in the div.

Paragraph 7. After a div.

Paragraph 8. After a div.

CSS Combinators

4. General Sibling Selector (~)

- The general sibling selector selects all elements that are next siblings of a specified element.

```
div ~ p {  
  background-color: yellow;  
}
```

Paragraph 1.

Paragraph 2 inside div.

Paragraph 3.

Paragraph 4.

CSS Pseudo-classes

- A pseudo-class is used to define a special state of an element.
- For example, it can be used to:
 - Style an element when a user mouses over it
 - Style visited and unvisited links differently
 - Style an element when it gets focus
- They are indicated by a colon (:) followed by the pseudo-class name.
- Here are some common pseudo-classes in CSS:

1. Styles an element when the user hovers over it with a mouse.

```
button:hover {  
    background-color: blue;  
    color: white;  
}
```


CSS Pseudo-classes

2. Styles an element when it has focus, such as when a user clicks on an input field.

```
input:focus {  
    border-color: green;  
    outline: none;  
}
```

3. Styles an element when it is being activated or visited, like when a button is clicked.

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

```
a:visited {  
    color: purple;  
}
```

4. Styles the first child or last child element within its parent.

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

```
p:last-child {  
    color: grey;  
}
```

CSS Pseudo-classes

5. Styles the nth child element within its parent, where n can be a number, keyword, or formula.

```
li:nth-child(2) {  
  color: blue;  
}
```

```
li:nth-child(odd) {  
  background-color: lightblue;  
}
```

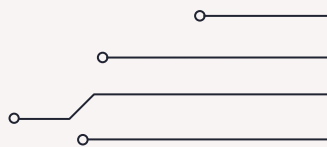
6. Styles checkboxes or radio buttons that are checked.

```
input:checked {  
  background-color: green;  
}
```

7. Inserts content before or after an element.

```
p:before {  
  content: "Note: ";  
  font-weight: bold;  
}
```

```
p:after {  
  content: " Read more.";  
}
```



CSS Pseudo-Elements

- Pseudo-elements in CSS are used to style specific parts of an element or insert content before or after an element without needing to modify the HTML directly.
- Unlike pseudo-classes, which apply styles based on the state or position of an element, pseudo-elements target specific parts of an element.
- Pseudo-elements are indicated by a double colon (::) followed by the pseudo-element name, though some older browsers may still support the single colon (:) notation.

1. **::before** & **::after**: Inserts content before and after the content of an element respectively.

```
p::before {  
  content: "Note: ";  
  color: red;  
  font-weight: bold;  
}
```

```
p::after {  
  content: " Read more...";  
  color: blue;  
}
```

CSS Pseudo-Elements

2. **::first-letter** & **::first-line**: Styles the first letter and first line of an element respectively.

```
p::first-letter {  
  font-size: 2em;  
  color: green;  
}
```

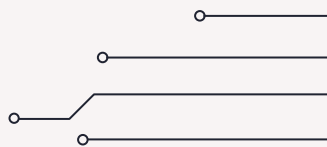
```
p::first-line {  
  font-weight: bold;  
  color: purple;  
}
```

3. **::selection**: Styles the portion of text that the user has highlighted or selected.

```
::selection {  
  background-color: yellow;  
  color: black;  
}
```

4. **::placeholder**: Styles the placeholder text inside an input field.

```
input::placeholder {  
  color: grey;  
  font-style: italic;  
}
```



CSS Pseudo-Elements

5. **::marker**: Styles the marker of list items, such as bullets or numbers.

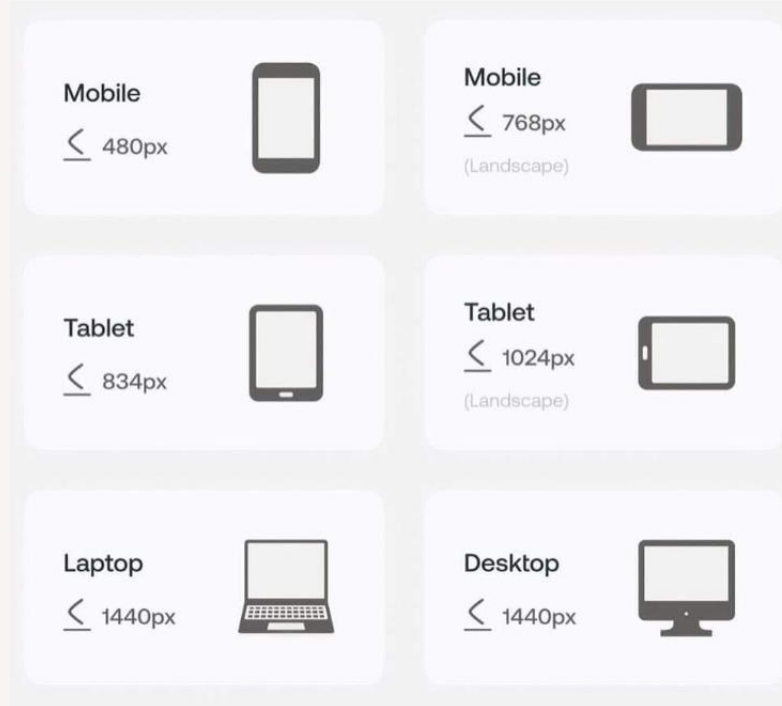
```
li::marker {  
  color: red;  
  font-size: 1.5em;  
}
```

Web Breakpoints

- **Breakpoints** in web design are specific points defined in the CSS where the layout or style of a webpage changes based on the screen size or device type. They are used in responsive design to ensure that a website looks good and functions well on different devices, such as desktops, tablets, and smartphones.

Breakpoint	Class infix	Dimensions
X-Small	<i>None</i>	<576px
Small	<i>sm</i>	≥576px
Medium	<i>md</i>	≥768px
Large	<i>lg</i>	≥992px
Extra large	<i>xl</i>	≥1200px
Extra extra large	<i>xxl</i>	≥1400px

Web Breakpoints



CSS Media Queries

- The @media rule, introduced in CSS2, made it possible to define different style rules for different media types.
- Media queries in CSS3 extended the CSS2 media types idea: Instead of looking for a type of device, they look at the capability of the device.
- **Media queries can be used to check many things, such as:**
 - width and height of the viewport
 - orientation of the viewport (landscape or portrait)
 - resolution
- Using media queries are a popular technique for delivering a tailored style sheet to desktops, laptops, tablets, and mobile phones (such as iPhone and Android phones).

CSS Media Queries

❑ CSS Media Type

Value	Description
all	Used for all media type devices
print	Used for print preview mode
screen	Used for computer screens, tablets, smart-phones etc.

❑ CSS Common Media Features

Value	Description
orientation	Orientation of the viewport. Landscape or portrait
max-height	Maximum height of the viewport
min-height	Minimum height of the viewport
height	Height of the viewport (including scrollbar)
max-width	Maximum width of the viewport
min-width	Minimum width of the viewport
width	Width of the viewport (including scrollbar)

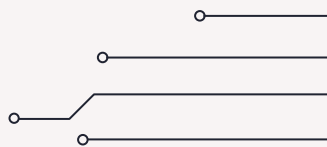
CSS Media Queries

❑ Media Query Syntax

- A media query consists of a media type and can contain one or more media features, which resolve to either true or false.

```
@media not|only mediatype and (media feature) and (media feature) {  
    CSS-Code;  
}
```

- The **mediatype** is optional (if omitted, it will be set to all). However, if you use not or only, you must also specify a **mediatype**.
- The result of the query is true if the specified media type matches the type of device the document is being displayed on and all media features in the media query are true. When a media query is true, the corresponding style sheet or style rules are applied, following the normal cascading rules.



CSS Media Queries

❑ Meaning of the not, only, and and keywords

- a. **not:** This keyword inverts the meaning of an entire media query.
 - b. **only:** This keyword prevents older browsers that do not support media queries from applying the specified styles. It has no effect on modern browsers.
 - c. **and:** This keyword combines a media type and one or more media features.
- You can also link to different stylesheets for different media and different widths of the browser window (viewport):

```
<link rel="stylesheet" media="print" href="print.css">
<link rel="stylesheet" media="screen" href="screen.css">
<link rel="stylesheet" media="screen and (min-width: 480px)" href="example1.css">
<link rel="stylesheet" media="screen and (min-width: 701px) and (max-width: 900px)" href="example2.css">
etc....
```

CSS Media Queries

❑ Examples:

- **Basic Media Query for Screen Width**

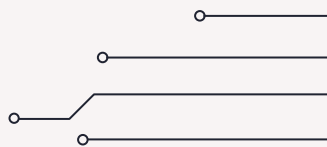
➤ This query applies styles only if the screen width is 600px or less.

```
@media (max-width: 600px) {  
  body {  
    background-color: lightblue;  
  }  
}
```

- **Media Query for Minimum Screen Width**

➤ This query applies styles if the screen width is at least 768px.

```
@media (min-width: 768px) {  
  body {  
    font-size: 18px;  
  }  
}
```



CSS Media Queries

- **Media Query for Orientation**

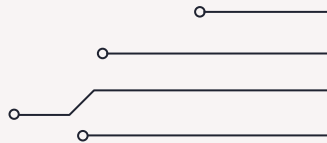
- This query applies styles based on the orientation of the device.

```
@media (orientation: landscape) {  
  .container {  
    flex-direction: row;  
  }  
}
```

- **Combining Conditions**

- You can combine multiple conditions in a single media query.

```
@media (min-width: 768px) and (max-width: 1024px) and (orientation: landscape) {  
  .sidebar {  
    display: none;  
  }  
}
```



CSS Media Queries

- **Media Query for Print**

- This query applies styles when printing the document.

```
@media print {  
  body {  
    font-size: 12pt;  
    color: black;  
  }  
  
  .no-print {  
    display: none;  
  }  
}
```

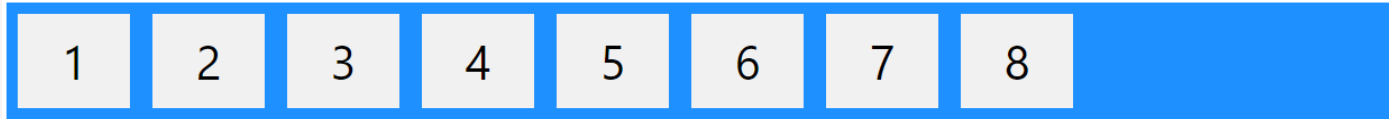
- **Media Query for Dark Mode**

- This query applies styles when the user has enabled dark mode on their device.

```
@media (prefers-color-scheme: dark) {  
  body {  
    background-color: #333;  
    color: #fff;  
  }  
}
```

CSS Flexbox Layout

- This query applies styles when printing the document.
- Before the Flexbox Layout module, there were four layout modes:
 - i. Block, for sections in a webpage
 - ii. Inline, for text
 - iii. Table, for two-dimensional table data
 - iv. Positioned, for explicit position of an element
- The **Flexible Box** Layout Module, makes it easier to design flexible responsive layout structure without using float or positioning.



CSS Flexbox Layout

❑ CSS Flex Container

➤ The flex container properties are:

- i. flex-direction
- ii. flex-wrap
- iii. flex-flow
- iv. justify-content
- v. align-items
- vi. align-content

CSS Flexbox Layout

i. flex-direction

- The **flex-direction** property defines in which direction the container wants to stack the flex items.
- The **column** value stacks the flex items vertically (from top to bottom):

```
<div class="flex-container">  
  <div>1</div>  
  <div>2</div>  
  <div>3</div>  
</div>
```

```
.flex-container {  
  display: flex;  
  flex-direction: column;  
  background-color: DodgerBlue;  
}  
  
.flex-container > div {  
  background-color: #f1f1f1;  
  width: 100px;  
  height: 100px;  
  border: 1px solid red  
}
```



CSS Flexbox Layout

- The **column-reverse** value stacks the flex items vertically (but from bottom to top):

```
<div class="flex-container">  
  <div>1</div>  
  <div>2</div>  
  <div>3</div>  
</div>
```

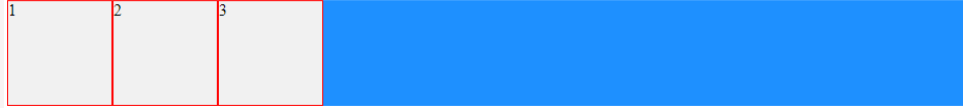
```
.flex-container {  
  display: flex;  
  flex-direction: column-reverse;  
  background-color: DodgerBlue;  
}  
  
.flex-container > div {  
  background-color: #f1f1f1;  
  width: 100px;  
  height: 100px;  
  border: 1px solid red  
}
```



CSS Flexbox Layout

- The **row** value stacks the flex items horizontally (from left to right):

```
.flex-container {  
  display: flex;  
  flex-direction: row;  
  background-color: DodgerBlue;  
}
```



- The **row-reverse** value stacks the flex items horizontally (but from right to left):

```
.flex-container {  
  display: flex;  
  flex-direction: row-reverse;  
  background-color: DodgerBlue;  
}
```

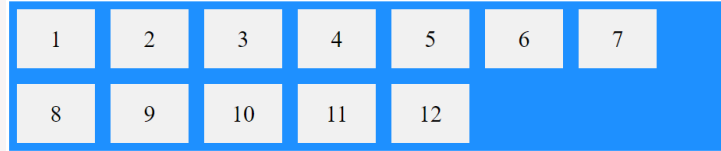


CSS Flexbox Layout

ii. flex-wrap

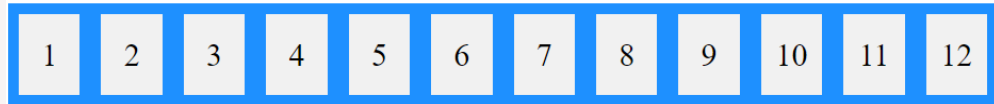
- The **flex-wrap** property specifies whether the flex items should wrap or not.
- The **wrap** value specifies that the flex items will wrap if necessary:

```
.flex-container {  
  display: flex;  
  flex-wrap: wrap;  
  background-color: DodgerBlue;  
}
```



- The **nowrap** value specifies that the flex items will not wrap (this is default):

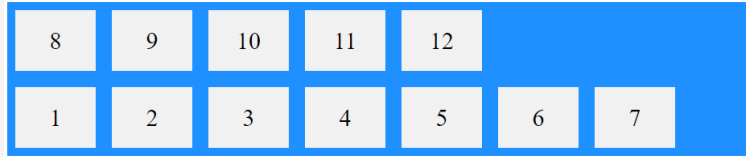
```
.flex-container {  
  display: flex;  
  flex-wrap: nowrap;  
  background-color: DodgerBlue;  
}
```



CSS Flexbox Layout

- The **wrap-reverse** value specifies that the flexible items will wrap if necessary, in reverse order:

```
.flex-container {  
  display: flex;  
  flex-wrap: wrap-reverse;  
  background-color: DodgerBlue;  
}
```

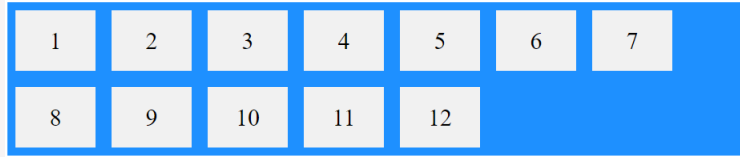


CSS Flexbox Layout

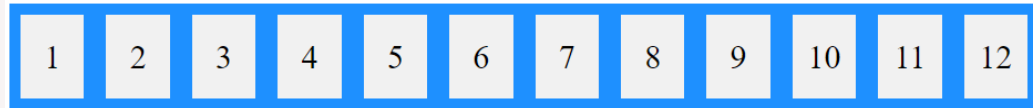
iii. flex-flow

- The **flex-flow** property is a shorthand property for setting both the **flex-direction** and **flex-wrap** properties:

```
.flex-container {  
  display: flex;  
  flex-flow: row wrap;  
  background-color: DodgerBlue;  
}
```



```
.flex-container {  
  display: flex;  
  flex-flow: row nowrap;  
  background-color: DodgerBlue;  
}
```

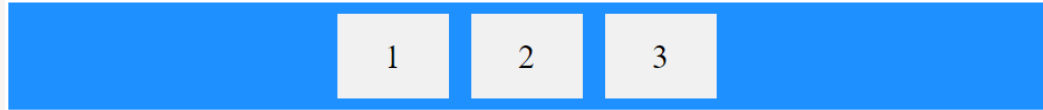


CSS Flexbox Layout

iv. justify-content

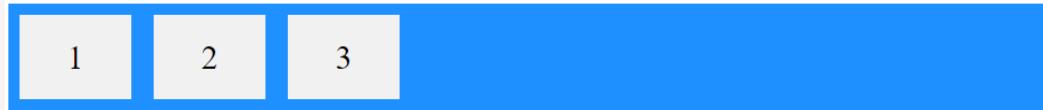
- The **justify-content** property is used to align the flex items.
- The **center** value aligns the flex items at the center of the container:

```
.flex-container {  
  display: flex;  
  justify-content: center;  
  background-color: DodgerBlue;  
}
```



- The **flex-start** value aligns the flex items at the beginning of the container (this is default):

```
.flex-container {  
  display: flex;  
  justify-content: flex-start;  
  background-color: DodgerBlue;  
}
```



CSS Flexbox Layout

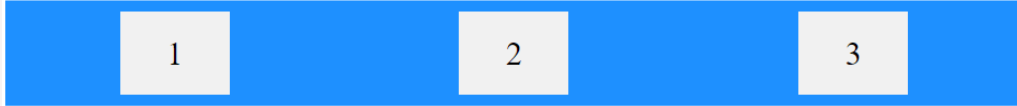
- The **flex-end** value aligns the flex items at the end of the container:

```
.flex-container {  
  display: flex;  
  justify-content: flex-end;  
  background-color: DodgerBlue;  
}
```



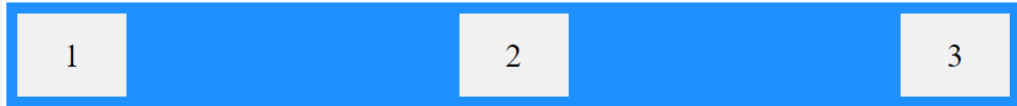
- The **space-around** value displays the flex items with space before, between, and after the lines:

```
.flex-container {  
  display: flex;  
  justify-content: space-around;  
  background-color: DodgerBlue;  
}
```



- The **space-between** value displays the flex items with space between the lines:

```
.flex-container {  
  display: flex;  
  justify-content: space-between;  
  background-color: DodgerBlue;  
}
```



CSS Flexbox Layout

v. align-items

- The **align-items** property is used to align the flex items.
- The **center** value aligns the flex items in the middle of the container:

```
.flex-container {  
  display: flex;  
  height: 200px;  
  align-items: center;  
  background-color: DodgerBlue;  
}
```



- The **flex-start** value aligns the flex items at the top of the container:

```
.flex-container {  
  display: flex;  
  height: 200px;  
  align-items: flex-start;  
  background-color: DodgerBlue;  
}
```



CSS Flexbox Layout

- The **flex-end** value aligns the flex items at the bottom of the container:

```
.flex-container {  
  display: flex;  
  height: 200px;  
  align-items: flex-end;  
  background-color: DodgerBlue;  
}
```



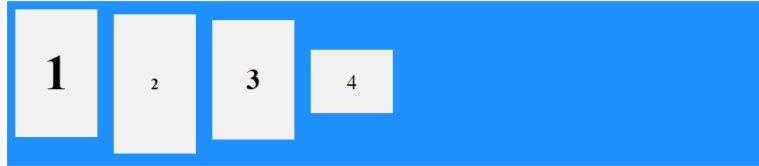
- The **stretch** value stretches the flex items to fill the container (this is default):

```
.flex-container {  
  display: flex;  
  height: 200px;  
  align-items: stretch;  
  background-color: DodgerBlue;  
}
```



- The **baseline** value aligns the flex items such as their baselines aligns:

```
.flex-container {  
  display: flex;  
  height: 200px;  
  align-items: baseline;  
  background-color: DodgerBlue;  
}
```

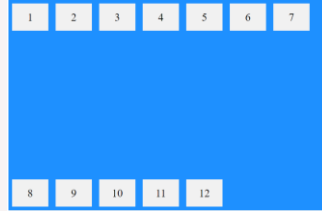


CSS Flexbox Layout

vi. align-content

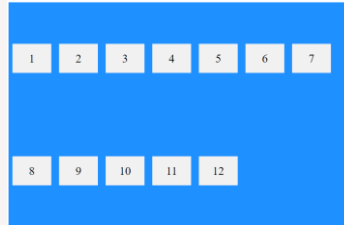
- The **align-content** property is used to align the flex lines.
- The **space-between** value displays the flex lines with equal space between them:

```
.flex-container {  
  display: flex;  
  height: 600px;  
  flex-wrap: wrap;  
  align-content: space-between;  
  overflow: scroll;  
  background-color: DodgerBlue;  
}
```



- The **space-around** value displays the flex lines with space before, between, and after them:

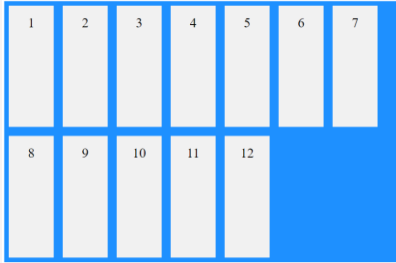
```
.flex-container {  
  display: flex;  
  height: 600px;  
  flex-wrap: wrap;  
  align-content: space-around;  
  overflow: scroll;  
  background-color: DodgerBlue;  
}
```



CSS Flexbox Layout

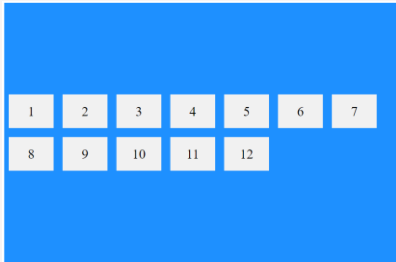
- The **stretch** value stretches the flex lines to take up the remaining space (this is default):

```
.flex-container {  
  display: flex;  
  height: 600px;  
  flex-wrap: wrap;  
  align-content: stretch;  
  overflow: scroll;  
  background-color: DodgerBlue;  
}
```



- The **center** value displays the flex lines in the middle of the container:

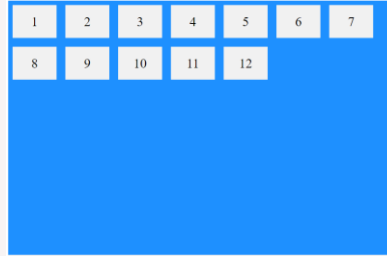
```
.flex-container {  
  display: flex;  
  height: 600px;  
  flex-wrap: wrap;  
  align-content: center;  
  overflow: scroll;  
  background-color: DodgerBlue;  
}
```



CSS Flexbox Layout

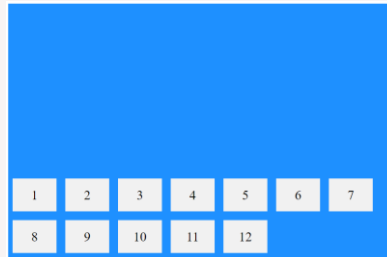
- The **flex-start** value displays the flex lines at the start of the container:

```
.flex-container {  
  display: flex;  
  height: 600px;  
  flex-wrap: wrap;  
  align-content: flex-start;  
  overflow: scroll;  
  background-color: DodgerBlue;  
}
```



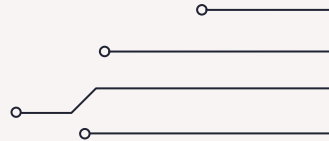
- The **flex-end** value displays the flex lines at the end of the container:

```
.flex-container {  
  display: flex;  
  height: 600px;  
  flex-wrap: wrap;  
  align-content: flex-end;  
  overflow: scroll;  
  background-color: DodgerBlue;  
}
```



CSS Flex Items

- The direct child elements of a flex container automatically becomes flexible (flex) items.
- The flex item properties are:
 - i. order
 - ii. flex-grow
 - iii. flex-shrink
 - iv. flex-basis
 - v. flex
 - vi. align-self

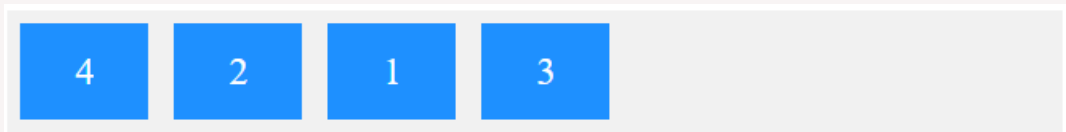


CSS Flex Items

i. order

- The **order** property specifies the order of the flex items.
- The first flex item in the code does not have to appear as the first item in the layout.
- The **order** value must be a number, default value is 0.

```
.flex-container {  
  display: flex;  
  align-items: stretch;  
  background-color: #f1f1f1;  
}  
  
.flex-container>div {  
  background-color: DodgerBlue;  
  color: white;  
  width: 100px;  
  margin: 10px;  
  text-align: center;  
  line-height: 75px;  
  font-size: 30px;  
}  
  
<div class="flex-container">  
  <div style="order: 3">1</div>  
  <div style="order: 2">2</div>  
  <div style="order: 4">3</div>  
  <div style="order: 1">4</div>  
</div>
```



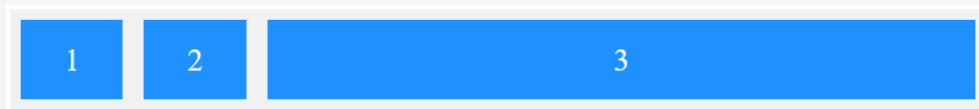
CSS Flex Items

ii. flex-grow

- The **flex-grow** property specifies how much a flex item will grow relative to the rest of the flex items.
- The value must be a number, default value is 0.

```
.flex-container {  
  display: flex;  
  align-items: stretch;  
  background-color: #f1f1f1;  
}  
  
.flex-container > div {  
  background-color: DodgerBlue;  
  color: white;  
  margin: 10px;  
  text-align: center;  
  line-height: 75px;  
  font-size: 30px;  
}
```

```
<div class="flex-container">  
  <div style="flex-grow: 1">1</div>  
  <div style="flex-grow: 1">2</div>  
  <div style="flex-grow: 8">3</div>  
</div>
```



CSS Flex Items

iii. flex-shrink

- The **flex-shrink** property specifies how much a flex item will shrink relative to the rest of the flex items.
- The value must be a number, default value is 1.

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="flex-shrink: 0">3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
  <div>8</div>
  <div>9</div>
  <div>10</div>
</div>
```

Do not let the third flex item shrink as much as the other flex items:



CSS Flex Items

iv. flex-basis

- The **flex-basis** property specifies the initial length of a flex item.

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="flex-basis:200px">3</div>
  <div>4</div>
</div>
```

Set the initial length of the third flex item to 200 pixels:



v. flex

- The **flex** property is a shorthand property for the **flex-grow**, **flex-shrink**, and **flex-basis** properties.

```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="flex: 0 0 200px">3</div>
  <div>4</div>
</div>
```

Make the third flex item not growable (0), not shrinkable (0), and with an initial length of 200 pixels:

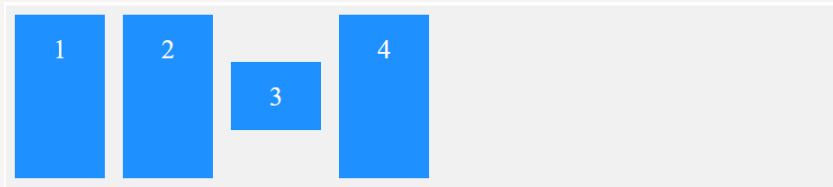


CSS Flex Items

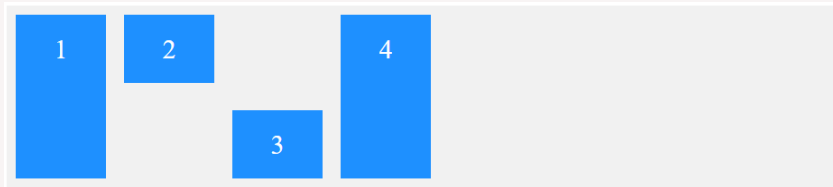
v. align-self

- The **align-self** property specifies the alignment for the selected item inside the flexible container.
- The **align-self** property overrides the default alignment set by the container's **align-items** property.

```
<div class="flex-container">  
  <div>1</div>  
  <div>2</div>  
  <div style="align-self: center">3</div>  
  <div>4</div>  
</div>
```



```
<div class="flex-container">  
  <div>1</div>  
  <div style="align-self: flex-start">2</div>  
  <div style="align-self: flex-end">3</div>  
  <div>4</div>  
</div>
```



CSS Grid Layout

- The CSS **Grid Layout Module** offers a grid-based layout system, with rows and columns, making it easier to design web pages without having to use floats and positioning.
- An HTML element becomes a grid container when its **display** property is set to **grid** or **inline-grid**.

```
.grid-container {  
  display: grid;  
  grid-template-columns: auto auto auto;  
  background-color: #2196F3;  
  padding: 10px;  
}
```

1	2	3
4	5	6
7	8	9

```
.grid-container {  
  display: inline-grid;  
  grid-template-columns: auto auto auto;  
  background-color: #2196F3;  
  padding: 10px;  
}
```

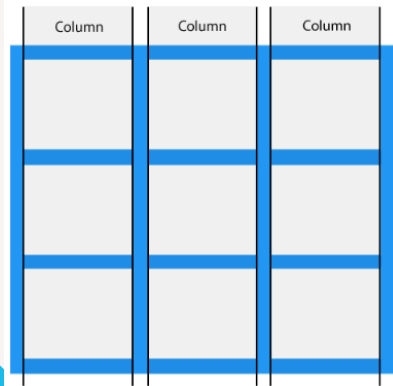
1	2	3
4	5	6
7	8	9

CSS Grid Layout

- All direct children of the grid container automatically become grid items.
- You can adjust the gap size by using one of the following properties:
 - **column-gap**
 - **row-gap**
 - **gap**

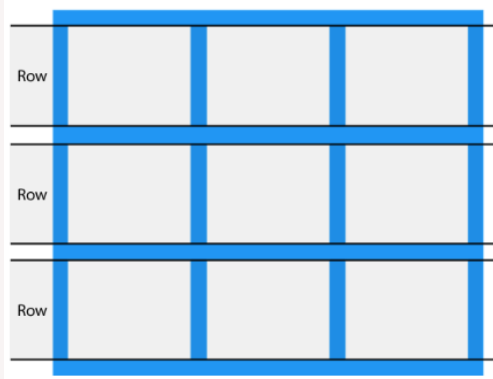
Grid Columns

The vertical lines of grid items are called *columns*.



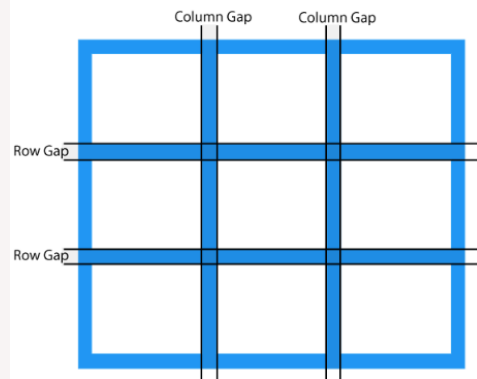
Grid Rows

The horizontal lines of grid items are called *rows*.



Grid Gaps

The spaces between each column/row are called *gaps*.



CSS Grid Layout

- The **column-gap** property sets the gap between the columns:

```
.grid-container {  
  display: grid;  
  column-gap: 50px;  
  grid-template-columns: auto auto auto;  
  background-color: #2196F3;  
  padding: 10px;  
}
```

1	2	3
4	5	6
7	8	9

- The **row-gap** property sets the gap between the rows:

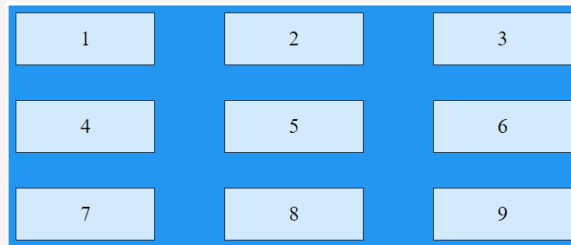
```
.grid-container {  
  display: grid;  
  row-gap: 50px;  
  grid-template-columns: auto auto auto;  
  background-color: #2196F3;  
  padding: 10px;  
}
```

1	2	3
4	5	6
7	8	9

CSS Grid Layout

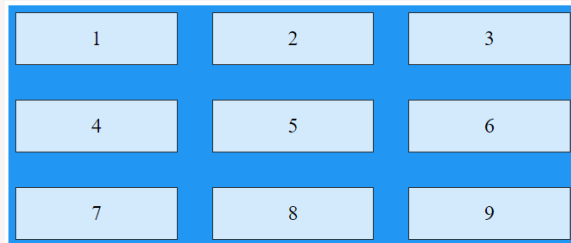
- The **gap** property is a shorthand property for the **row-gap** and the **column-gap** properties:

```
.grid-container {  
  display: grid;  
  gap: 50px 100px;  
  grid-template-columns: auto auto auto;  
  background-color: #2196F3;  
  padding: 10px;  
}
```



- The **gap** property can also be used to set both the **row gap** and the **column gap** in one value:

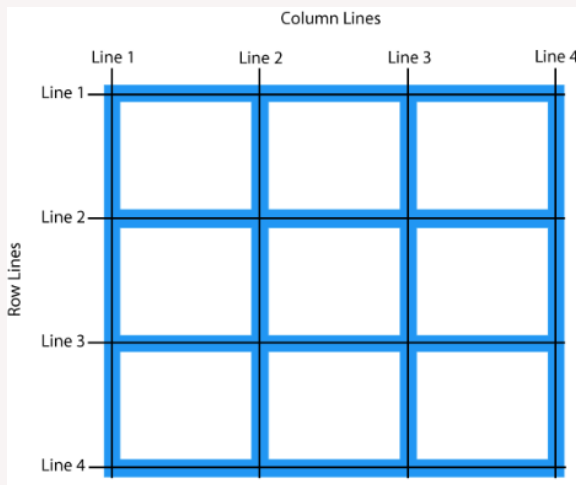
```
.grid-container {  
  display: grid;  
  gap: 50px;  
  grid-template-columns: auto auto auto;  
  background-color: #2196F3;  
  padding: 10px;  
}
```



CSS Grid Layout

❑ Grid Lines

- The lines between columns are called column lines.
- The lines between rows are called row lines.
- Refer to line numbers when placing a grid item in a grid container:



CSS Grid Layout

```
<div class="grid-container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
  <div class="item6">6</div>
  <div class="item7">7</div>
  <div class="item8">8</div>
</div>
```

```
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
```

```
.item1 {
  grid-column-start: 1;
  grid-column-end: 3;
}
```

1		2
3	4	5
6	7	8

CSS Grid Layout

```
<div class="grid-container">  
  <div class="item1">1</div>  
  <div class="item2">2</div>  
  <div class="item3">3</div>  
  <div class="item4">4</div>  
  <div class="item5">5</div>  
  <div class="item6">6</div>  
  <div class="item7">7</div>  
  <div class="item8">8</div>  
</div>
```

```
.grid-container {  
  display: grid;  
  grid-template-columns: auto auto auto;  
  gap: 10px;  
  background-color: #2196F3;  
  padding: 10px;  
}
```

```
.item1 {  
  grid-row-start: 1;  
  grid-row-end: 3;  
}
```

1	2	3
	4	5
6	7	8

CSS Grid Layout

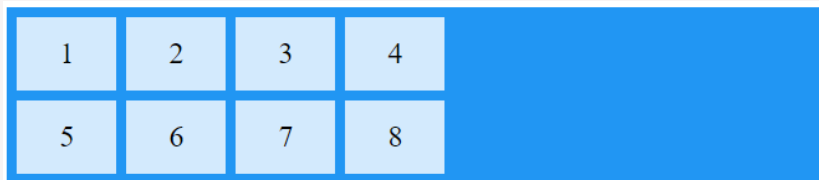
❑ Grid Container

- To make an HTML element behave as a grid container, you have to set the display property to **grid** or **inline-grid**.
- Grid containers consist of grid items, placed inside columns and rows.

i. The **grid-template-columns** Property

- The **grid-template-columns** property defines the number of columns in your grid layout, and it can define the width of each column.
- The value is a space-separated-list, where each value defines the width of the respective column.
- If you want your grid layout to contain 4 columns, specify the width of the 4 columns, or "auto" if all columns should have the same width.

```
.grid-container {  
  display: grid;  
  grid-template-columns: 100px 100px 100px 100px;  
  gap: 10px;  
  background-color: #2196F3;  
  padding: 10px;  
}
```



CSS Grid Layout

ii. The grid-template-rows Property

- The **grid-template-rows** property defines the height of each row.
- The value is a space-separated-list, where each value defines the height of the respective row:

```
.grid-container {  
  display: grid;  
  grid-template-columns: auto auto auto;  
  grid-template-rows: 80px 200px;  
  gap: 10px;  
  background-color: #2196F3;  
  padding: 10px;  
}
```

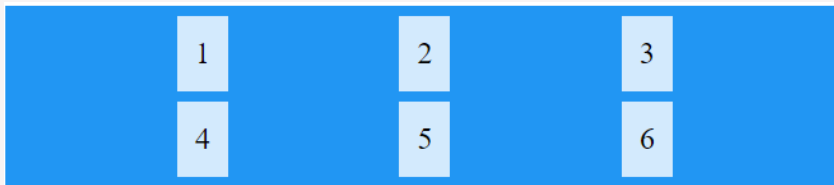
1	2	3
4	5	6

CSS Grid Layout

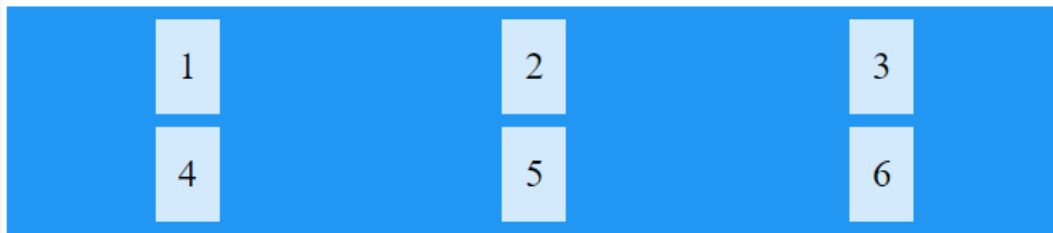
iii. The justify-content Property

- The **justify-content** property is used to align the whole grid inside the container.
- The value **space-evenly** will give the columns equal amount of space between, and around them:

```
.grid-container {  
  display: grid;  
  justify-content: space-evenly;  
  grid-template-columns: 50px 50px 50px;  
  gap: 10px;  
  background-color: #2196F3;  
  padding: 10px;  
}
```



- The value **space-around** will give the columns equal amount of space around them:

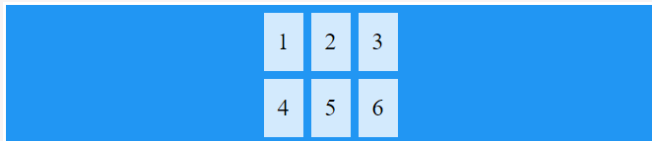


CSS Grid Layout

- The value **space-between** will give the columns equal amount of space between them:



- The value **center** will align the grid in the middle of the container:



- The value **start** and **end** will align the grid at the beginning and end of the container respectively:

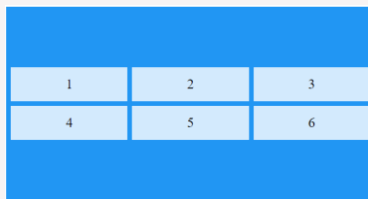


CSS Grid Layout

iv. The align-content Property

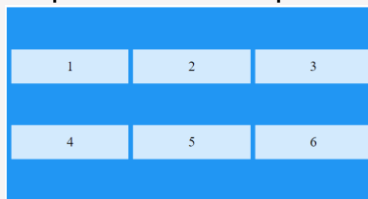
- The **align-content** property is used to vertically align the whole grid inside the container.
- The grid's total height has to be less than the container's height for the align-content property to have any effect.
- The value **center** will align the rows in the middle of the container:

```
.grid-container {  
  display: grid;  
  height: 400px;  
  align-content: center;  
  grid-template-columns: auto auto auto;  
  gap: 10px;  
  background-color: #2196F3;  
  padding: 10px;  
}
```



- The value **space-evenly** will give the rows equal amount of space between, and around them:

```
.grid-container {  
  display: grid;  
  height: 400px;  
  align-content: space-evenly;  
  grid-template-columns: auto auto auto;  
  gap: 10px;  
  background-color: #2196F3;  
  padding: 10px;  
}
```



CSS Grid Layout

- The value **space-around** will give the rows equal amount of space around them:

1	2	3
4	5	6

- The value **space-between** will give the rows equal amount of space between them:

1	2	3
4	5	6

- The value **start** and **end** will align the rows at the beginning and end of the container respectively:

1	2	3
4	5	6

1	2	3
4	5	6

CSS Grid Layout

❑ CSS Grid Item

- A grid container contains grid items.
- By default, a container has one grid item for each column, in each row, but you can style the grid items so that they will span multiple columns and/or rows.

i. The grid-column Property:

- The **grid-column** property defines on which column(s) to place an item.
- You define where the item will start, and where the item will end.
- The **grid-column** property is a shorthand property for the **grid-column-start** and the **grid-column-end** properties.

```
.grid-container {  
  display: grid;  
  grid-template-columns: auto auto auto auto auto auto;  
  gap: 10px;  
  background-color: #2196F3;  
  padding: 10px;  
}  
  
.item1 {  
  grid-column: 1 / 5;  
}
```

1				2	3
4	5	6	7	8	9
10	11	12	13	14	15

CSS Grid Layout

- To place an item, you can refer to line numbers, or use the keyword **span** to define how many columns the item will span.

```
.item1 {  
  grid-column: 1 / span 3;  
}
```

1			2	3	4
5	6	7	8	9	10
11	12	13	14	15	16

- Item2 will start on column line 2 and span 3 columns:

```
.item2 {  
  grid-column: 2 / span 3;  
}
```

1	2			3	4
5	6	7	8	9	10
11	12	13	14	15	16

CSS Grid Layout

ii. The grid-row Property:

- The **grid-row** property defines on which row to place an item.
- You define where the item will start, and where the item will end.
- The **grid-row** property is a shorthand property for the **grid-row-start** and the **grid-row-end** properties.

```
.item1 {  
  grid-row: 1 / 4;  
}
```

1	2	3	4	5	6
	7	8	9	10	11
	12	13	14	15	16

- To place an item, you can refer to line numbers, or use the keyword **span** to define how many rows the item will span:

```
.item1 {  
  grid-row: 1 / span 2;  
}
```

1	2	3	4	5	6
	7	8	9	10	11
	12	13	14	15	16
17					

CSS Grid Layout

iii. The grid-area Property:

- The **grid-area** property can be used as a shorthand property for the **grid-row-start**, **grid-column-start**, **grid-row-end** and the **grid-column-end** properties.
- Make **item8** start on **row-line 1** and **column-line 2**, and end on **row-line 5** and **column-line 6**:

```
.item8 {  
  grid-area: 1 / 2 / 5 / 6;  
}
```

1	8				2
3					4
5					6
7					9
10	11	12	13	14	15

- Make **item8** start on **row-line 2** and **column-line 1**, and span 2 rows and 3 columns:

```
.item8 {  
  grid-area: 2 / 1 / span 2 / span 3;  
}
```

1	2	3	4	5	6
8			7	9	10
			11	12	13

End of CSS