

Positioning

by Sophia



WHAT'S COVERED

In this lesson, you will learn about the different concepts and CSS properties that impact an element's positioning. You will begin with the positioning property and its different values that impact how an element is arranged on a page. You will learn about the CSS `display` property and how inline and block values differ from each other. You will learn about the CSS `float` property, along with its counterpart property `clear`. You will learn about two popular approaches to organizing and arranging content using flexbox and the CSS grid structure. Lastly, you will see an example of using a grid layout to provide a responsive layout for an entire page.

Specifically, this lesson will cover the following:


1. Positioning
2. Inline Versus Block
3. Float
4. Flexbox
5. Grid View
6. Grid for Page Layout

1. Positioning

Positioning elements around a webpage and creating the desired layout often require the use of various techniques at different levels of the HTML and CSS structure. We need to be able to control individual element positions as well as position entire sections of content. We need to be able to create multiple columns of content whenever needed, and those content sections need to be reorganized appropriately for smaller screens. All of this requires an understanding of the different ways we can use HTML structures and CSS styles to control the positioning of elements and sections of content.

To start, each element contains a CSS property called **position**, which accepts either **static**, **relative**, **fixed**, **absolute**, or **sticky**.

Value	Description
static	(Default) Elements are positioned relative to their location within the code. The top, bottom, left, and right properties are ignored.
relative	The element is positioned relative to its location within the code. The top or bottom, right or left properties will move the element further from its relative position.
fixed	This positions the element relative to the viewport and uses top or bottom and left or right in pixels to position the element. Does not move when the user scrolls. This can cause the fixed element to appear on top of other elements as the user scrolls the page.
absolute	This positions the element relative to its nearest positioned parent and uses top or bottom and left or right to position the element. Moves with the content when the user scrolls.
sticky	<p>This positions an element based on the user's scroll position. It allows an element to scroll with its surrounding content but then stick to the screen when the user scrolls beyond the element.</p> <p>A sticky element changes between relative and fixed based on the scroll position of the page. When in its default position, the element is positioned relative to the other elements. As the user begins to scroll, the position is changed to fixed, which could cover other elements in the page as the user scrolls, keeping the sticky element visible.</p>


LEARN MORE

To learn more and see examples of these position properties in action, visit the [w3schools.com](#) page on **CSS positioning**.

TERMS TO KNOW

Position

A CSS property that determines how a browser positions an element on the page.

Static

A value of the position property that positions an element where it exists in the code and ignores the top, left, right, and bottom properties.

Relative

A value of the position property that positions an element relative to its location in the code but also can be adjusted further by using either the top or bottom and left or right properties.

Fixed

A value of the position property that positions an element relative to the viewport and uses the top or bottom, left or right, properties, does not move when the user scrolls.

Absolute

A value of the position property that positions an element based solely on the top left corner of the page's content and uses the top or bottom and left or right properties to control its position, wherein the element scrolls with the surrounding content.

Sticky

A value of the position property that positions an element within a container and will remain within view of the container once the user scrolls past the element.

2. Inline Versus Block

The first aspect of controlling the position of an element is understanding the **CSS display property**. When an element's display property is set to **inline**, the content will automatically position the content as if it were text being added to the middle of a paragraph. An inline element, such as an anchor tag for creating a hyperlink, would not break the paragraph into two, but it would add the hyperlink as part of the paragraph.

If an element with its display property set to **block** was added to a paragraph, such as a `<div>`, then it would break the paragraph into two and separate them. Block elements by default want to take up the full width of space available and will only be as tall as the content inside.

⇒ **EXAMPLE** Consider an `<h1>` tag for example. An `</h1>` tag is used to create a heading on a webpage. Content outside of the `<h1>` element is, by default, on a separate line from the headings. That is because the `block display` property takes up the entire breadth of the line.

```
<body>
```

```
<h1>This is display: block</h1>
```

```
<p>This paragraph is a display:  
block element. However,
```

```
<span>this is a display:
```

```
inline</span> so is this
```

```
<a href="#">hyperlink</a> which  
flows just like a normal line of text.
```

Block elements can have their height and width properties overridden. Elements can also have their display setting changed. You can change an inline element such as an anchor tag into a block element, and the other way around, by simply overriding its display property.



TERMS TO KNOW

Display Property

The CSS property that controls how an element is displayed on a webpage.

Inline

A value of the display property that allows an element to flow with the text, as if it were part of the paragraph.

Block

A value of the display property that allows an element to be resized using height and width and be repositioned within its relative position.

3. Float

The CSS **float** property controls how an element floats within the content. The possible values are left, right, none, and inherit. When the float property is given the value of left or right, this indicates which side of the element's container the element will move to. Additionally, the content will be allowed to wrap around the space of the floating element without overlapping.

⇒ **EXAMPLE** The following is an example of an image with the `float right` property applied:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent lupta...



Float with a value of none will simply position the element where it occurs in the HTML code. Block elements that are siblings of each other can all be given the same float value and will stack next to each other, side by side. However, just because we float an element to one side does not mean that we always want content beside it as in the example above. This is where the **clear** property comes into the picture. The CSS clear property accepts the values of left, right, both, none, and inherit. When you want to move content from the side of the floating element and force it below, such as the paragraph in the example above, you can add the clear property to the paragraph (the content you want to move) and give it the same value as its float property. To move the paragraph below the image, the image is set to float right and clear right.

⇒ **EXAMPLE** The CSS code

```
img {  
  float: right;  
}  
  
p {  
  clear: right;  
}
```

⇒ **EXAMPLE** The output



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praese...



HINT

Remember to match the float direction with the clear direction, unless you are using float none.



TERMS TO KNOW

Float

A CSS property that allows a block-level element to float either to the left or to right of the page or its container.

Clear

A CSS property, used in conjunction with float, given to the neighboring element of a floating element, which forces the surrounding content to be pushed below the floating element.

4. Flexbox

Although flexbox was discussed in a previous challenge, we will dive deeper into flexbox properties related to the container as well as the properties of the flex items. Remember that flexbox is applied to a container element and its direct children automatically become the flex items. This is accomplished by setting the element's display property to "flex." Additional CSS properties can be given to change the behavior of the flex container and how it organizes the flex items. Furthermore, each flex child can be given additional properties, which will impact how that specific item behaves.

The following table lists additional properties that can be modified to change a container's behavior.

Property Name	Values	Description
flex-direction	column row	This determines how elements are lined up across the container. Column lines the items up vertically, one on top of the other

	column-reverse row-reverse	starting at the top. Row lines the items up horizontally, side by side, starting on the left. The reverse options invert the sequence of items and their origin.
flex-wrap	nowrap wrap wrap-reverse	This causes elements to break into the next line or column (wrap) when there is not enough space for the items. Nowrap causes the items to shrink as needed in order to fit all items on the same line or column. Wrap-reverse changes the origin of the items to the bottom left corner and wrap them up to the next line.
flex-flow	flex-flow	This is a shorthand property that combines the direction and wrap options.
justify-content	center flex-start flex-end space-around space-between	This is used to align the items horizontally within the container. Center positions the items in the middle of the container's horizontal space. Flex-start aligns the list of items starting from the top. Flex-end aligns the list of items starting from the bottom. Space-around positions the items with a space between each item and around the items. Space-between only positions the items with a space between them; the left- and right-most items are flush with the container.
align-items	center flex-start flex-end stretch baseline	Align-items performs the same operations as justify-content, but it handles vertical alignment. Center will align the items in the middle of the container's vertical space. Flex-start aligns the items to the left. Flex-end aligns the items to the right. Stretch will cause the items to grow vertically in order to fill all of the vertical space of the container. Baseline allows items of different heights to be aligned based on the baseline of the content within the item, such as text.
align-content	space-between space-around stretch center flex-start flex-end	Align-content is used when flex-wrap is turned on, and it allows you to control the vertical alignment and spacing of items that are wrapped.

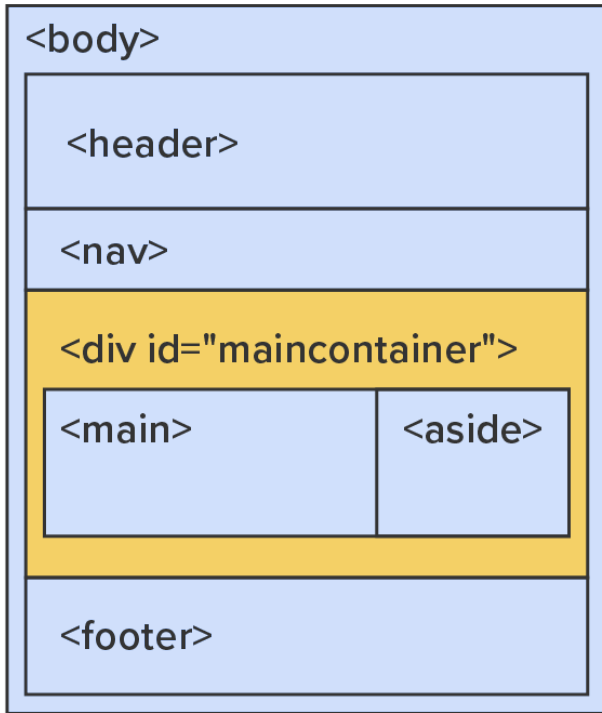


KEY CONCEPT

Much of these flexbox properties can be applied together to create different orientations and effects. However, keep in mind that whenever you use any of the reverse options, they can also reverse the effects of additional properties.

Flex containers are often used for organizing and arranging smaller items such as hyperlinks, images, and image thumbnails. However, flex containers are also used on section containers to rearrange entire sections of content.

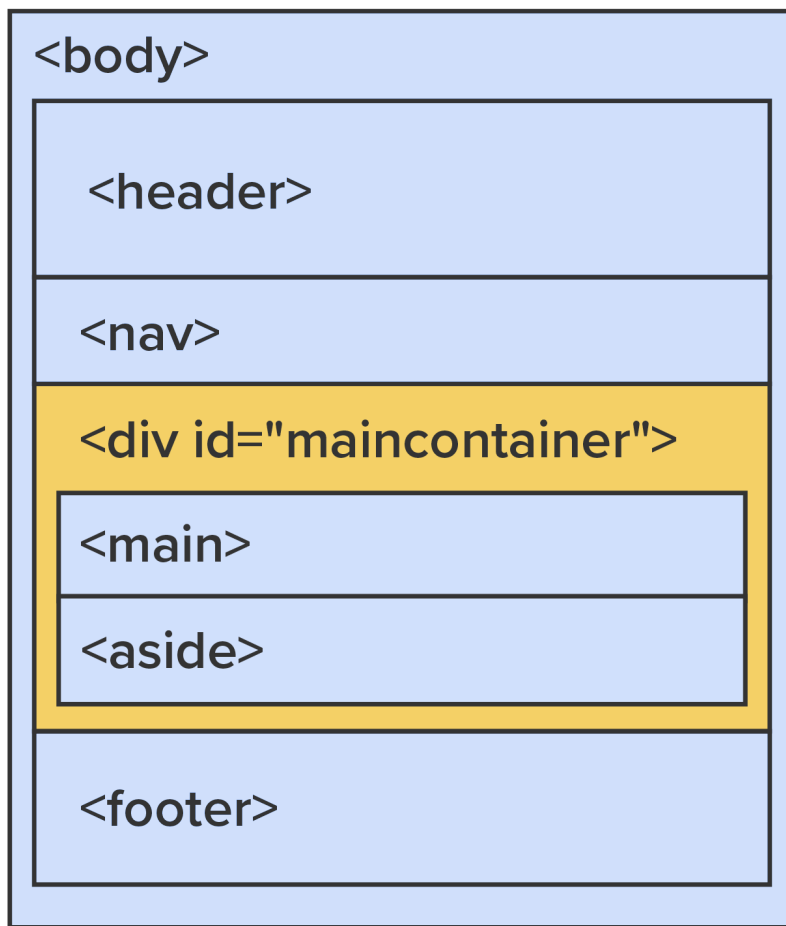
⇒ EXAMPLE Let us revisit a previous example, where a body section contained two semantic tags that needed to be side by side, the main and aside.



HTML source code:

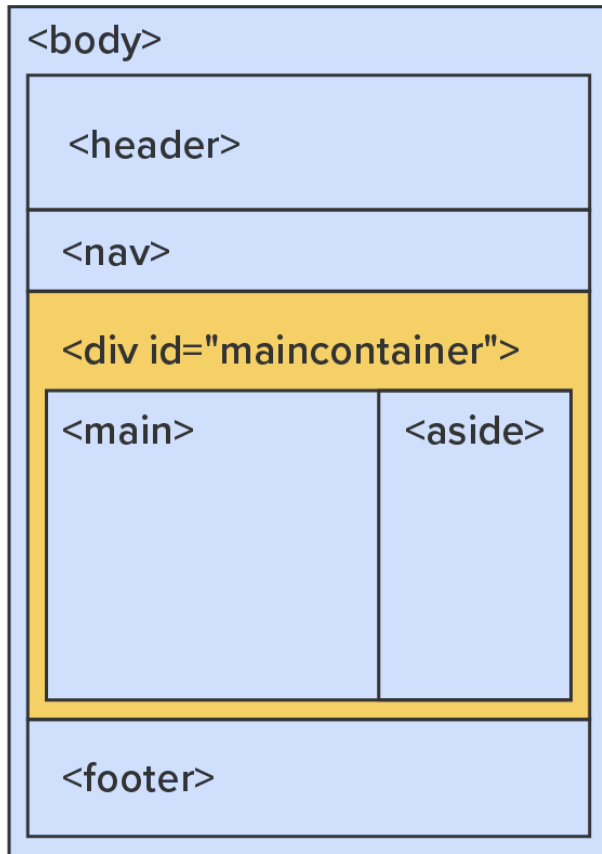
```
<body>
  <header></header>
  <nav></nav>
  <div id="maincontainer">
    <main></main>
    <aside></aside>
  </div>
  <footer></footer>
</body>
```

⇒ EXAMPLE Without using flexbox, the main and aside would still stack on top of each other even though they are contained within the div, like so:



While this may be ideal when designing for smaller screens, it is not ideal for full-size screens.

⇒ **EXAMPLE** Since flex containers default to `flex-direction: row`, simply changing the display property of the `maincontainer` div will position `main` and `aside` side by side.



HTML source code:

```
<body>
  <header></header>
  <nav></nav>
  <div id="maincontainer">
    <main></main>
    <aside></aside>
  </div>
  <footer></footer>
</body>
```

CSS source code:

```
#maincontainer {
  display: flex;
}
```



BIG IDEA

Using something like flexbox to easily reorient items, whether they are small elements or whole sections, provides flexibility for different screen sizes. Remember that responsive web design is designing a website to function well and look good regardless of screen size. Furthermore, remember that we can use a CSS feature called media queries to detect the viewport size and override the flex-direction, alignments, and wrap properties as needed. In the example above, when the screen is too narrow, the media query will give the flex-direction property the value of column for the maincontainer.



TERMS TO KNOW

Flex-Direction

A property of a flex container that changes the orientation of the items from a row of items or a column of items.

Flex-Wrap

A property of a flex container that allows flex items to wrap down to the next row or column when there is not enough space.

Flex-Flow

A shorthand property for both flex-direction and flex-wrap.

Justify-Content

A property of a flex container that controls the horizontal alignment of flex items.

Align-Items

A property of a flex container that controls the vertical alignment of flex items.

Align-Content

A property of a flex container that works with flex-wrap to control the spacing of the rows or columns of wrapped items.

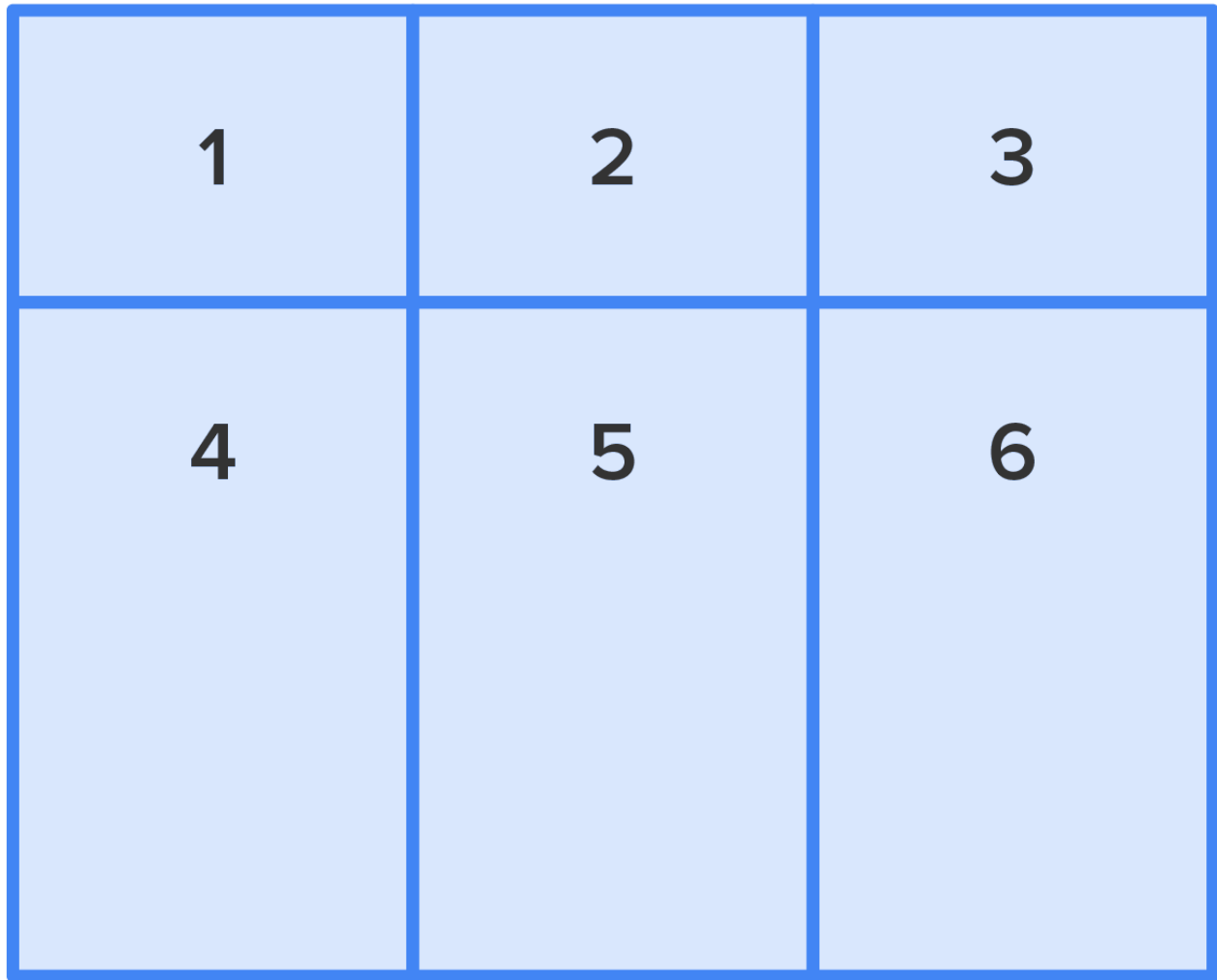
5. Grid View

Another option for organizing and controlling the layout of content is to use the CSS **grid** display property. Similar to flexbox mentioned previously, the CSS grid layout uses a grid container and grid items. The difference is that the grid was designed to keep everything in a grid-type layout and is slightly more rigid than flexbox. As such, you get to decide the number of rows and columns and their sizes, using the **grid-template-rows** and **grid-template-columns** properties. This is done by giving grid-template-columns the number of size values in pixels for as many columns as you want and doing the same for the grid-template-rows.

⇒ **EXAMPLE** We can use the following grid container setup to create a two-row, three-column grid:

```
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
  grid-template-rows: 80px 200px;
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}
.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
```

⇒ **EXAMPLE** The output



Notice that we created the three by two grid by giving the `grid-template-columns` three values; we used `auto` in this case, which will automatically and evenly distribute the column's widths. Then, we gave two values to `grid-template-rows` to indicate two rows, the first at 80 pixels high and the second at 200 pixels high.

Also, notice the `gap` and `padding` properties; these are used to provide the darker blue outline around and between each item. Without this, the items would be flush with each other. You may also see this property written as `grid-gap` instead of `gap`. Early versions of this property name were `grid-gap`, and to maintain compatibility with older websites, browsers will still accept `grid-gap` instead of `gap`.

In addition to the previous settings, the container also has the `justify-content` property for horizontal alignment and `align-content` property for vertical alignment, both with similar values to flexbox.

Value	Effect
space-evenly	Evenly spaces the columns within the container
space-around	Evenly distributes the available space around each item, slightly different than space-evenly

space-between	Only distributes the available space between the items, with the first and last columns being flush with containers
center	Centers the grid in the middle of the available space
start	Aligns items to the left
end	Aligns items to the right

The `align-content` property has the same set of values, but it handles the vertical alignment and positioning of the grid items.

The CSS grid layout, while very similar to flexbox, is ideal for situations where you simply need to organize a gallery of items on the screen and do not want to manually come up with some sort of layout technique or trickery using CSS . . . or worse, an HTML table (not that there's anything wrong with a table, they are just just less responsive and more tedious to code).

In addition to the grid container, there are properties that can be applied to the items themselves. These properties are primarily dedicated to spanning items across multiple columns or rows or both. Note that there are two different value formats, with both producing the same results.

Property	Possible Value Formats	Description
grid-column	start / end before 1 / 5 start / span across additional # columns 1 / span 3	This starts the items on Grid Column 1 and spans across, ending prior to Column 5 (spans across Columns 1 through 4).
grid-row	1 / 5 1 / span 3	This starts the item on Row 1 and spans down, stopping just prior to Row 5 (spans across Rows 1 through 4).
grid-area	starting row / starting column / ending row / ending column 1 / 2 / 5 / 6 starting row / starting column / span across additional # rows / span across additional # columns 1 / 2 / span 2 / span 3	Grid-area is shorthand for grid-row and grid-columns. In this case, the item will start on Row 1, Column 2, and end prior to Row 5 and Column 6.



BIG IDEA

CSS grids are much easier to make responsive since the item's properties can easily be adjusted using media queries, in addition to adjusting the container itself by changing the number of columns or rows to better fit smaller or larger screens.



TERMS TO KNOW

Grid

A value for the CSS display property that turns an HTML element into a grid container.

Grid-Template-Rows

A CSS property for grid display containers that controls the number and width of rows in the grid layout.

Grid-Template-Columns

A CSS property for grid display containers that controls the number and width of columns in the grid layout.

Grid-Column

A CSS property for grid items that spans an item's space across multiple columns.

Grid-Row

A CSS property for grid items that spans an item's space across multiple rows.

Grid-Area

A CSS property shorthand for grid items that combine grid-column and grid-row.

6. Grid for Page Layout

So, when do we use flex and when do we use grid?

The choice is up to you. Both can be used effectively to create whole-page layouts. Flex is great for providing layout control for sections of a page and reorienting items vertically or horizontally. Grid, on the other hand, is great for whole-page layouts as well as neatly organizing a subsection of content, such as a gallery of products or pictures.

The key difference between the two is that flex only gives you control over the structure in one direction (horizontal or vertical), while grid gives you control over both. Grid allows you to define the number of columns and rows needed for the elements. Flex only controls a list of elements.

⇒ **EXAMPLE** Let us use grid to lay out an entire page:

```
<style>
  .item1 { grid-area: header; }
  .item2 { grid-area: menu; }
```

```

.item3 { grid-area: main; }
.item4 { grid-area: right; }
.item5 { grid-area: footer; }

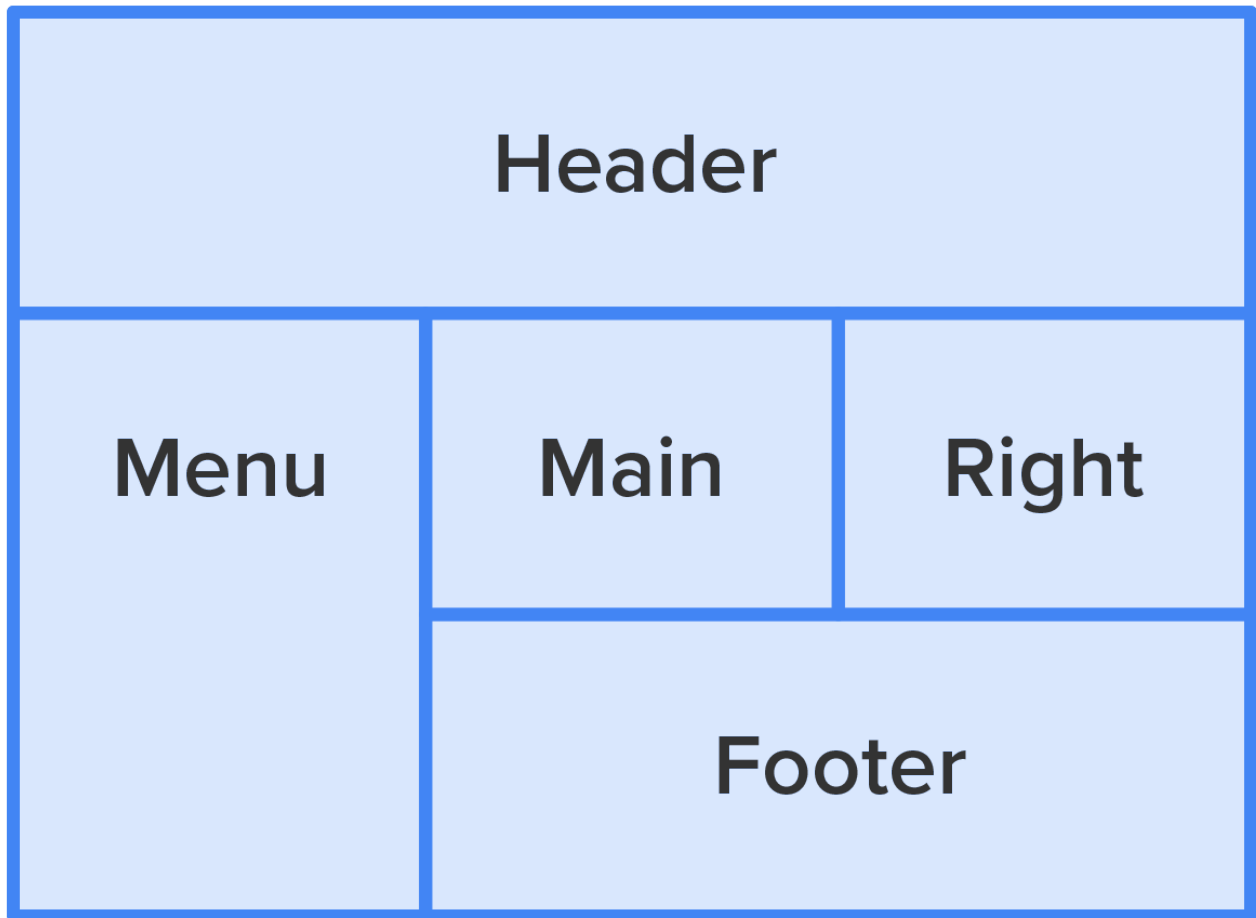
.grid-container {
  display: grid;
  grid-template-areas:
    'header header header header header header'
    'menu main main main main right'
    'menu footer footer footer footer footer';
  gap: 10px;
  background-color: #2196F3;
  padding: 10px;
}

.grid-container > div {
  background-color: rgba(255, 255, 255, 0.8);
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}
</style>

<body>
  <div class="grid-container">
    <div class="item1">Header</div>
    <div class="item2">Menu</div>
    <div class="item3">Main</div>
    <div class="item4">Right</div>
    <div class="item5">Footer</div>
  </div>
</body>

```

⇒ EXAMPLE The output



Let us take a closer look at the code.

⇒ **EXAMPLE** We started by giving each of the items a grid-area value equal to their name.

```
.item1 { grid-area: header; }  
.item2 { grid-area: menu; }  
.item3 { grid-area: main; }  
.item4 { grid-area: right; }  
.item5 { grid-area: footer; }
```

Then, we used those names to indicate what area each item should occupy in the next style rule for the **grid-template-area** property.

Next, we set the style rule for the grid-container.

⇒ **EXAMPLE** We set the display property to grid, and then we use the shorthand grid-template-area (shorthand for grid-template-row and grid-template-column) to configure the number of rows and columns.


```
grid-container {  
  display: grid;  
  grid-template-areas:  
    'header header header header header header'  
    'menu main main main main right'  
    'menu footer footer footer footer footer';  
  gap: 10px;  
  background-color: #2196F3;  
  padding: 10px;  
}
```

Each row is represented by a space-separated string of grid-area names, like “header header header header header header,” which tells CSS to configure the grid to have six columns and, for this one row, assign the header item all six spaces. The second row is defined the same way, but using different item names. This creates the second row and gives the Menu one portion (one of six); the Main gets four portions, and the last portion is given to the Right item. Lastly, the final row gives one portion to Menu and the remaining five portions to Footer.

In order to make this responsive, you will simply use a media query and assign the section names a different grid-template-areas value that fits the screen size and orientation.



TERM TO KNOW

Grid-Template-Areas

The shorthand property for grid-template-rows and grid-template-columns that accepts one string of space-separated sizes or item grid-area names for each row.



SUMMARY

In this lesson, you learned about the **positioning** property, as well as the **inline** and **block** display values. You also learned about the **float** property and how it can be used to stack block-level elements side by side. You learned about how the **flexbox** and **grid display** settings can easily organize any number of elements to neatly stack side by side or in a grid-type layout. Lastly, you learned about how grid can be used to provide a full-page layout.

Source: This Tutorial has been adapted from "The Missing Link: An Introduction to Web Development and Programming " by Michael Mendez. Access for free at <https://open.umn.edu/opentextbooks/textbooks/the-missing-link-an-introduction-to-web-development-and-programming>. License: **Creative Commons attribution: CC BY-NC-SA**.



TERMS TO KNOW

Absolute

A value of the position property that positions an element based solely on the top left corner of the page's content and uses the top or bottom and left or right properties to control its position, wherein the element scrolls with the surrounding content.

Align-Content

A property of a flex container that works with flex-wrap to control the spacing of the rows or columns of wrapped items.

Align-Items

A property of a flex container that controls the vertical alignment of flex items.

Block

A value of the display property that allows an element to be resized using height and width and be repositioned within its relative position.

Clear

A CSS property, used in conjunction with float, given to the neighboring element of a floating element, which forces the surrounding content to be pushed below the floating element.

Display Property

The CSS property that controls how an element is displayed on a webpage.

Fixed

A value of the position property that positions an element relative to the viewport and uses the top or bottom, left or right, properties, does not move when the user scrolls.

Flex-Direction

A property of a flex container that changes the orientation of the items from a row of items or a column of items.

Flex-Flow

A shorthand property for both flex-direction and flex-wrap.

Flex-Wrap

A property of a flex container that allows flex items to wrap down to the next row or column when there is not enough space.

Float

A CSS property that allows a block-level element to float either to the left or to right of the page or its container.

Grid

A value for the CSS display property that turns an HTML element into a grid container.

Grid-Area

A CSS property shorthand for grid items that combine grid-column and grid-row.

Grid-Column

A CSS property for grid items that spans an item's space across multiple columns.

Grid-Row

A CSS property for grid items that spans an item's space across multiple rows.

Grid-Template-Areas

The shorthand property for grid-template-rows and grid-template-columns that accepts one string of space-separated sizes or item grid-area names for each row.

Grid-Template-Columns

A CSS property for grid display containers that controls the number and width of columns in the grid layout.

Grid-Template-Rows

A CSS property for grid display containers that controls the number and width of rows in the grid layout.

Inline

A value of the display property that allows an element to flow with the text, as if it were part of the paragraph.

Justify-Content

A property of a flex container that controls the horizontal alignment of flex items.

Position

A CSS property that determines how a browser positions an element on the page.

Relative

A value of the position property that positions an element relative to its location in the code but also can be adjusted further by using either the top or bottom and left or right properties.

Static

A value of the position property that positions an element where it exists in the code and ignores the top, left, right, and bottom properties.

Sticky

A value of the position property that positions an element within a container and will remain within view of the container once the user scrolls past the element.