



AMIT

BEYOND EDUCATION



CSS

Cascading style sheet

CSS POSITION PROPERTY

The CSS position property is used to define the position of an element on a webpage.

The location of the positioned element is set with the four properties: top, left, right, and bottom. These properties only work when the position property is set and have different positioning behaviors.

The position property has the following five values:

- static (default value)
- relative
- absolute
- fixed
- sticky

CSS STATIC POSITION

The static value of the position property allows elements to be positioned accordingly to the normal flow of the document. The static position is not affected by the top, right, bottom, and left values.

Note: The static value is the default value

🌐 CSS position >

This is a normal paragraph.

This paragraph is positioned using the static value.

This is a normal paragraph.

```
p {  
  border: 1px solid black;  
  padding: 4px;  
}  
  
p.main {  
  position: static;  
  top: 50px; /* doesn't work */  
  right: 50px; /* doesn't work */  
  bottom: 50px; /* doesn't work */  
  left: 50%; /* doesn't work */  
  background-color: greenyellow;  
}
```

```
<body>  
  <p>This is a normal paragraph.</p>  
  <p class="main">This paragraph is positioned using the static value.</p>  
  <p>This is a normal paragraph.</p>  
</body>
```

CSS RELATIVE POSITION

The relative value positions the element relative to the original position in the document. The element is positioned with the top, right, bottom, and left values.

The space is preserved in the original position of the element.

🌐 CSS position >

This is a normal paragraph.

This is a normal paragraph.

This paragraph is positioned with a relative value of top 50px and left 40px.

```
p {  
  border: 1px solid black;  
  padding: 4px;  
}  
  
p.main {  
  position: relative;  
  /* positions 90px from the top */  
  top: 90px;  
  
  /* positions 40px from the left */  
  left: 40px;  
  background-color: greenyellow;  
}
```

```
<body>  
  <p>This is a normal paragraph.</p>  
  <p class="main">  
    This paragraph is positioned with a relative value of top 50px and  
    left 40px.  
  </p>  
  <p>This is a normal paragraph.</p>  
</body>
```

CSS ABSOLUTE POSITION

The absolute value removes the element completely from its normal flow in the document.

The element is positioned relative to their closest positioned parent element (an ancestor element with a position value other than static).

If there is no positioned ancestor, they are positioned relative to the document itself.

second paragraph doesn't have a positioned parent element, so it is positioned relative to the viewport (visible area of the browser).

```
p {  
  border: 1px solid black;  
  padding: 4px;  
}  
  
p.main {  
  position: absolute;  
  top: 70px;  
  left: 60px;  
  background-color: greenyellow;  
}
```

🌐 CSS position >

This is a normal paragraph.

This is a normal paragraph.

This is an absolutely positioned paragraph at 40px top and 60px left.

```
<body>  
  <p>This is a normal paragraph.</p>  
  <p class="main">  
    This is an absolutely positioned paragraph at 40px top and 60px left.  
  </p>  
  <p>This is a normal paragraph.</p>  
</body>
```

CSS ABSOLUTE PROPERTY

even though both paragraphs of the main class are given the same styles, their positions are different. This is because the first paragraph doesn't have a positioned parent element, so it is positioned relative to the document (viewport).

On the other hand, the second paragraph has a parent div element with a relative position value and is hence positioned relative to the div element.

Note: An absolutely positioned element loses its size and original space in the document flow.

This is a normal paragraph.

This is a normal paragraph.

This is an absolutely positioned paragraph at 40px top and 60px left.

This paragraph is inside the div element.

This paragraph is inside the div element.

This is an absolutely positioned paragraph inside the div element at 40px top and 60px left.

```
p {  
  border: 1px solid black;  
  padding: 4px;  
}  
  
p.main {  
  position: absolute;  
  top: 70px;  
  left: 60px;  
  background-color: greenyellow;  
}  
  
div.parent {  
  position: relative;  
  background-color: orange;  
  border: 4px solid red;  
}
```


CSS FIXED PROPERTY

The fixed value positions an element to remain fixed in the same position, even when the page is scrolled. It is similar to the absolute value, but it remains relative to the viewport at all times.

fixes the paragraph at the 10px distance from the top in the webpage. The paragraph doesn't scroll with the other content in the document

```
p {  
  border: 2px solid black;  
  padding: 4px;  
}  
  
p.main {  
  position: fixed;  
  top: 10px;  
  background-color: greenyellow;  
}  
  
<body>  
  <p class="main">  
    This paragraph has a fixed position value at 10px top.  
  </p>  
  <p>This is a normal paragraph.</p>  
  <p>This is a normal paragraph.</p>  
  <p>This is a normal paragraph.</p>  
  <p>This is a normal paragraph.</p>  
  <p>This is a normal paragraph.</p>  
  <p>This is a normal paragraph.</p>  
  <p>This is a normal paragraph.</p>  
</body>
```

This paragraph has a fixed position value at 10px top.

This is a normal paragraph.

This is a normal paragraph.

CSS STICKY POSITION

The sticky value positions the element as a combination of relative and fixed values.

The sticky position behaves like relative positioning until the element reaches a certain scroll point on the screen. After that, the element sticks to the top of the viewport like a fixed element.

the paragraph remains scrollable until it reaches to 10px distance from the top of the viewport. After that, it remains in the sticky position.

```
<body>
  <p class="main">
    This paragraph has a fixed position value at 10px top.
  </p>
  <p>This is a normal paragraph.</p>
  <p>This is a normal paragraph.</p>
  <p>This is a normal paragraph.</p>
  <p>This is a normal paragraph.</p>
  <p>This is a normal paragraph.</p>
  <p>This is a normal paragraph.</p>
  <p>This is a normal paragraph.</p>
</body>
```

```
p {
  border: 2px solid black;
  padding: 4px;
}

p.main {
  position: fixed;
  top: 10px;
  background-color: greenyellow;
}
```

This is a normal paragraph.

This paragraph is set sticky position at top 10px.

This is a normal paragraph.

CSS Z-INDEX PROPERTY

The CSS z-index property is used to control the stacking order of the positioned elements.

The z-index property only works with positioned elements and items of the flex container.

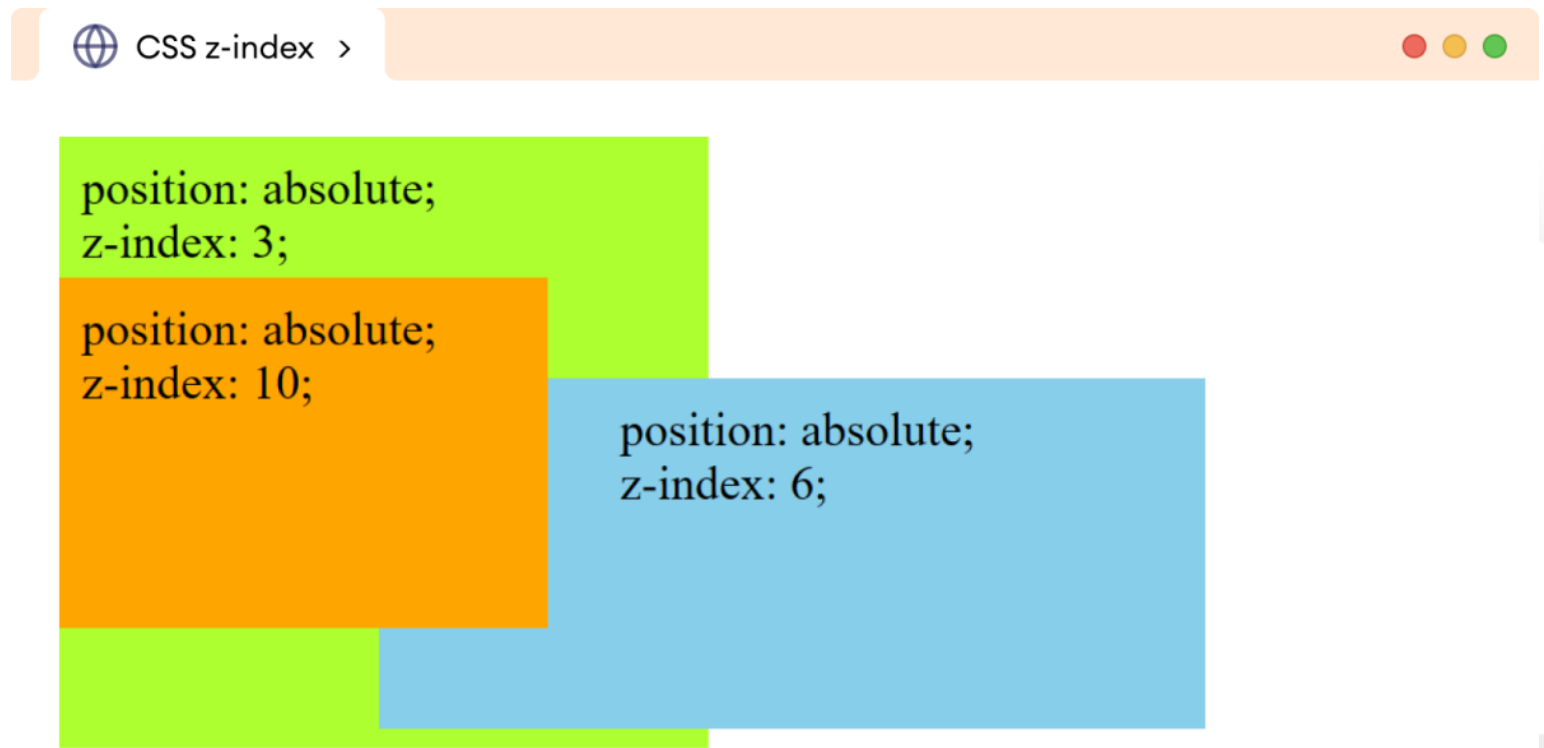
The position property value should be different from the default static value.

The syntax of the z-index property is as follows:

z-index: auto | number | initial | inherit; Here,

- auto: determines the stacking order based on the element's position in the HTML document (default value)
- number: sets the stacking order of an element, negative values are allowed
- initial: sets the property value to the default
- inherit: inherits the property value from the parent

```
div {
  font-size: 24px;
  padding: 12px;
}
div.greenyellow {
  position: absolute;
  top: 0;
  left: 0;
  /* specifying z-index value */
  z-index: 3;
  width: 300px;
  height: 280px;
  background-color: greenyellow;
}
div.orange {
  position: absolute;
  top: 190px;
  left: 0;
  /* specifying z-index value */
  z-index: 10;
  width: 220px;
  height: 150px;
  margin-top: -120px;
  background-color: orange;
}
div.skyblue {
  position: absolute;
  top: 120px;
  left: 160px;
  /* specifying z-index value */
  z-index: 6;
  width: 280px;
  height: 150px;
  padding-left: 120px;
  background-color: skyblue;
}
```



CSS OVERFLOW PROPERTY

The CSS overflow property is used to adjust the content when its size is too big relative to the element box.

The syntax of the overflow property is as follows:

`overflow: visible | hidden | scroll | auto | initial | inherit;`Here,

- `visible`: allows the content to overflow (default value)
- `hidden`: hides the overflowing content
- `scroll`: adds scrollbars to the element box and allows to scroll the content
- `auto`: adds scrollbars only when necessary
- `initial`: sets the property value to the default value
- `inherit`: inherits the property value from the parent element

CSS OVERFLOW: VISIBLE

The visible value of the overflow property allows the content to overflow from the container element. This is the default value of the overflow property.

the content of the p element is not clipped off and overflows from the container div element.

```
div {  
  height: 100px;  
  
  /* default value */  
  overflow: visible;  
  
  background-color: greenyellow;  
  border: 2px solid black;  
}  
  
p {  
  font-size: 24px;  
  margin: 0;  
}
```



CSS Overflow >

“Mostly, when you see programmers, they aren’t doing anything. One of the attractive things about programmers is that you cannot tell whether or not they are working simply by looking at them. Very often they’re sitting there seemingly drinking coffee and gossiping, or just staring into

space. What the programmer is trying to do is get a handle on all the individual and unrelated ideas that are scampering around in his head.”

— Charles M. Strauss

LAYOUTS

Modern CSS has exceptionally powerful layout tooling. We have dedicated systems for layout and we're going to have a high-level look at what we have at our disposal, before digging into more detail of Flexbox and Grid in the next modules.

Understanding the display property

The display property does two things. The first thing it does is determine if the box it is applied to acts as **inline** or **block**.

LAYOUTS

CSS Display Syntax

The commonly used values for the display property are as follows:

- **inline**: allows an element to behave like an inline-level element.
- **block**: allows an element to behave like a block-level element.
- **inline-block**: formats the element as inline-level but also allows to set height/width like block-level element.
- **none**: removes the element from the document leaving no space.
- **flex**: sets the element as a flex container to have a flexible layout of its child elements.
- **grid**: sets the element as a grid container to create complex layouts.

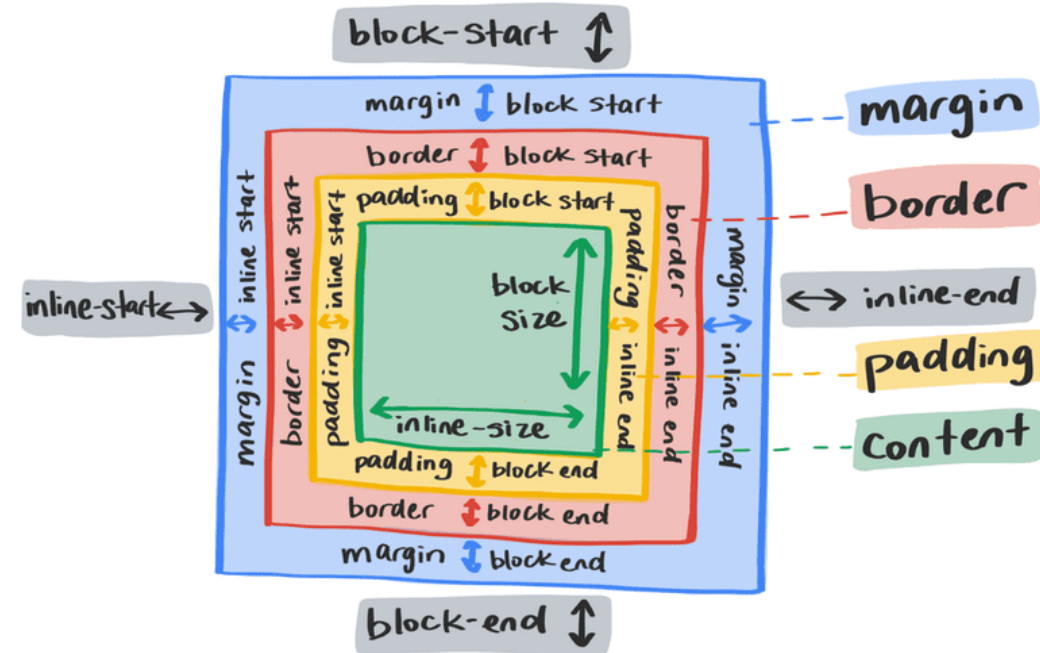
LAYOUTS

Inline

Inline elements behave like words in a sentence. They sit next to each other in the inline direction.

Elements such as `` and ``, which are typically used to style pieces of text within containing elements like a `<p>` (paragraph), are inline by default. They also preserve surrounding whitespace.

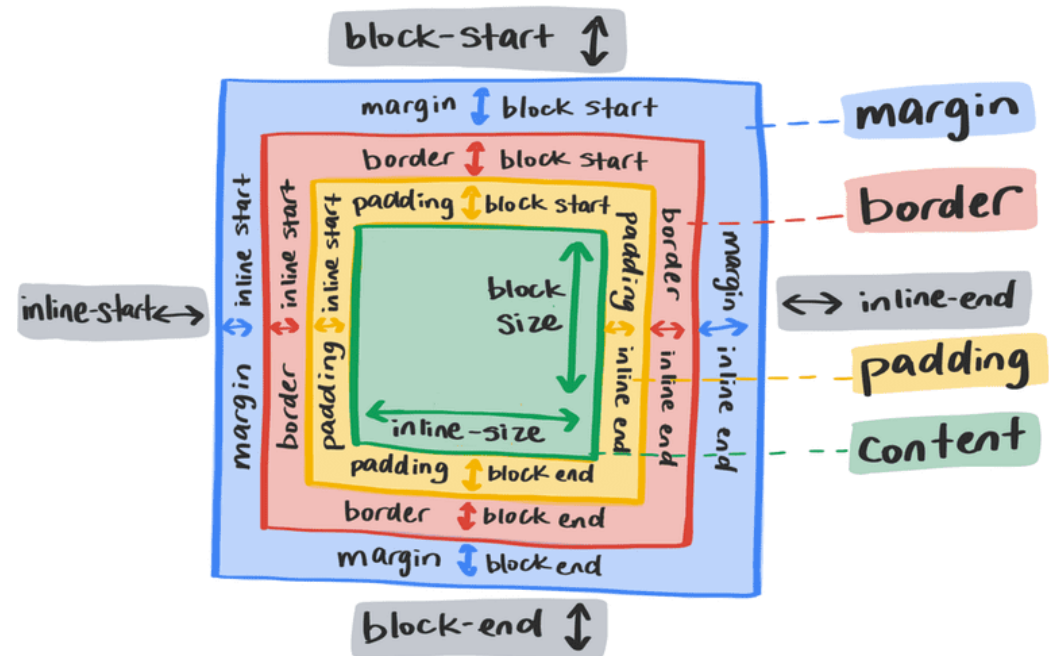
You can't set an explicit width and height on inline elements. Any block level margin and padding will be ignored by the surrounding elements.



LAYOUTS

Block

Block elements don't sit alongside each other. They create a new line for themselves. Unless changed by other CSS code, a block element will expand to the size of the inline dimension, therefore spanning the full width in a horizontal writing mode. The margin on all sides of a block element will be respected.



LAYOUTS

CSS Display Inline

The inline value of the display property allows a block-level element to behave like an inline element. The inline elements occupy only the required width for their content.

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

🌐 CSS Display >

Before display: inline

I am a first div element.

I am a second div element.

After display: inline

I am a first div element.

I am a second div element.

LAYOUTS

She **sells** **seashells**



Inline flow

LAYOUTS

CSS Display Block

The block value of the display property causes an inline element to behave like a block element.

```
span.after {  
  display: block;  
}
```

🌐 CSS Display >

Before display: block

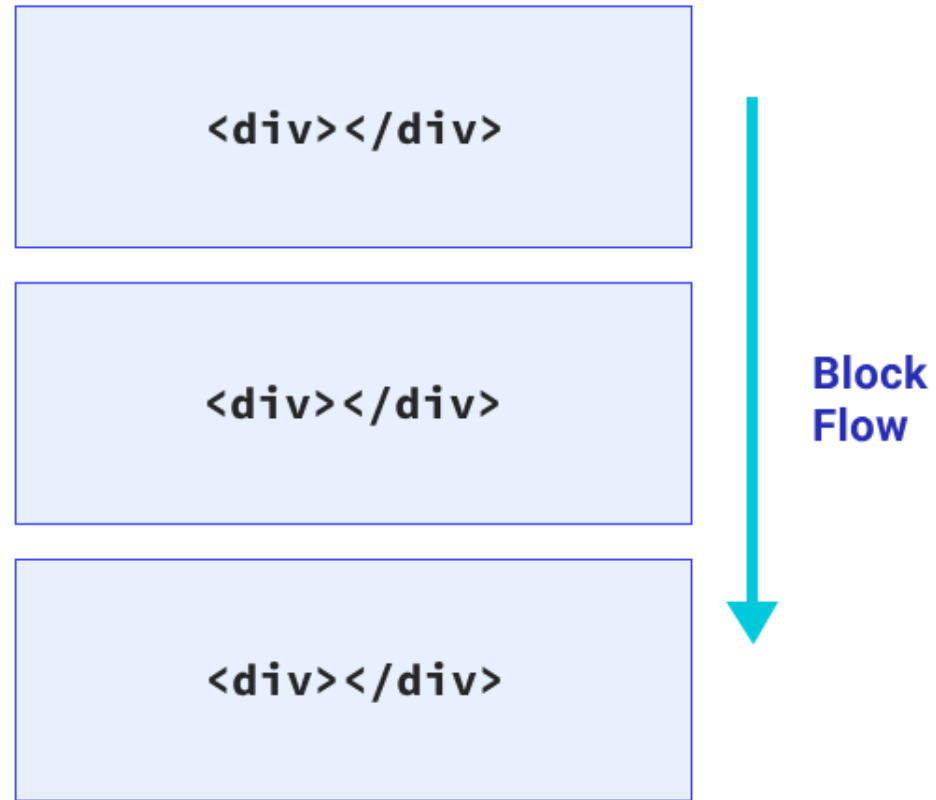
I am a first span element. I am a second span element.

After display: block

I am a first span element.

I am a second span element.

LAYOUTS



LAYOUTS

CSS Display Inline Block

The inline-block value of the display property combines characteristics of inline and block elements.

When an element is set to inline-block,

It flows like an inline element allowing other elements to flow around horizontally.

It accepts the width and height properties. (By default, width and height don't work in inline elements.)

It accepts the padding and margin values. (By default, top and bottom margins don't work for inline elements, and padding also doesn't push other elements away.)

🌐 CSS Display >

Before display: inline-block

In this paragraph, the span element is color in green-yellow color. The width and height don't work, the padding value doesn't push surrounding content away, and the top/bottom margins don't work.

After display: inline-block

In this paragraph, the span element is color in green-yellow color. The width and height work, the padding value push surrounding content away, and the top/bottom margins work.

LAYOUTS

CSS Display Flex

The flex value of the display property allows for efficient alignment and distribution of the space between the child elements within the parent element.

Flex is a powerful tool for creating a flexible and responsible layout. It encompasses a range of associated properties and values that enable the creation of complex layouts.



```
ul.parent {  
  display: flex;  
  background-color: greenyellow;  
  padding: 0px;  
}  
  
li {  
  background: skyblue;  
  border: 1px solid black;  
  padding: 12px;  
  margin: 8px;  
  list-style: none;  
}
```

LAYOUTS

CSS Display None

The none value of the display property is used to hide an element and remove it from the normal flow of the document.

```
h1 {  
  border: 2px solid black;  
  background-color: greenyellow;  
}
```

🌐 CSS Display >

Heading: CSS Display Property

The display: none hides the element and removes the space from the document.

```
h1 {  
  display: none;  
}
```

🌐 CSS Display >

The display: none hides the element and removes the space from the document.

LAYOUTS

CSS Units

CSS units define the size, length and measurement of the web elements

Absolute units

Absolute units are the constant measurements that remain unchanged regardless of the device's screen size.

There are following types of absolute units:

- Pixels (px): Equivalent to one pixel on the screen. For example, width: 100px.
- Inches (in): Correspond to one inch in physical size. For example, height: 1in.
- Centimeters (cm): Equivalent to one centimeter. For example, margin: 2cm.
- Millimeters (mm): Equivalent to one millimeter. For example, padding: 5mm.
- Points (pt): Equal to one point (1/72 of an inch). For example, font-size: 12pt.

Absolute units are used for elements that need to have a specific size, such as buttons, images, and logos.

LAYOUTS

CSS Units

CSS units define the size, length and measurement of the web elements

Relative Units

Relative units are the measurements that change with the screen size or other elements on the page.

There are following types of relative units:

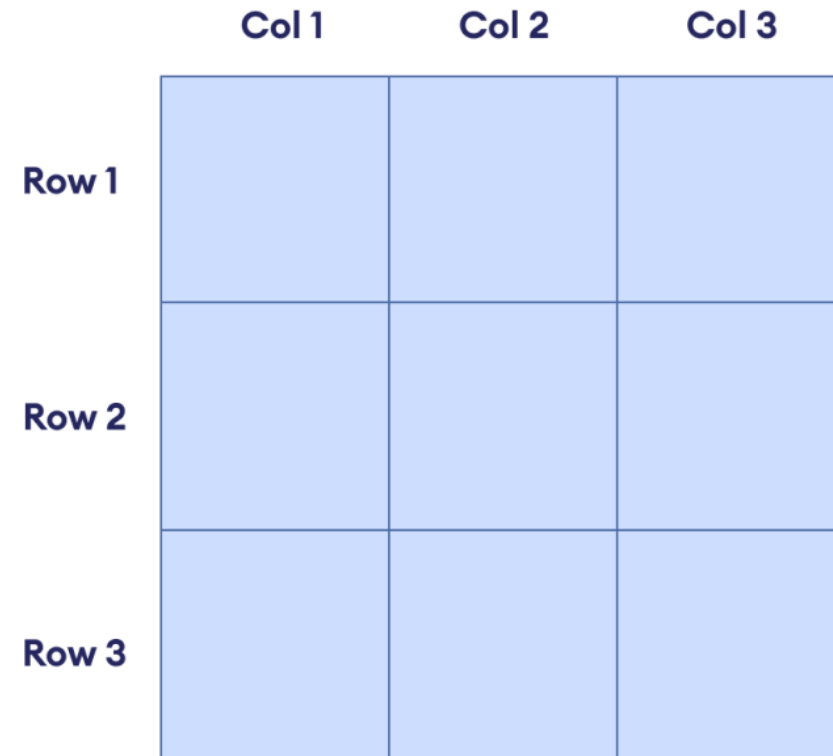
- Ems (em): Relative to the parent element's font size. For example, font-size: 1em scales with the parent's font size.
- If the parent has a font size of 16px, then font-size: 1em for a child equals 16px.
- Rems (rem): Relative to the root element's font size i.e. size set to HTML element. For example, margin: 2rem adjusts based on the root font size.
- Viewport widths (vw): Relative to the viewport width. For example, width: 50vw adapts to 50% of the viewport's width.
- Viewport heights (vh): Relative to the viewport height. For example, height: 100vh takes up the entire viewport (screen) height.
- Percentages (%): Relative to the containing element. For example, padding: 5% scales based on the parent container's size.

Relative units are preferred when we need flexible sizing and responsive design across various devices.

CSS GRID INTRODUCTION

The CSS Grid is a two-dimensional layout system that allows designers and developers to create complex and responsive layouts with ease.

Grid layout creates a grid structure of rows and columns and positions elements within the grid's individual cells.

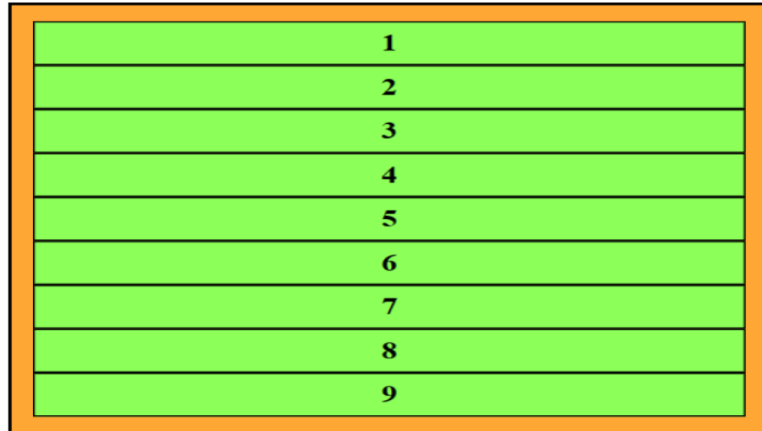


CSS GRID CONTAINER

CSS grid container is a HTML element that serves as a parent element of the grid items.

The grid value of the display property sets the element as a grid container.

🌐 CSS Grid Container >



A browser window titled "CSS Grid Container" displays a grid container. The container is a rectangle with an orange border, containing nine horizontal green bars. Each bar is numbered from 1 to 9, centered within the bar. The bars are stacked vertically, filling the container.

1
2
3
4
5
6
7
8
9

```
div.container {  
  
    /* sets the element as a grid container */  
    display: grid;  
  
    width: 400px;  
    border: 2px solid black;  
    background-color: orange;  
    padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
    border: 1px solid black;  
    background-color: greenyellow;  
    text-align: center;  
    padding: 5px;  
    font-weight: bold;  
}
```

CSS GRID-TEMPLATE-COLUMNS

The grid-template-columns property defines the size and the number of columns within the grid container.

🌐 CSS Grid Container >

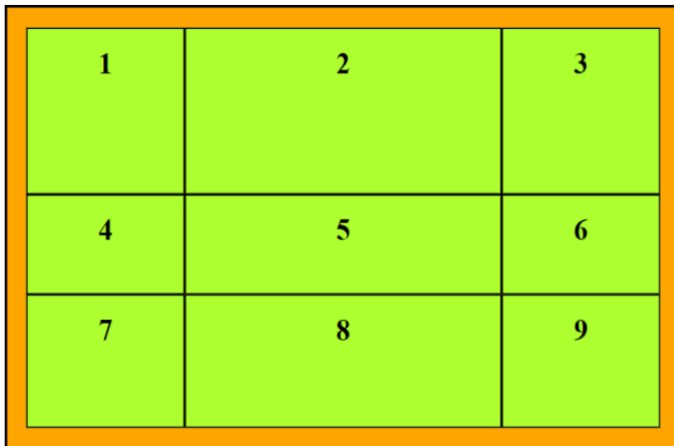
1	2	3
4	5	6
7	8	9

```
div.container {  
  display: grid;  
  
  /* creates a three columns of size 100px, 200px, 100px respectively */  
  grid-template-columns: 100px 200px 100px;  
  
  width: 400px;  
  border: 2px solid black;  
  background-color: orange;  
  padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
  border: 1px solid black;  
  background-color: greenyellow;  
  text-align: center;  
  padding: 10px;  
  font-weight: bold;  
}
```


CSS GRID-TEMPLATE-ROWS

The grid-template-rows property defines the size and number of rows in a grid container.

🌐 CSS Grid Container >



1	2	3
4	5	6
7	8	9

```
div.container {  
  display: grid;  
  grid-template-columns: 100px 200px 100px;  
  
  /* sets three rows of size 100px, 60px, and 80px respectively */  
  grid-template-rows: 100px 60px 80px;  
  
  width: 400px;  
  border: 2px solid black;  
  background-color: orange;  
  padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
  border: 1px solid black;  
  background-color: greenyellow;  
  text-align: center;  
  padding: 10px;  
  font-weight: bold;  
}
```

CSS FRACTIONAL (FR) UNIT

The fractional unit (fr) divides the available space in the grid container into fractions. It distributes available space within the grid container among columns or rows in a flexible and dynamic way.

🌐 CSS Grid Container >

1	2	3
4	5	6
7	8	9

```
div.container {  
  display: grid;  
  
  /* column-1, column-2, column-3 */  
  grid-template-columns: 1fr 2fr 1fr;  
  
  /* row-1, row-2, row-3 */  
  grid-template-rows: 1fr 1fr 2fr;  
  
  width: 400px;  
  border: 2px solid black;  
  background-color: orange;  
  padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
  border: 1px solid black;  
  background-color: greenyellow;  
  text-align: center;  
  padding: 10px;  
  font-weight: bold;  
}
```

MINIMUM AND MAXIMUM GRID TRACK SIZES

The minmax() function defines the minimum and maximum size for the grid track i.e. for a row or column size.

🌐 CSS Grid Container >

1	2	3
4	5	6
The first column track has a minimum size of auto. This means that the minimum size it will take is to fit the content. Its maximum size of 50% will prevent it from getting wider than 50% of the grid container width.	8	9

```
div.container {
  display: grid;

  /* column-1, column-2, column-3 */
  grid-template-columns: minmax(auto, 80%) 1fr 3em;

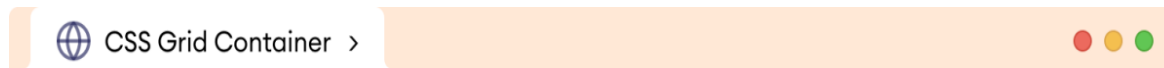
  /* row-1, row-2, row-3 */
  grid-template-rows: 1fr 1fr 3fr;

  width: 400px;
  border: 2px solid black;
  background-color: orange;
  padding: 12px;
}

/* styles all grid items */
div.item {
  border: 1px solid black;
  background-color: greenyellow;
  text-align: center;
  padding: 10px;
  font-weight: bold;
}
```

REPEATING GRID TRACKS

The repeat() function defines the repeating grid tracks. This is useful for grids with items of equal sizes.



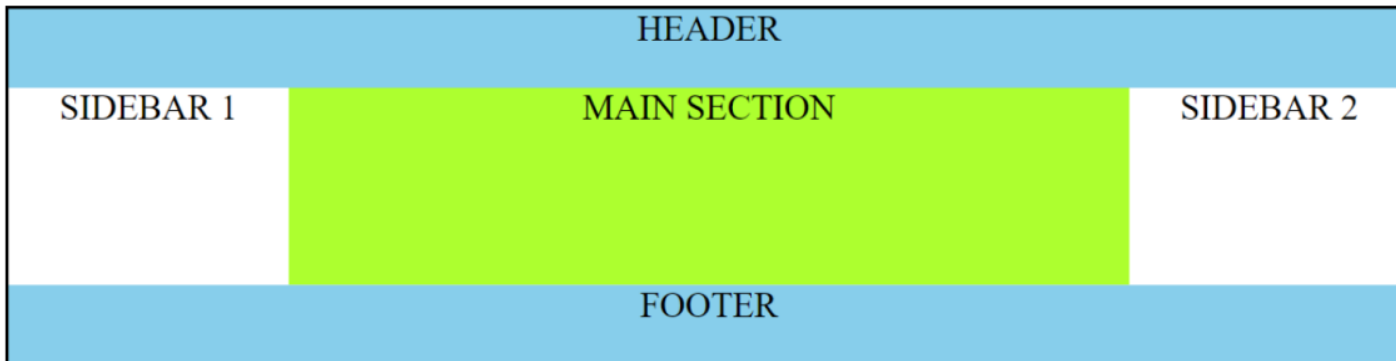
1	2	3
4	5	6
7	8	9

```
div.container {  
  display: grid;  
  
  /* creates 3 columns of 100px each */  
  grid-template-columns: repeat(3, 100px);  
  
  /* creates 3 rows of equal fraction */  
  grid-template-rows: repeat(3, 1fr);  
  
  width: 400px;  
  border: 2px solid black;  
  background-color: orange;  
  padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
  border: 1px solid black;  
  background-color: greenyellow;  
  text-align: center;  
  padding: 10px;  
  font-weight: bold;  
}
```

CSS GRID- TEMPLATE-AREAS

The grid-template-areas property defines the layout of the grid container using named grid areas.

🌐 CSS Grid Container >

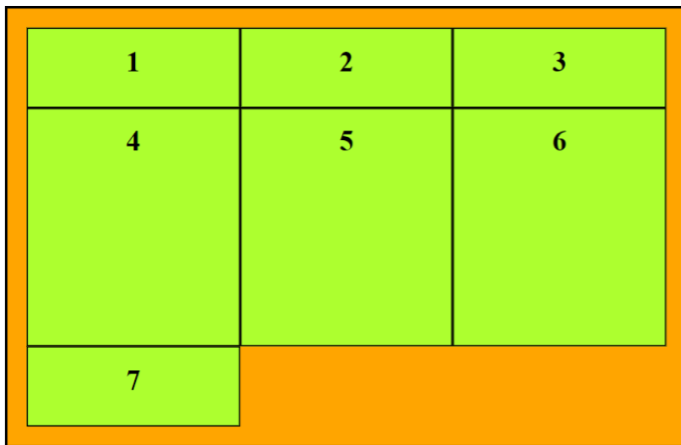


```
1 div.container {
2   display: grid;
3
4   /* defines the columns */
5   grid-template-columns: 1fr 3fr 1fr;
6
7   /* defines the rows */
8   grid-template-rows: 40px 100px 40px;
9
10  /* defines the grid areas placing the grid items */
11  grid-template-areas:
12    "header header header"
13    "sidebar1 main sidebar2"
14    "footer footer footer";
15
16  border: 2px solid black;
17  text-align: center;
18 }
19
20 header {
21   /* places the header within the header grid area */
22   grid-area: header;
23   background-color: skyblue;
24 }
25
26 aside.left {
27   /* places left sidebar within the sidebar1 grid area */
28   grid-area: sidebar1;
29 }
30
31 aside.right {
32   /* places right sidebar within the sidebar2 grid area */
33   grid-area: sidebar2;
34 }
35
36 main {
37   /* places the main content within the main grid area */
38   grid-area: main;
39   background-color: greenyellow;
40 }
41
42 footer {
43   /*places the footer in the footer grid area*/
44   grid-area: footer;
45   background-color: skyblue;
46 }
```

CSS GRID-TEMPLATE PROPERTY

The grid-template is a shorthand property to specify grid-template-rows, grid-template-columns, and grid-template-areas in a single declaration.

🌐 CSS Grid Container >



```
div.container {  
  display: grid;  
  
  /* grid-template-rows / grid-template-columns */  
  grid-template: 50px 150px 50px / 1fr 1fr 1fr;  
  
  width: 400px;  
  border: 2px solid black;  
  background-color: orange;  
  padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
  border: 1px solid black;  
  background-color: greenyellow;  
  text-align: center;  
  padding: 10px;  
  font-weight: bold;  
}
```

CSS GRID-COLUMN-GAP PROPERTY

The grid-column-gap property defines the spacing between columns in a grid container.



CSS Grid Container >

1	2	3
4	5	6
7	8	9

```
div.container {
  display: grid;

  /* defines 3 rows of 80px each */
  grid-template-rows: repeat(3, 80px);

  /* defines 3 columns of equal fraction */
  grid-template-columns: repeat(3, 1fr);

  /* creates a column gap of 20px */
  grid-column-gap: 20px;

  width: 400px;
  border: 2px solid black;
  background-color: orange;
}

/* styles all grid items */
div.item {
  border: 1px solid black;
  background-color: greenyellow;
  text-align: center;
  padding: 30px;
  font-weight: bold;
}
```

CSS GRID-ROW-GAP PROPERTY

The grid-row-gap property defines the spacing between the rows of the grid container.

🌐 CSS Grid Container >

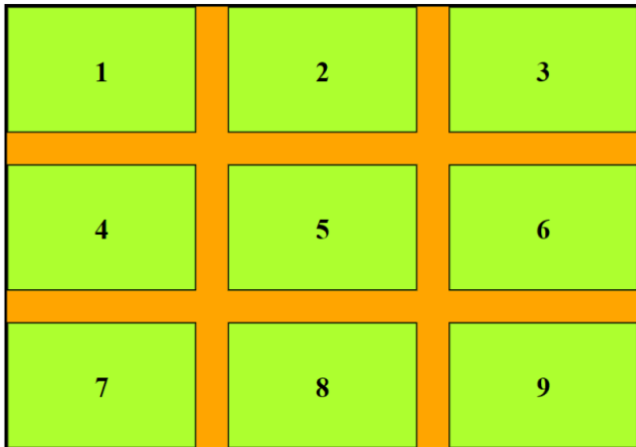
1	2	3
4	5	6
7	8	9

```
div.container {  
  display: grid;  
  
  /* defines three rows of 80px each */  
  grid-template-rows: repeat(3, 80px);  
  
  /* defines three columns of equal fraction */  
  grid-template-columns: repeat(3, 1fr);  
  
  /* creates a row gap of 20px */  
  grid-row-gap: 20px;  
  
  width: 400px;  
  border: 2px solid black;  
  background-color: orange;  
}  
  
/* styles all grid items */  
div.item {  
  border: 1px solid black;  
  background-color: greenyellow;  
  text-align: center;  
  padding: 30px;  
  font-weight: bold;  
}
```


CSS GRID-GAP PROPERTY

The grid-gap is a shorthand property to specify the grid-column-gap and grid-row-gap simultaneously.

🌐 CSS Grid Container >



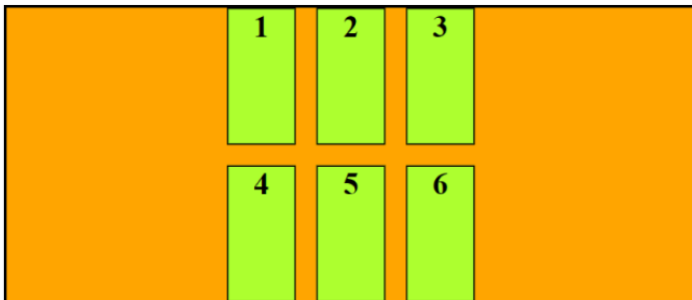
```
div.container {  
  display: grid;  
  
  /* defines three rows of 80px each */  
  grid-template-rows: repeat(3, 80px);  
  
  /* defines three columns of equal fraction */  
  grid-template-columns: repeat(3, 1fr);  
  
  /* creates a row and column gap of 20px */  
  grid-gap: 20px;  
  
  width: 400px;  
  border: 2px solid black;  
  background-color: orange;  
}  
  
/* styles all grid items */  
div.item {  
  border: 1px solid black;  
  background-color: greenyellow;  
  text-align: center;  
  padding: 30px;  
  font-weight: bold;  
}
```

CSS JUSTIFY-CONTENT PROPERTY

The justify-content property controls the horizontal alignment of the grid within the grid container.

The possible values of the justify-content property are start, end, center, space-around, space-between, and space-evenly.

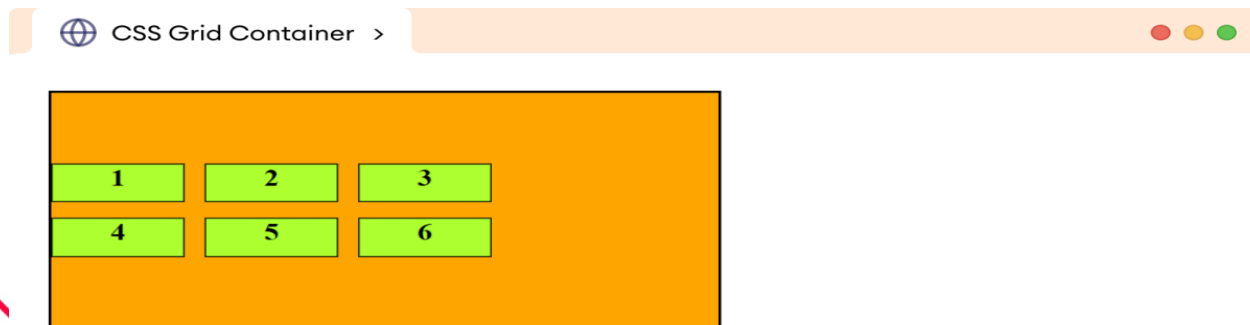
🌐 CSS Grid Container >



```
div.container {  
  display: grid;  
  
  /* defines rows and columns */  
  grid-template-rows: repeat(2, 80px);  
  grid-template-columns: repeat(3, 40px);  
  
  /* aligns the entire grid to center in the grid container*/  
  justify-content: center;  
  
  width: 400px;  
  gap: 12px;  
  border: 2px solid black;  
  background-color: orange;  
}  
  
/* styles all grid items */  
div.item {  
  text-align: center;  
  border: 1px solid black;  
  background-color: greenyellow;  
  font-weight: bold;  
}
```

CSS ALIGN-CONTENT PROPERTY

The align-content property controls the alignment of the grid within the grid container along the horizontal direction. The possible values of the align-content property are start, end, center, space-around, space-between, and space-evenly.

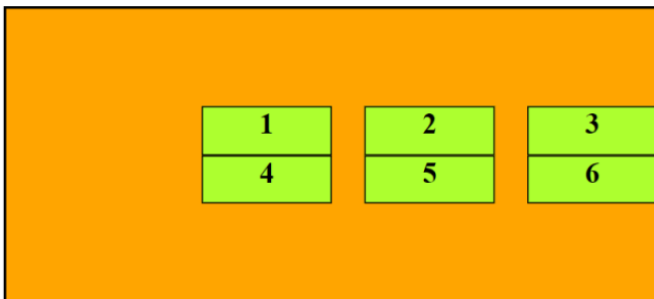


```
div.container {  
  display: grid;  
  height: 180px;  
  
  /* defines rows and columns */  
  grid-template-rows: repeat(2, 30px);  
  grid-template-columns: repeat(3, 80px);  
  
  /* aligns the grid vertically to center of grid container */  
  align-content: center;  
  
  width: 400px;  
  gap: 12px;  
  border: 2px solid black;  
  background-color: orange;  
}  
  
/* styles all grid items */  
div.item {  
  text-align: center;  
  border: 1px solid black;  
  background-color: greenyellow;  
  font-weight: bold;  
}
```

CSS PLACE-CONTENT PROPERTY

The place-content property is a shorthand property to specify the justify-content and align-content in a single declaration.

🌐 CSS Grid Container >

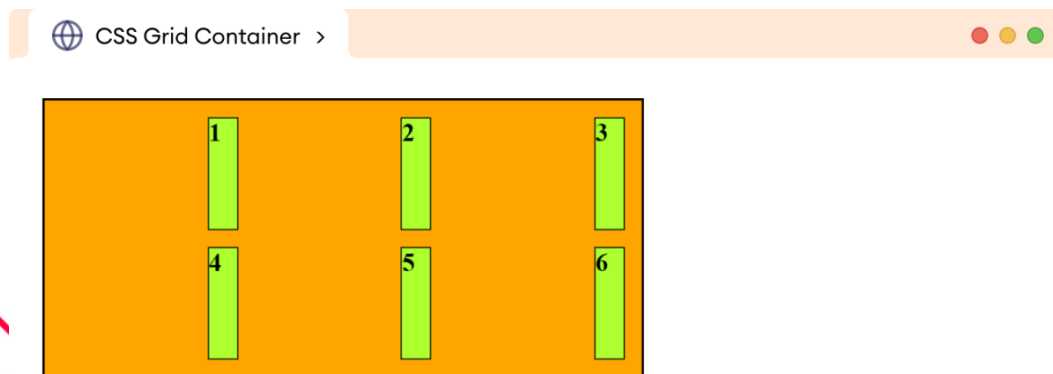


```
div.container {  
  display: grid;  
  width: 400px;  
  height: 180px;  
  
  /* defines rows and columns */  
  grid-template-rows: repeat(2, 30px);  
  grid-template-columns: repeat(3, 80px);  
  
  /* sets align-content to center and justify-content to end of grid container*/  
  place-content: center end;  
  
  column-gap: 20px;  
  border: 2px solid black;  
  background-color: orange;  
}  
  
/* styles all grid items */  
div.item {  
  text-align: center;  
  border: 1px solid black;  
  background-color: greenyellow;  
  font-weight: bold;  
}
```

CSS JUSTIFY-ITEMS PROPERTY

The justify-items property controls the horizontal alignment of the grid items within the respective grid cells.

The common values for the justify-items property are start, end, center, and stretch.



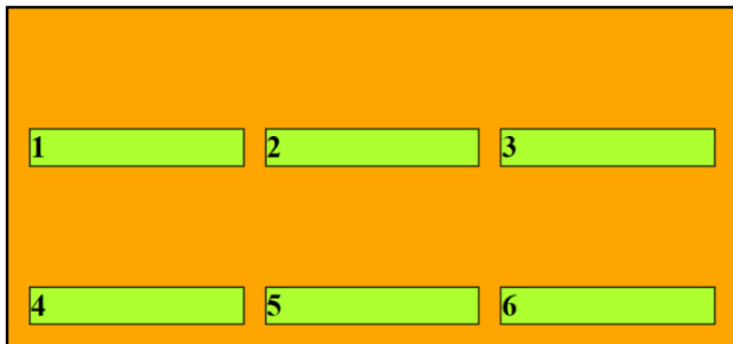
```
div.container {  
  display: grid;  
  
  /* defines rows and columns */  
  grid-template-rows: repeat(2, 80px);  
  grid-template-columns: repeat(3, 1fr);  
  
  /* aligns the grid items at the end of grid cells */  
  justify-items: end;  
  
  width: 400px;  
  gap: 12px;  
  border: 2px solid black;  
  background-color: orange;  
  padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
  width: 20px;  
  border: 1px solid black;  
  background-color: greenyellow;  
  font-weight: bold;  
}
```

CSS ALIGN-ITEMS PROPERTY

The align-items property controls the vertical alignment of the grid items within the respective grid cells.

The common values for the align-items property are start, end, center, and stretch.

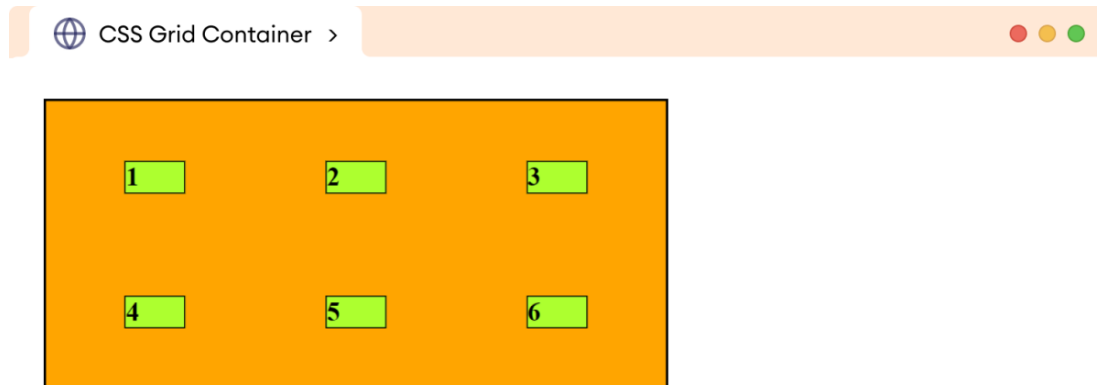
🌐 CSS Grid Container >



```
div.container {  
  display: grid;  
  
  /* defines rows and columns */  
  grid-template-rows: repeat(2, 80px);  
  grid-template-columns: repeat(3, 1fr);  
  
  /* aligns the grid items at the bottom of grid cells */  
  align-items: end;  
  
  width: 400px;  
  gap: 12px;  
  border: 2px solid black;  
  background-color: orange;  
  padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
  border: 1px solid black;  
  background-color: greenyellow;  
  font-weight: bold;  
}
```

CSS PLACE-ITEMS

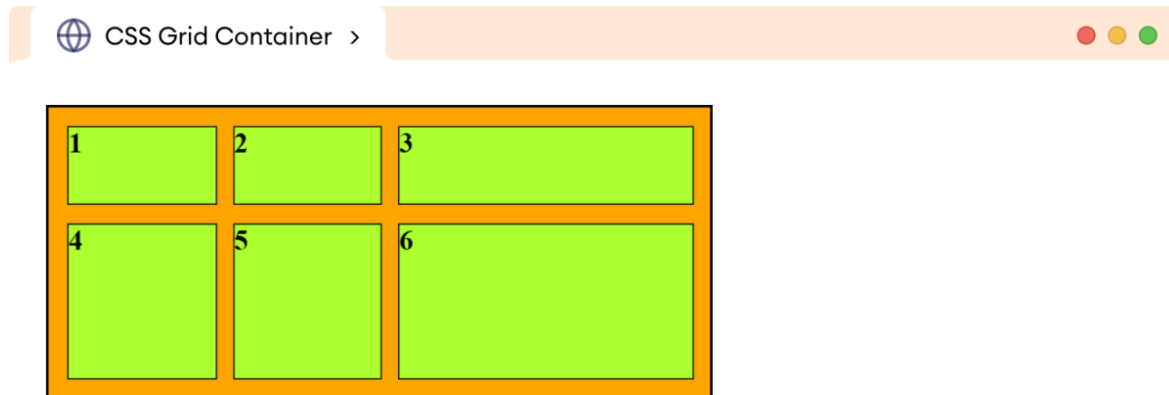
The place-items is a shorthand property to specify the align-items and justify-items property in a single declaration.



```
div.container {  
  display: grid;  
  
  /* defines rows and columns */  
  grid-template-rows: repeat(2, 80px);  
  grid-template-columns: repeat(3, 1fr);  
  
  /* sets align-items and justify-items to the center of grid cell */  
  place-items: center;  
  
  width: 400px;  
  gap: 12px;  
  border: 2px solid black;  
  background-color: orange;  
  padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
  width: 40px;  
  border: 1px solid black;  
  background-color: greenyellow;  
  font-weight: bold;  
}
```

CSS GRID PROPERTY

The grid is a shorthand property to specify grid-template-rows and grid-template-columns properties in a single declaration.



```
div.container {  
  display: grid;  
  
  /* grid-template-rows / grid-template-columns */  
  grid: 50px 100px / 1fr 1fr 2fr;  
  
  width: 400px;  
  gap: 12px;  
  border: 2px solid black;  
  background-color: orange;  
  padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
  width: 100%;  
  border: 1px solid black;  
  background-color: greenyellow;  
  font-weight: bold;  
}
```


CSS GRID ITEMS

Grid items are the direct child elements of the flex container.

Grid Container

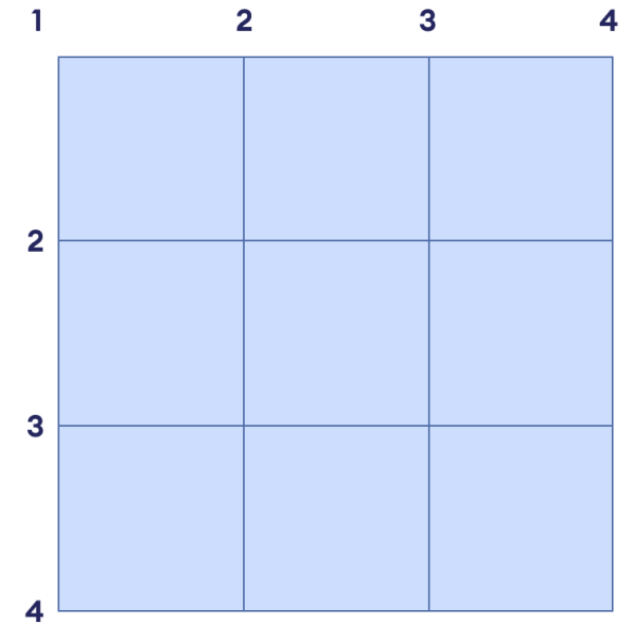


GRID LINE NUMBERS

CSS grid consists of a set of horizontal and vertical lines, forming rows and columns. These lines are referred to by their line numbers.

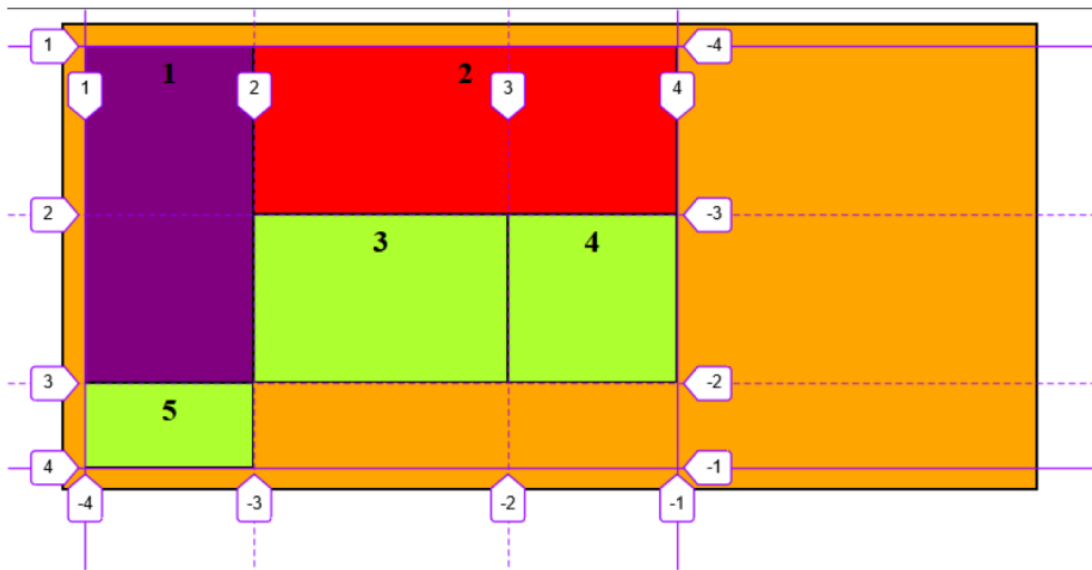
The grid items can be placed with the help of grid line numbers. The following properties control the placement of grid items using grid line numbers.

- `grid-column-start`: defines the starting column line number
- `grid-column-end`: defines the ending column line number
- `grid-row-start`: defines the starting row line number
- `grid-row-end`: defines the ending row line number



PLACING GRID ITEMS

🌐 CSS Grid Items >



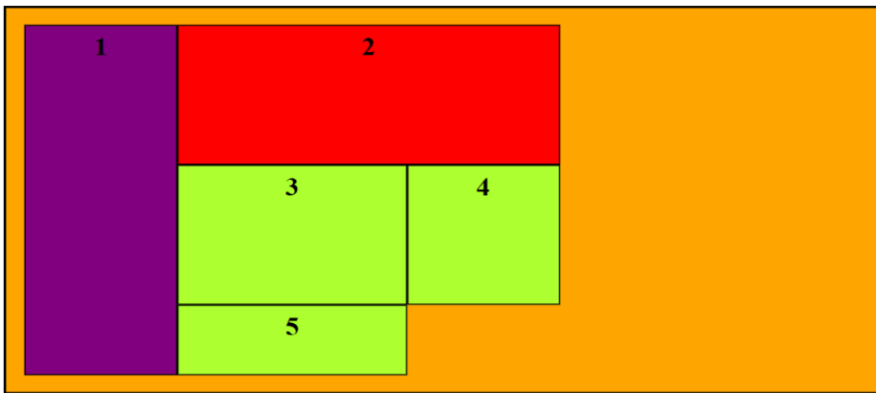
```
div.container {  
  /* sets the element as a grid container */  
  display: grid;  
  
  /* defines the grid of 3 rows and three columns */  
  grid-template: 100px 100px 50px / 100px 150px 100px;  
  
  width: 550px;  
  border: 2px solid black;  
  background-color: orange;  
  padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
  border: 1px solid black;  
  background-color: greenyellow;  
  text-align: center;  
  padding: 5px;  
  font-weight: bold;  
}  
  
/* places the first grid item */  
div.item-1 {  
  grid-row-start: 1;  
  grid-row-end: 3;  
  background-color: purple;  
}  
  
/* places the second grid item */  
div.item-2 {  
  grid-column-start: 2;  
  grid-column-end: 4;  
  background-color: red;  
}
```

CSS GRID-ROW AND GRID-COLUMN PROPERTIES

The grid-row and grid-column are the shorthand properties to specify the grid line numbers.

- grid-row: defines the grid-row-start and grid-row-end properties
- grid-column: defines the grid-column-start and grid-column-end properties

CSS Grid Items >

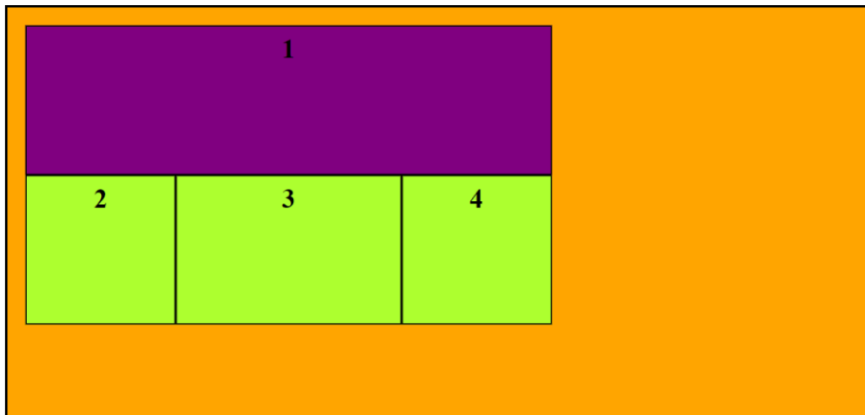


```
div.container {  
  /* sets the element as a grid container */  
  display: grid;  
  
  /* defines the grid of 3 rows and three columns */  
  grid-template: 100px 100px 50px / 100px 150px 100px;  
  
  width: 550px;  
  border: 2px solid black;  
  background-color: orange;  
  padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
  border: 1px solid black;  
  background-color: greenyellow;  
  text-align: center;  
  padding: 5px;  
  font-weight: bold;  
}  
  
/* places the first grid item */  
div.item-1 {  
  grid-row: 1/4;  
  background-color: purple;  
}  
  
/* places the second grid item */  
div.item-2 {  
  grid-column: 2/4;  
  background-color: red;  
}
```

SPANNING THE ROWS AND COLUMNS

The **span** keyword is used to extend rows and columns by a specified number.

🌐 CSS Grid Items >

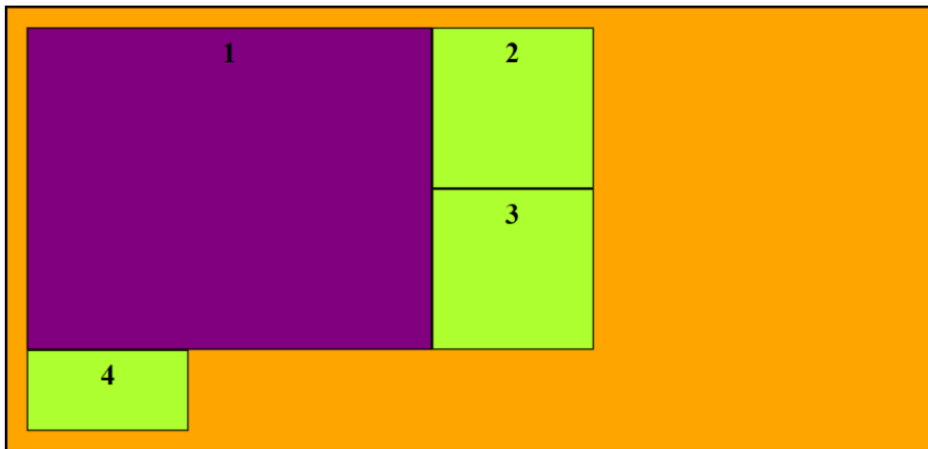


```
div.container {  
  /* sets the element as a grid container */  
  display: grid;  
  
  /* defines the grid of 3 rows and three columns */  
  grid-template: 100px 100px 50px / 100px 150px 100px;  
  
  width: 550px;  
  border: 2px solid black;  
  background-color: orange;  
  padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
  border: 1px solid black;  
  background-color: greenyellow;  
  text-align: center;  
  padding: 5px;  
  font-weight: bold;  
}  
  
/* places the first grid item */  
div.item-1 {  
  grid-column: 1 / span 3;  
  background-color: purple;  
}
```

CSS GRID-AREA PROPERTY

The **grid-area** property specifies both the row and column positions of a grid item in a single declaration.

🌐 CSS Grid Items >

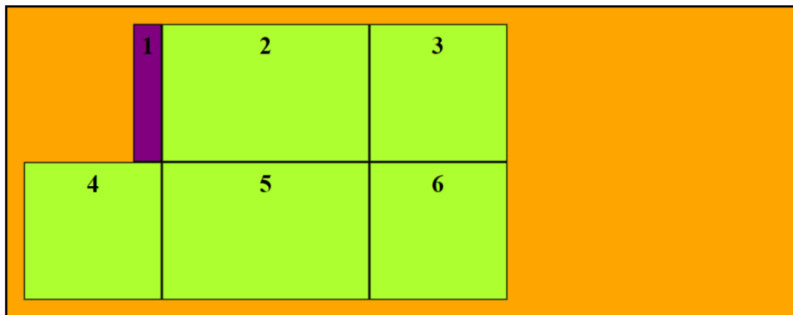


```
div.container {  
  /* sets the element as a grid container */  
  display: grid;  
  
  /* defines the grid of 3 rows and three columns */  
  grid-template: 100px 100px 50px / 100px 150px 100px;  
  
  width: 550px;  
  border: 2px solid black;  
  background-color: orange;  
  padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
  border: 1px solid black;  
  background-color: greenyellow;  
  text-align: center;  
  padding: 5px;  
  font-weight: bold;  
}  
  
/* places the first grid item */  
div.item-1 {  
  grid-area: 1 / 1 / 3/ 3;  
  background-color: purple;  
}
```

CSS JUSTIFY-SELF PROPERTY

The **justify-self** property is used to control the horizontal alignment of a grid item within its grid cell. The possible values of the justify-self property are **start**, **end**, **center**, and **stretch**.

🌐 CSS Grid Items >

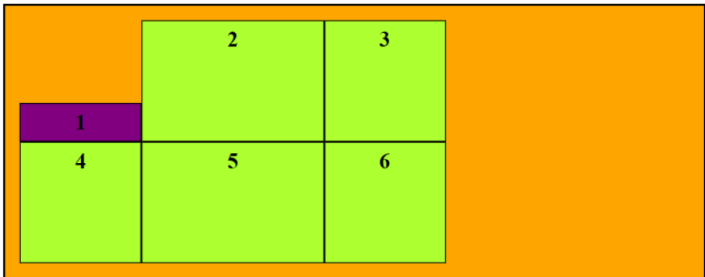


```
div.container {  
  /* sets the element as a grid container */  
  display: grid;  
  
  /* defines the grid of 2 rows and three columns */  
  grid-template: 100px 100px / 100px 150px 100px;  
  
  width: 550px;  
  border: 2px solid black;  
  background-color: orange;  
  padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
  border: 1px solid black;  
  background-color: greenyellow;  
  text-align: center;  
  padding: 5px;  
  font-weight: bold;  
}  
  
/* aligns the content to the end of grid cell */  
div.item-1 {  
  justify-self: end;  
  background-color: purple;  
}
```

CSS ALIGN-SELF PROPERTY

The **align-self** property is used to control the vertical alignment of a grid item within its grid cell. The possible values of the align-self property are **start**, **end**, **center**, and **stretch**.

🌐 CSS Grid Items >

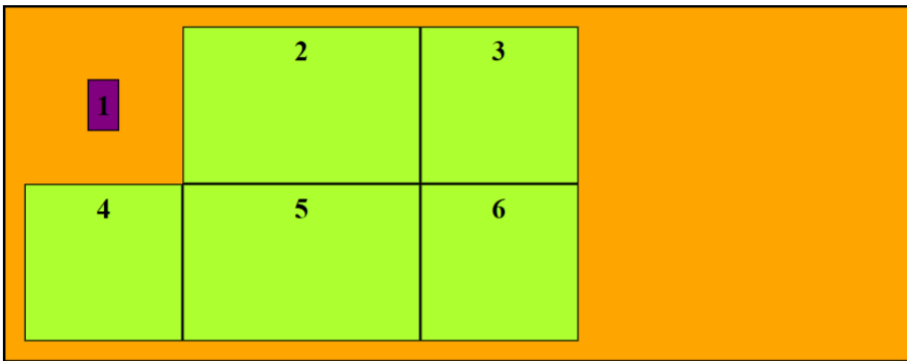


```
div.container {  
  /* sets the element as a grid container */  
  display: grid;  
  
  /* defines the grid of 2 rows and three columns */  
  grid-template: 100px 100px / 100px 150px 100px;  
  
  width: 550px;  
  border: 2px solid black;  
  background-color: orange;  
  padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
  border: 1px solid black;  
  background-color: greenyellow;  
  text-align: center;  
  padding: 5px;  
  font-weight: bold;  
}  
  
/* aligns the content to the end of grid cell */  
div.item-1 {  
  align-self: end;  
  background-color: purple;  
}
```


CSS PLACE-SELF PROPERTY

The **place-self** property is a shorthand property to specify the **justify-self** and **align-self** property in a single declaration

🌐 CSS Grid Items >



```
div.container {  
  /* sets the element as a grid container */  
  display: grid;  
  
  /* defines the grid of 2 rows and three columns */  
  grid-template: 100px 100px / 100px 150px 100px;  
  
  width: 550px;  
  border: 2px solid black;  
  background-color: orange;  
  padding: 12px;  
}  
  
/* styles all grid items */  
div.item {  
  border: 1px solid black;  
  background-color: greenyellow;  
  text-align: center;  
  padding: 5px;  
  font-weight: bold;  
}  
  
/* aligns the content to the end of grid cell */  
div.item-1 {  
  place-self: center center;  
  background-color: purple;  
}
```



THANKS

AMIT