# CSS Layouts

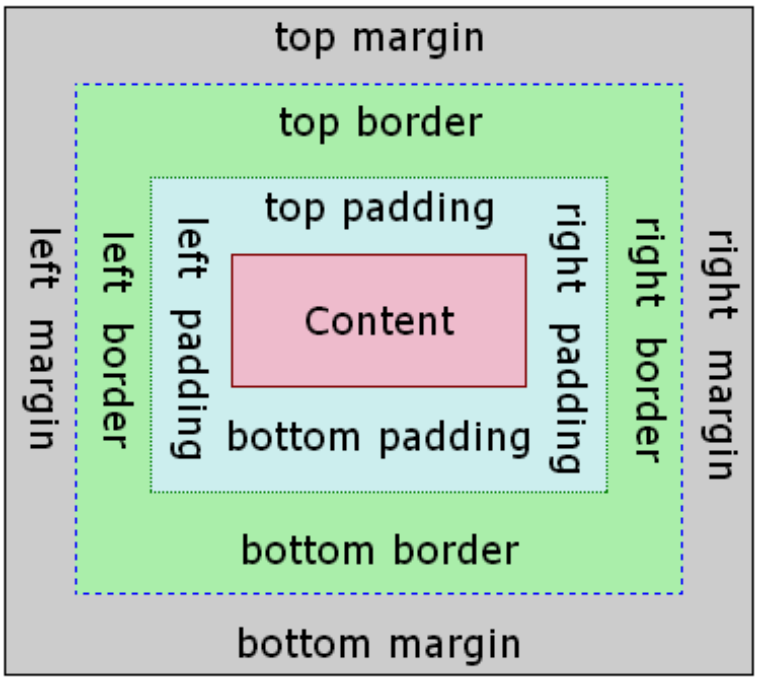
## Box Model ----------

Every section element (such as article, footer, and nav) and every grouping element (such as paragraph, list, and blockquote) is placed inside a CSS "box."

- Boxes stack on top of each other by default.
- They can be repositioned with CSS float or display rules.
- They can also be nested inside other boxes.

Use the ** ** element in HTML for layout purposes when no semantic element is a good fit.

**CSS** `padding`**:** property that adds spacing between the content and border – **use em unit**s
**CSS** `border`**:** property that draws a line around the content and padding area – **use px units**
**CSS** `margin`**:** property that adds spacing outside the border – **use em units**




### Box Model Example

```
div {
    width: 50%;
    padding-top: 1em;
    padding-right: 1em;
    padding-bottom: 1em;
    padding-left: 1em;
    border: 1px solid black;
    margin: 1em;
}
```

## Box Model Properties

**`width`**: most commonly specified using em (fixed width) or % (fluid percentage width) units.

```
width: 50%;
width: 30em;
```

**`height`**: *avoid choosing a value for the heights of elements!*

**padding-top**, **padding-right**, **padding-bottom**, **padding-left**

```
padding-top: 1em;
padding-right: 1em;
padding-bottom: 1em;
padding-left: 1em;
```

**padding:** shorthand property. A single value is applied to all sides. Four values are applied clockwise – top, right, bottom, left. 0 represents none and doesn't require units.

```
padding: 1em;
padding: 0 1em 0 1em;
```

**margin-top**, **margin-right**, **margin -bottom**, **margin-left**

```
margin-top: 1em;
margin-right: 1em;
margin-bottom: 1em;
margin-left: 1em;
```

**margin:** shorthand property. A single value is applied to all sides. Four values are applied clockwise – top, right, bottom, left. 0 represents none and doesn't require units.

```
margin: 1em;
margin: 0 1em 0 1em;
```

**border-width, border-style, border-color:** use px (pixel) units for the border width.

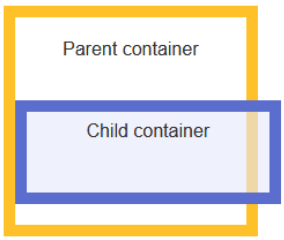
```
border-width: 1px;
border-style: solid;
border-color: #000000;
```

**border:** shorthand property including the width, style, and color.

```
border: 1px solid #000000;
```

## Box Sizing

The default CSS sizing is **box-sizing**: content-box, which means the width value is applied to the Content area only. Be careful when calculating your layouts, because the total width of the box in your layout will be wider than your width value.




If you apply the **box-sizing**: border-box rule, the width value is applied to the Content, Padding, and Border areas. Use this rule anytime you specify **width**: 100%, along with either **border** or **padding** values, because otherwise the border lines will spill outside the parent container as shown in the example on the left .

There is a CSS overflow property that specifies what should happen when the content is larger than its container, like in the above example or when an image is wider than its parent box. By default, it is visible, and spills outside the box. You could change it to hidden which effectively crops the content outside the box. You could also add scrollbars and turn it into a scrollable area. However, both options have usability problems. Following recommendations in this document will result in more usable, future-proof designs.

**Example 1:** width specified in em, so box-sizing is not needed.

```
div {
    width: 30em;
}
```

**Example 2:** width specified as 100% but there is no border or padding, so box-sizing is not needed.

```
div {
    width: 100%;
}
```

**Example 3:** width specified as 100% and there is border or padding, so box-sizing is recommended.

```
div {
    box-sizing: border-box;
    width: 100%;
    padding: 1em;
    border: 1px solid #000000;
}
```

## Centering a Box in its Parent

The box model **does not allow for vertical centering** without hacks or specifying the height, which is not recommended. However, horizontal centering is supported using margin: auto;

**For horizontal centering, the width of the centered box must be smaller than the container.**


### Box Model Example

```
div {
    background color: black;
    width: 80%;
    margin-left: auto;
    margin-right: auto;
}
```

## Centering Content in a Box

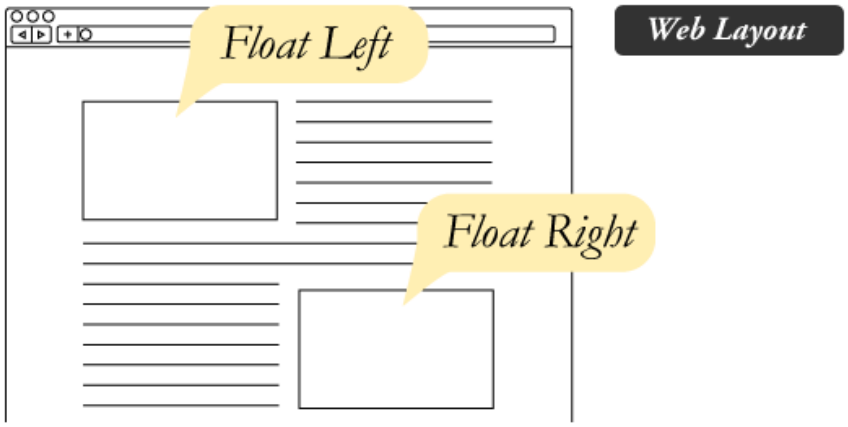

https://internetingishard.com/html-and-css/floats/

**text-align**

```
text-align: left;
text-align: center;
text-align: right;
```

## CSS Floats

The CSS float property allows on block to “float” within another. It specifies how that block will be aligned with adjacent elements, which will wrap around it.




# HTML

```
<article>
    <figure class="floatleft"><img rc="image/photo,jpg"></figure>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing lit…</p>
</article>
```

# CSS

```
.floatleft {
    float: left;
    width: 10em;
    margin-right: 1em;
    margin-bottom: 1em;
}

.floatright {
    float: right;
    width: 10em;
    margin-left: 1em;
    margin-bottom: 1em;
}

img {
    width: 100%;
}
```

### CSS Float Notes

You should create a container for the floated element. In many cases, the <figure> element is semantically appropriate. If not, you can use the  element.

**Images should always be set to 100% width** and placed inside a container with a width defined either with em or percent units. That way the image will always scale to fill its parent container and will look good in any sized browser or device.

**The floated container should have a smaller width than the parent container.**

Note that just one container element is needed to both float and size an image.

## CSS Flexbox ----------

CSS Flex is a powerful 1D layout system used to position a group of elements inside a container, like for links inside a menu or a boxes containing product summaries on a search results page.
Each direct child of the flexbox container will be one of the elements positioned in the container.
**Apply the following CSS properties to the container, not the contents!**

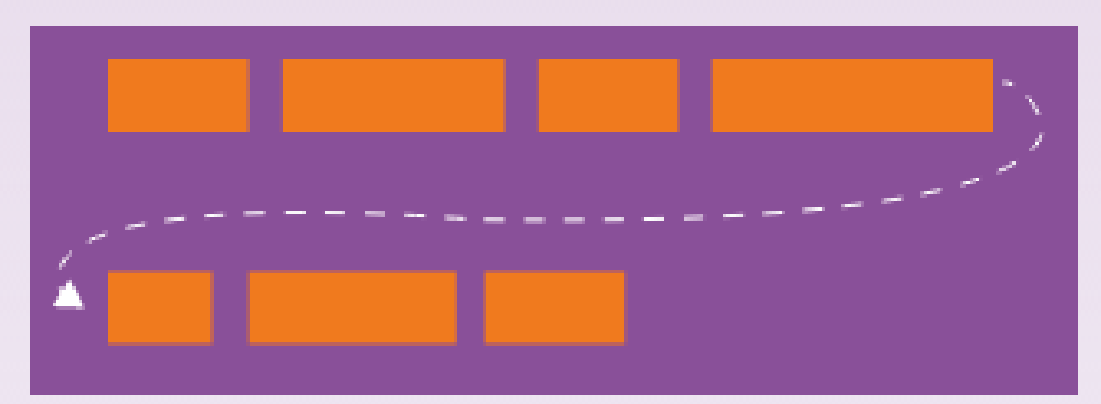
From: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

**display:** applies the specified layout inside a container

```
display: flex;
```

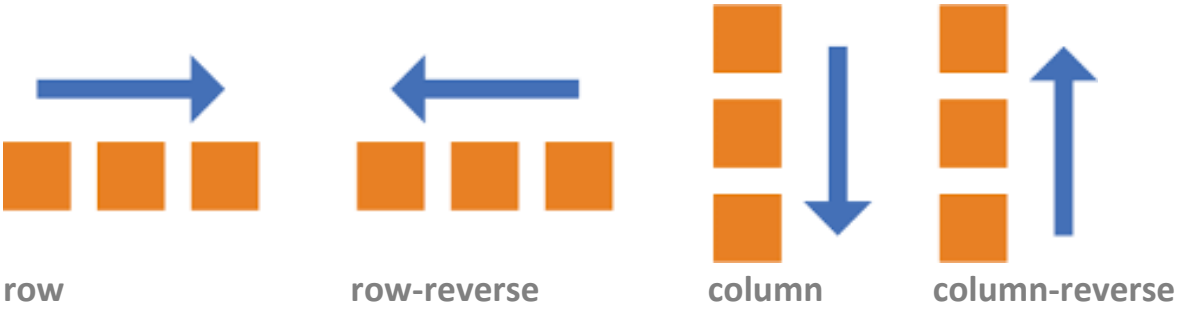
**flex-wrap:** specify if items fit in a single row or column or wrap to the next one.

```
flex-wrap: nowrap;
flex-wrap: wrap;
```



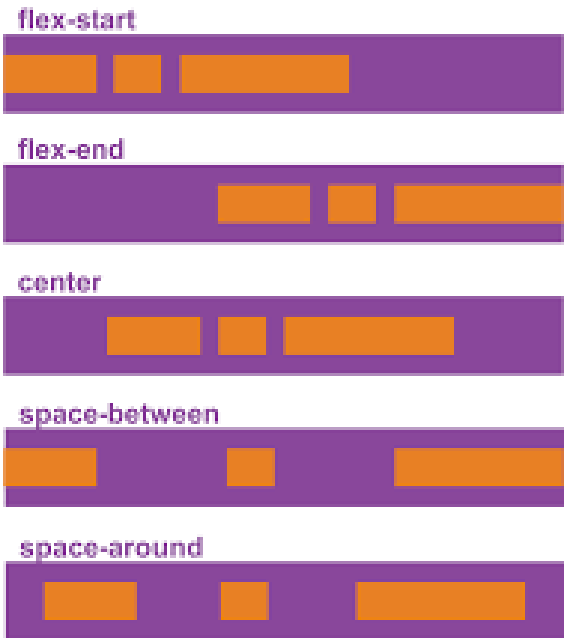
**flex-direction:** place the items in a row or column and specifies the direction

```
flex-direction: row;
flex-direction: row-reverse;
flex-direction: column;
flex-direction: column-reverse;
```



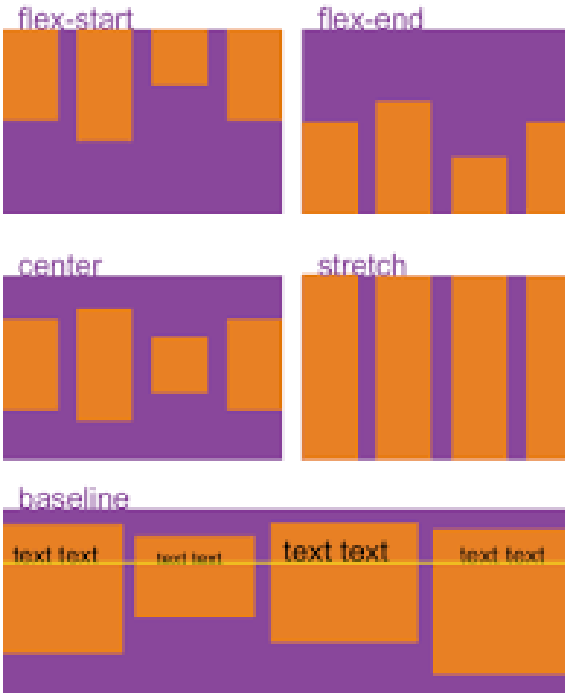
**justify-content:** horizontal positioning inside the container

```
justify-content: flex-start;
justify-content: flex-end;
justify-content: center;
justify-content: space-between;
justify-content: space-around;
```




**align-items:** vertical positioning inside the container

```
align-items: flex-start;
align-items: flex-end;
align-items: center;
align-items: space-between;
align-items: space-around;
```



## Flexbox Menu Example

Be careful when using space-between or space-around for the justify-content value, because the principle of proximity requires that menu items be closer to each other than other design elements.

# HTML


```
<nav>
    <ul>
        <li>Link 1</li>
        <li class="selected">Link 2</li>
        <li>Link 3</li>
        <li>Link 4</li>
    </ul>
</nav>
```

# CSS

```
nav ul {
    font-family: sans-serif;
    font-weight: bold;
    background-color: lightgray;

    /* flexbox rules */
    display: flex;
    flex-direction: row;
    flex-wrap: wrap;
    justify-content: flex-start;
    align-items: center;

    /* remove the list bullet points */
    list-style: none;
    padding-left: 0;
}

nav ul li {
    padding: .5em 1em .5em 1em;
}

nav ul li.selected {
    background-color: lightsteelblue;
}
```

## CSS Menu Notes

The <nav> element is only used for menus that navigate the main structure of the website. If you have both a main navigation menu and a secondary level navigation menu, you can select them using IDs.

HTML:

```
<nav id="mainmenu">...</nav>
```

CSS:

```
#mainmenu ul {...}
```

The links will eventually be placed inside <a> elements. So, remember to change their colors using pseudo selectors.

CSS:

```
nav ul li a:link {...}

nav ul li a:visited {...}

nav ul li a:hover {...}
```

# CSS Grid ----------

CSS Grid is a powerful 2D layout system used to create page layouts.

From https://css-tricks.com/snippets/css/complete-guide-grid/

**display:** applies the specified layout inside a container

```
display: grid;
```

**grid-template-columns** specify the number of columns and the width of each.

When mixing units, fixed **(em)** and percent **(%)** widths are calculated first.

Remaining space is divided among fractional units **(fr)**. They are proportional, so "1fr 1fr" makes two columns with equal widths, and "2fr 1fr" will make the first column twice as wide as the second.

The automatic **(auto)** value is calculated using the container size and column contents.

```
grid-template-columns: 15em 15em 15em 15em;
grid-template-columns: 30em 25% 1fr 1fr;
grid-template-columns: auto auto auto auto;
```

**grid-template-rows** specify the number of rows and the height of each.

The automatic **(auto)** value lets the browser calculate the height. Remember that fixed heights should not be specified. Let the browser determine how much space is needed for content to flow downward.

One exception is when there is a small amount of content on the page, and additional CSS code – media queries – are used to make sure it fits completely inside the browser window. The viewport height **(vh)** unit lets you make the grid container the full height of the browser view. 100vh = 100 percent of the viewport height. This can be used to vertically center an element in the browser.

```
grid-template-rows: auto auto auto auto;
grid-template-rows: 100vh;
```

**grid-column-gap:** specify the spacing between columns
**grid-row-gap:** specify the spacing between rows

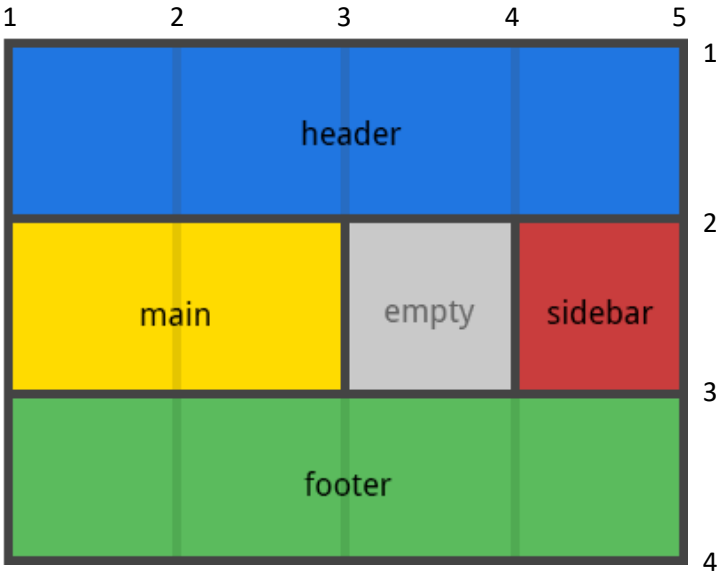
These properties will soon be replaced by column-gap and row-gap properties, which will also be available for flexbox and the upcoming multiple column layout specification.

```
grid-column-gap: 2em;
grid-row-gap: 2em;
```

**grid-column-start:** the column gridline number where a child element should start
**grid-column-end:** the column gridline number where it should end (optional for 1 cell element)
**grid-row-start:** the row gridline number where a child element should start
**grid-row-end:** the row gridline number where a it should end (optional for 1 cell element)

These properties are applied to the child elements, not the container.

## Grid Example




## HTML

```
<div id="content">
    <header id="header">...</header>
    <article id="main">...</article>
    <aside id="sidebar">...</aside>
    <footer id="footer">...</footer>
</div>
```

## CSS

```
#content {
    display: grid;
    grid-template-columns: 1fr 1fr 1fr 1fr;
    grid-template-rows: auto auto auto;
    grid-column-gap: 2em;
    grid-row-gap: 2em;
}

#header {
    grid-column-start: 1;
    grid-column-end: 5;
    grid-row-start: 1;
}

#main {
    grid-column-start: 1;
    grid-column-end: 3;
    grid-row-start: 2;
}

#sidebar {
    grid-column-start: 4;
    grid-column-end: 5;
    grid-row-start: 2;
}

#footer {
    grid-column-start: 1;
    grid-column-end: 5;
    grid-row-start: 3;
}
```

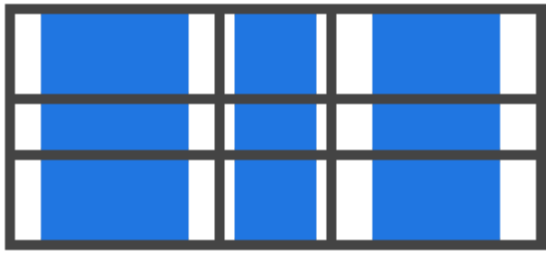
## Grid Alignment

**justify-items:** horizontal placement of content inside grid cells

```
justify-items: start;
justify-items: end;
justify-items: center;
justify-items: stretch;
```

**align-content:** vertical placement of content inside grid cells

```
align-items: start;
align-items: end;
align-items: center;
align-items: stretch;
```



`justify-items: center`

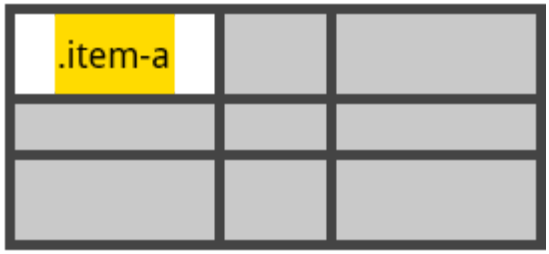
`align-content: center`

**justify-self**: horizontal placement of content inside a single grid cell

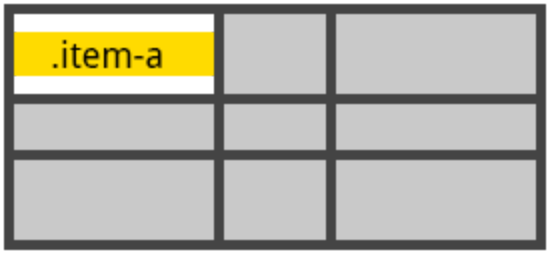
```
justify-self: start;
justify-self: end;
justify-self: center;
justify-self: stretch;
```

**align-self:** vertical placement of content inside a single grid cell

```
align-self: start;
align-self: end;
align-self: center;
align-self: stretch;
```




justify-self: center




align-self: center