

# **Introduction to Web Design and Development**

**Web Design and Development  
Bootcamp**

Version 1.0

# Table of Contents

<b>Chapter 1: HTML and CSS.....</b>	<b>1-1</b>
Topic: HTML and CSS Overview.....	1-1
HTML .....	1-1
The Power of CSS .....	1-2
Zen Garden without CSS.....	1-3
Many Designs.....	1-4
Learning from the Designs .....	1-6
Topic: HTML.....	1-8
HTML .....	1-8
HTML Elements.....	1-8
The HTML Grid .....	1-9
block elements.....	1-10
inline elements.....	1-12
inline-block elements .....	1-13
Semantic HTML.....	1-15
Other Semantic HTML.....	1-16
Common Semantic Elements.....	1-19
CSS.....	1-20
CSS Selectors .....	1-20
CSS Formatting.....	1-23
Developer Tools.....	1-24
<b>Chapter 2: Layout .....</b>	<b>2-1</b>
Topic: CSS Position .....	2-1
Static.....	2-1
Relative .....	2-2
Absolute .....	2-3
Fixed .....	2-4
Sticky.....	2-5
<b>Chapter 3: CSS FlexBox and Grid .....</b>	<b>3-1</b>
Topic: FlexBox .....	3-1
Direction .....	3-1
Justify Content.....	3-2
Row distribution .....	3-2
Column distribution.....	3-3
justify-content: space-between.....	3-3
justify-content: space-around .....	3-4
justify-content: center .....	3-4
justify-content: flex-end.....	3-5
FlexBox Summary .....	3-5
Topic: Grid.....	3-5
Defining a Grid.....	3-6
Column Width .....	3-6
Spacing Between Cells .....	3-8
Column / Row Span.....	3-10
Grid Summary.....	3-11

# Chapter 1: HTML and CSS

---

In this chapter we will look at the structure of a web page and how a page is built. HTML is the framework that defines the different regions and content of the page, while CSS is the artist that makes the page look beautiful.

## Topic: HTML and CSS Overview

---

### HTML

---

I like to think of building a web page like building a house. A good house must have a solid structure: a foundation, framed walls, a roof, floors, and stairs.



A solid foundation and framing are essential to building a house that will last. After the house is framed, and the electrical, plumbing and walls are completed, it is time for painting and finishing carpentry.



**HTML** is like the concrete foundation and framing of your page, and **CSS** is like the finishing carpentry of the site.

Just like it is impossible to have beautiful finish work in a house without a solid foundation, it is critical to also have a solid HTML foundation for your website that CSS can build on to create a beautiful design.

## The Power of CSS

---

The CSS Zen Garden is a website aimed at helping to demonstrate how HTML and CSS can work together to create many stunning designs (<http://csszengarden.com>).

The site was created to help designers share their designs, and to teach developers and designers how to use CSS in their websites.

The home page has a clean design with text that describes the purpose and mission of the site.

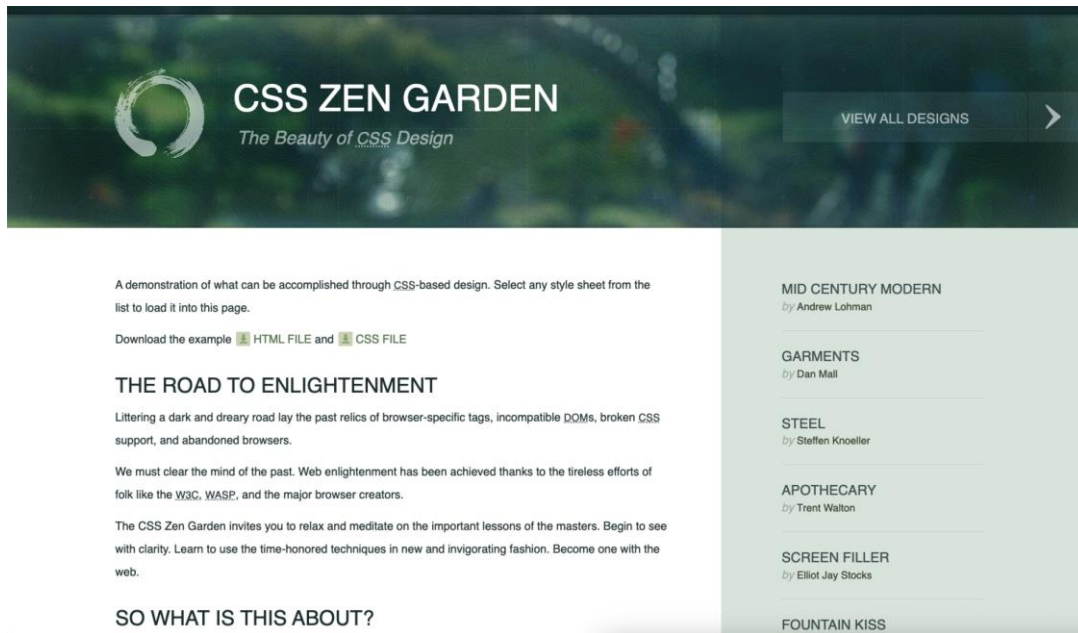


Image: <http://CSSZenGarden.com>

## Zen Garden without CSS

Below is the same website, but without the beauty of CSS formatting. It is just a regular HTML page with content without any styling (it's not particularly attractive).

### CSS Zen Garden

#### The Beauty of CSS Design

A demonstration of what can be accomplished through CSS-based design. Select any style sheet from the list to load it into this page.

Download the example [html file](#) and [css file](#)

#### The Road to Enlightenment

Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, broken CSS support, and abandoned browsers.

We must clear the mind of the past. Web enlightenment has been achieved thanks to the tireless efforts of folk like the W3C, WAMP, and the major browser creators.

The CSS Zen Garden invites you to relax and meditate on the important lessons of the masters. Begin to see with clarity. Learn to use the time-honored techniques in new and invigorating fashion. Become one with the web.

#### So What is This About?

There is a continuing need to show the power of CSS. The Zen Garden aims to excite, inspire, and encourage participation. To begin, view some of the existing designs in the list. Clicking on any one will load the style sheet into this very page. The HTML remains the same, the only thing that has changed is the external CSS file. Yes, really.

CSS allows complete and total control over the style of a hypertext document. The only way this can be illustrated in a way that gets people excited is by demonstrating what it can truly be, once the reins are placed in the hands of those able to create beauty from structure. Designers and coders alike have contributed to the beauty of the web; we can always push it further.

#### Participation

Strong visual design has always been our focus. You are modifying this page, so strong CSS skills are necessary too, but the example files are commented well enough that even CSS novices can use them as starting points. Please see the [CSS Resource Guide](#) for advanced tutorials and tips on working with CSS.

You may modify the style sheet in any way you wish, but not the HTML. This may seem daunting at first if you've never worked this way before, but follow the listed links to learn more, and use the sample files as a guide.

Download the sample [HTML](#) and [CSS](#) to work on a copy locally. Once you have completed your masterpiece (and please, don't submit half-finished work) upload your CSS file to a web server under your control. [Send us a link](#) to an archive of that file and all associated assets, and if we choose to use it we will download it and place it on our server.

#### Benefits

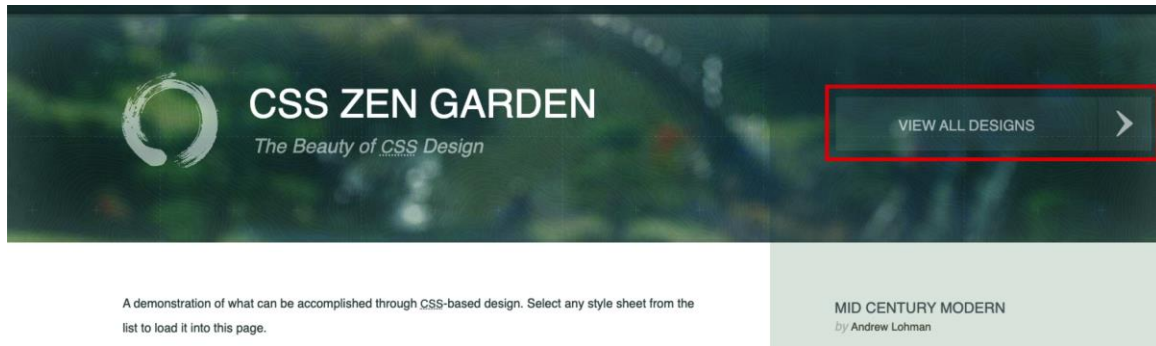
Why participate? For recognition, inspiration, and a resource we can all refer to showing people how amazing CSS really can be. This site serves as equal parts inspiration for those working on the web today, learning tool for those who will be tomorrow, and gallery of future techniques we can all look forward to.

#### Requirements

Where possible, we would like to see mostly CSS 1 & 2 usage. CSS 3 & 4 should be limited to widely-supported elements only, or strong fallbacks should be provided. The CSS Zen Garden is about functional, practical CSS and not the latest bleeding-edge tricks viewable by 2% of the browsing public. The only real requirement we have is that your CSS validates.

## Many Designs

The primary goal of this website is to allow web designers to submit their own CSS designs to demonstrate how the same content can be formatted to look completely different using just CSS. To view all designs that have been submitted, follow the **View All Designs** link.



Listed below are a few of the designs found on the website. The text of each page is identical. The only thing that has changed is that a different style has been applied.

### Design Name: A Robot Named Jimmy



Image: <https://csszengarden.com/215/>

Notice that the content of the web page is the same, but the design is different.

## Design Name: Under the Sea

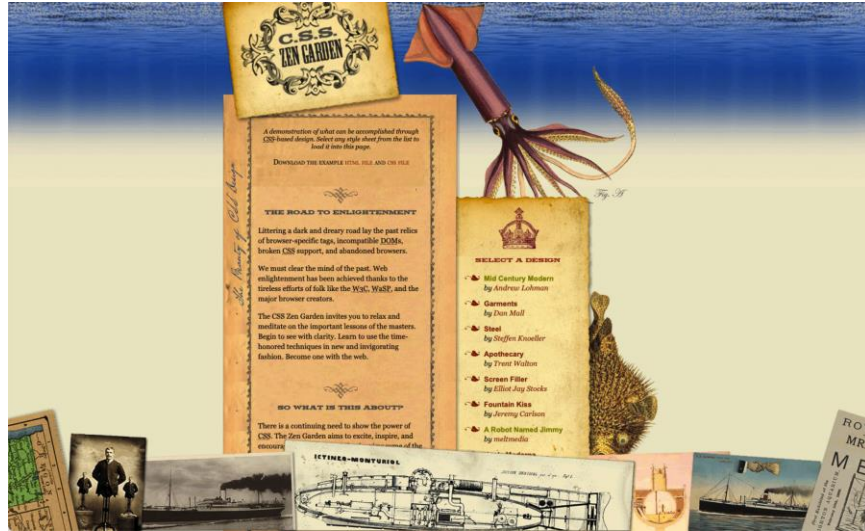


Image: <https://csszengarden.com/213/>

## Design Name: Make Em' Proud

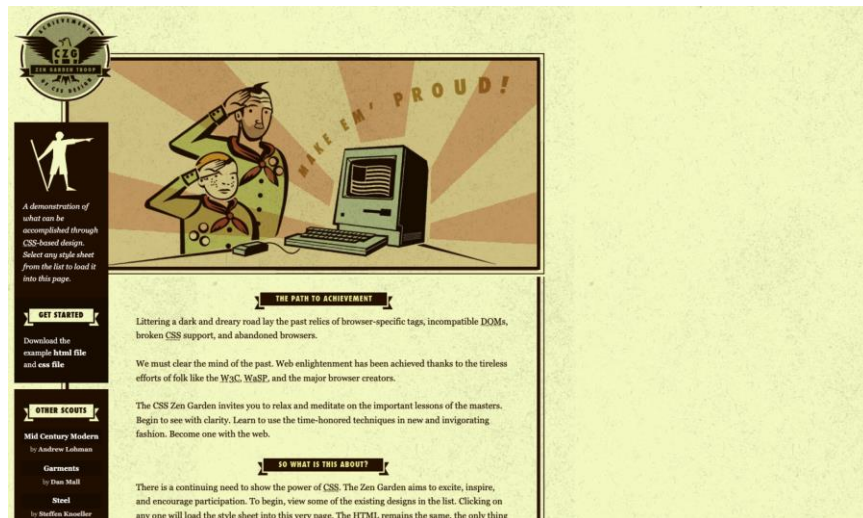


Image: <https://csszengarden.com/212/>



# Design Name: Release One



Image: <https://csszengarden.com/035/>

This is an early design, notice the section header fonts and the dotted lines. This style/design was popular in the early 2000s. Also, each section is clearly defined by hard lines and narrow margins. Today hard lines are largely avoided, and margins are wide, with an emphasis on empty space.


## Learning from the Designs

CSS Zen Garden allows you to download the raw HTML and the original CSS files. You can then open the files in VS Code to learn how the page was designed. You can also modify the CSS to try out your own designs.

NOTE: You should NOT edit the HTML file. The concept of this website is that you should ONLY modify the CSS file to achieve your design goals.


You can download both the HTML and CSS files to a new project folder, then open the new folder in VS Code to edit the page.







# CSS ZEN GARDEN

*The Beauty of CSS Design*

[VIEW ALL DESIGNS](#) 

A demonstration of what can be accomplished through CSS-based design. Select any style sheet from the list to load it into this page.

Download the example  HTML FILE and  CSS FILE

## THE ROAD TO ENLIGHTENMENT

### MID CENTURY MODERN

by Andrew Lohman

---

### GARMENTS

by Dan Mall

# Topic: HTML

---

In this topic we will discuss how HTML defines the structure of a page.

## HTML

---

**HTML:** **H**yper **T**ext **M**arkup **L**anguage is a markup language designed to describe the structure of a web page.

Browsers are programmed to understand HTML markup and then display the contents on a screen. This markup is made up of elements (or tags). Elements come in pairs that consist of an opening `<>` and a closing tag `</>`. The `<body></body>` element is the main canvas for our pages.

```
<html>
  <head>
  </head>
  <body>
  </body>
</html>
```

## HTML Elements

HTML elements consist of an opening and closing tag

```
<div></div>
```

The closing tag name must match the name of the opening tag.

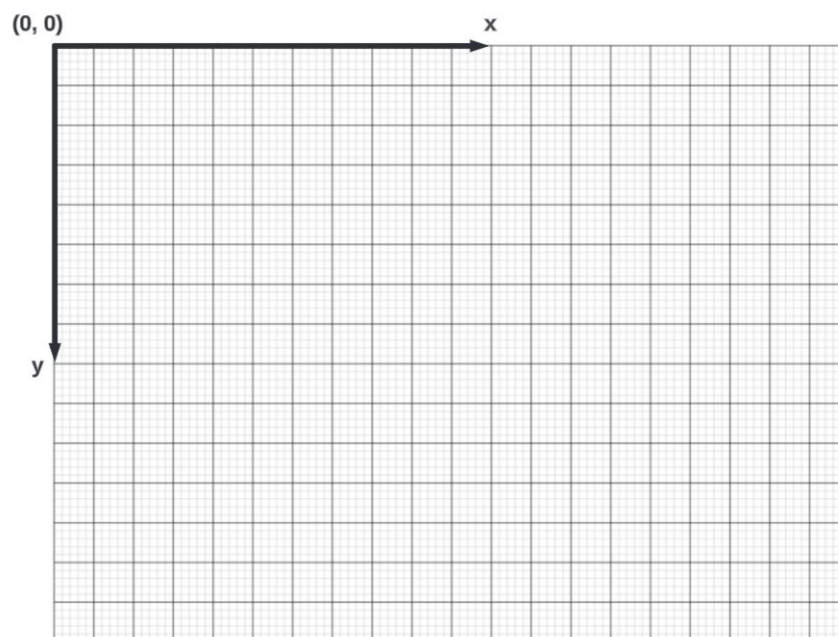
The opening tag can contain descriptive attributes. These attributes provide additional information to the browser on how to process the element.

```
<div id="main"  
      class="container"  
      style="background-color: whitesmoke;"  
      visible>  
</div>
```

Attributes can only exist in the opening tag. CSS frequently uses these attributes to define an element's style.

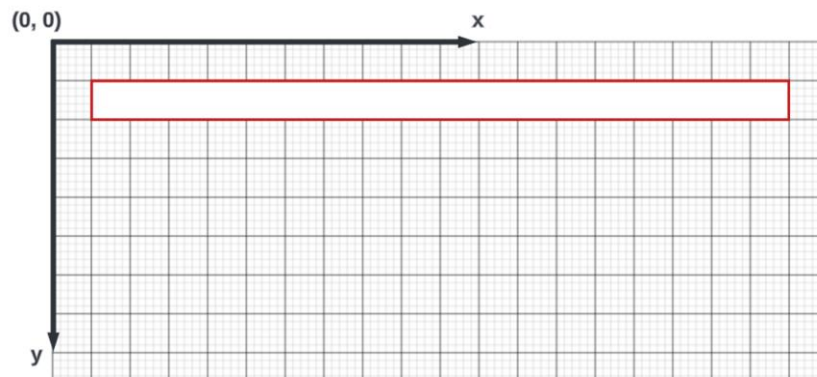
## The HTML Grid

Almost all browser screens are rectangular (except some phone screens now), so it is best to think of a screen as a rectangular canvas. The canvas is like a grid, or (x,y) coordinate plane, that begins at point (0,0) in the top left corner.



**Image:** The HTML grid

All HTML elements are also rectangular boundaries that will be placed on that canvas. Each element has a beginning  $(x, y)$  point as well as a `width` and `height`. This means that each element occupies a rectangular space on the grid.



**Image:** HTML element on the grid

Note: The border of elements are not visible by default, but they occupy a defined space on the grid.

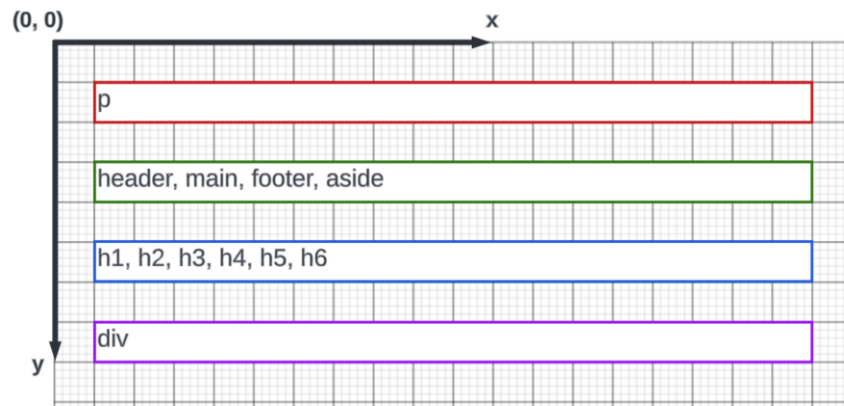
All HTML elements fall into 3 categories that define how the grid space is used: `block`, `inline` and `inline-block`.

W3Schools has a comprehensive list of HTML elements that fall into these categories [https://www.w3schools.com/html/html\\_blocks.asp](https://www.w3schools.com/html/html_blocks.asp).

Note: On this page W3Schools does not differentiate between `inline` and `inline-block` elements. All `inline-block` elements are grouped with the `inline` elements.

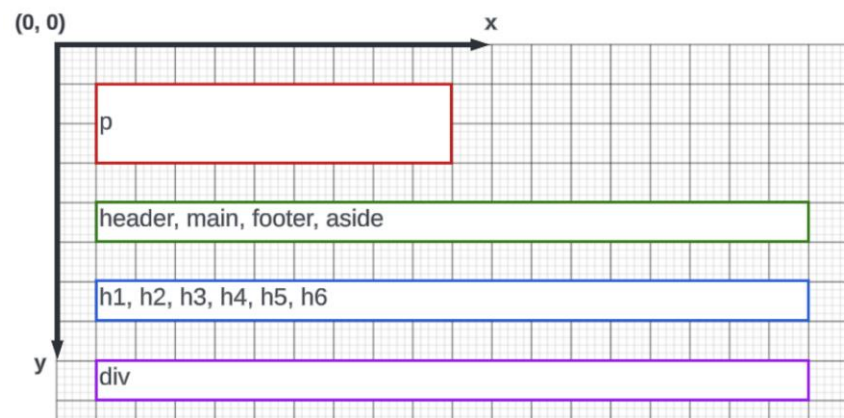
## **block** elements

Elements defined in the `block` category occupy an entire row (or block) of the grid. The browser will ensure that they are displayed at the beginning of the next line and that they occupy the entire width of that line.



**Image:** block elements

The width and the height of block elements can be manipulated through element attributes `<p width="50%" height="200px">` or through CSS. But the remaining space in that row will not be used by other elements.



**Image:** block elements with width/height values

Common **block** elements include:

```
<p></p>
<header></header>
<main></main>
<footer></footer>
<h1></h1> ... <h6></h6>
<div></div>
<ul>
  <li></li>
</ul>
```

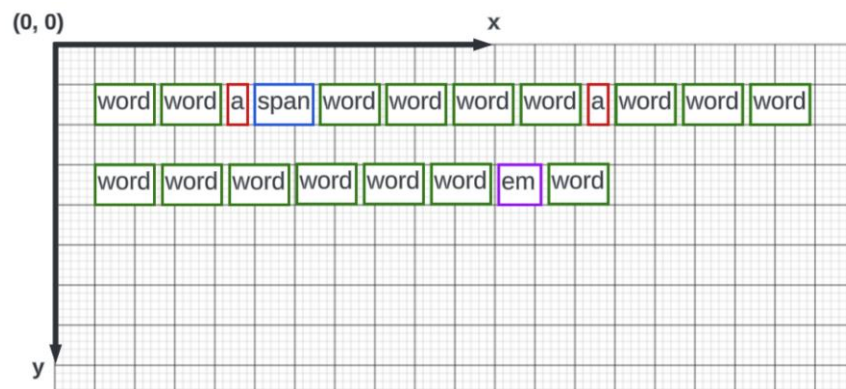
## inline elements

Elements defined in the `inline` category will only occupy as much space as is required by their content. For example, a hyperlink will only take the space needed to display the text of the link.

`Inline` elements DO NOT occupy the entire row. Instead, the next `inline` element is placed directly after the previous element on the same row.

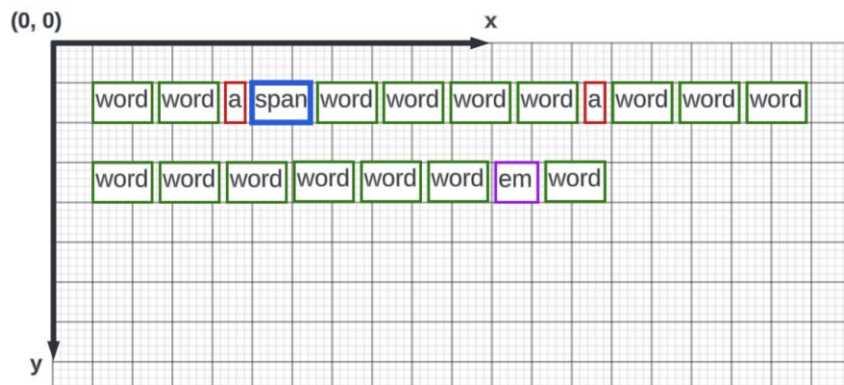
When there is no more space available on the row, the browser wraps the content and begins using the next row.

When `plain text` is added to a web page, each word is treated as its own `inline` element.



**Image:** inline elements wrap on the grid

Unlike `block` elements, if you attempt to change the `width` and `height` of an `inline` element, the amount of occupied space WILL NOT be changed on the grid. For instance, if I attempt to change the width and height of the `span` tag `<span width="500px" height="200px">`, the browser will not change how the `span` is displayed on the screen.



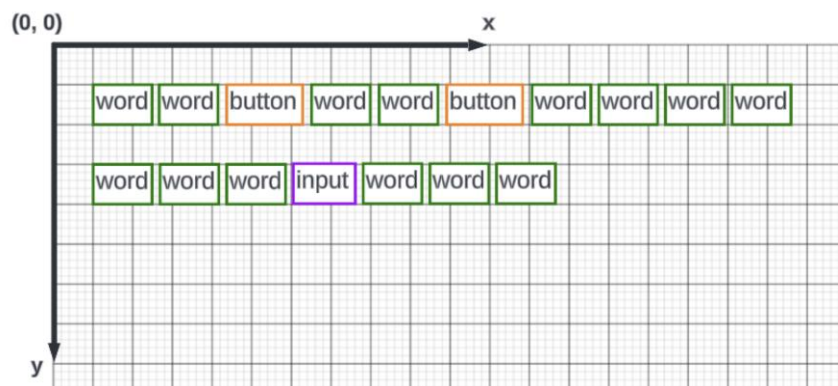
**Image:** inline span width/height has no effect on display

Common `inline` elements include:

```
plain text
<a></a>
<strong></strong>
<em></em>
<span></span>
```

## `inline-block` elements

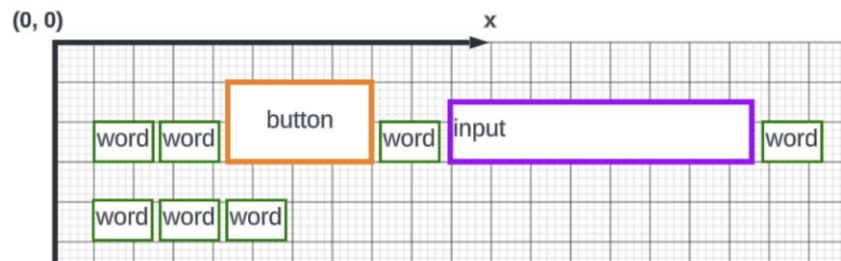
Like the `inline` elements, elements defined in the `inline-block` category will only occupy as much space as is required by their content. For example, a `button` will only take the space needed to display the text of the `button`. These elements are placed **in line** with the other elements.



**Image:** inline-block elements include `button` and `input`



The width and height of these elements, however, can be manipulated. This means that the display of the canvas will be altered. Changing the height of an `inline-block` element will change the height of the entire row. For instance specifying `<button width="200px" height="70px">` or `<input width="400px" height="50px">`



**Image:** inline-block elements width/height

Common `inline-block` elements include:

```
<button></button>
<input></input>
<select></select>
<image></image>
<textarea></textarea>
```

# Semantic HTML

---

Each HTML element serves a specific purpose. I.e., a hyperlink or `<a>` element is intended to display a clickable link to the user, and an `<input>` element will be rendered as a text box.

The supported list of HTML elements has evolved over time. This is largely because modern browsers are more powerful than early browsers, and have evolved to meet the needs of modern web application. But it is also because the intent of our HTML has changed.

Early HTML was treated mostly like a Word document, so elements were only thought of as a way to indicate if a word needed to be **bold**, underlined, or *italicized*. So early HTML included `<b>`, `<u>` and `<i>` elements. Over time it became apparent that these elements did not adequately represent the intent of what a content was trying to convey to the user. Rather, these elements were used to define the formatting of the page.

HTML evolved to support primarily **semantic** elements whose names convey their intent, not just containers that define visual formatting. Instead of a `<b>` tag which marks a word to be **bold**, we now use the `<strong>` tag. Browsers still format the content as **bold**, but screen readers, used by visually impaired individuals, can now more appropriately emphasize a word or phrase as important.

Elements like `<b>`, `<u>`, `<i>`, and `<center>` are no longer supported by many browsers. They have either been replaced by new semantic elements or removed entirely. The idea is that the HTML elements should be used to **describe** the **content** and CSS should be used to **define** the **formatting**.

<code>&lt;b&gt;</code>	-> <code>&lt;strong&gt;</code>
<code>&lt;i&gt;</code>	-> <code>&lt;em&gt;</code>
<code>&lt;u&gt;</code>	-> use CSS
<code>&lt;center&gt;</code>	-> use CSS

## Other Semantic HTML

In the past, when we needed to define different regions of our web page, such as a navigation bar, a main content area, header, footer or other common area, we would define each area using a `<div>`. The placement and formatting would be done using CSS, but all **containers** were defined as a `<div>`. The HTML almost always looked like some variation of the following example:

```
<body>
  <div id="nav">Navigation</div >

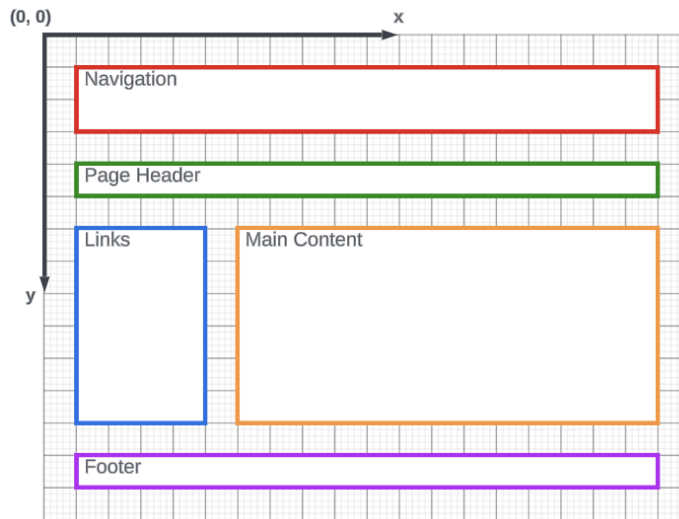
  <div id="header">Page Header</div >

  <div id="left">Links</div>

  <div id="main">Main Content</div >

  <div id="footer">Footer</div >
</body>
```

The pages would look something like this:



**Image:** Basic page design

Notice that all content on this page is contained inside of a `div`. The `div` tag was everywhere. It became an all purpose tool to solve every design problem. But like the `<b>` tag, it did not convey the meaning of the content, it was just a way to define a region that needed to be formatted.

All web pages are made up of regions that have specific meanings. They are similar to the image above: site or page navigation, main content area, an articles or links section, a section header, a page footer, etc.

The current HTML specification now includes elements that more appropriately define these page regions. Using semantic HTML, the above page might more appropriately be written like this:

```
<body>
  <nav>Navigation</nav>

  <header>Page Header</header>

  <section>Links</section>

  <main>Main Content</main>

  <footer>Footer</footer>
</body>
```

Notice that the HTML element names now almost exactly match what used to be the `id` of each div in the previous example. The elements now describe the content, instead of just defining a container that needs to be formatted.

## Common Semantic Elements

Common semantic elements include:

```
<article></article>  
<aside></aside>  
<details></details>  
<figcaption></figcaption>  
<figure></figure>  
<footer></footer>  
<header></header>  
<main></main>  
<mark></mark>  
<nav></nav>  
<section></section>  
<summary></summary>  
<time></time>
```

# CSS

---

CSS (Cascading Style Sheets) is used to format a web page. A CSS stylesheet has 2 parts: the selector, and the format rule.

## CSS Selectors

A selector is used to locate one or more HTML elements on a page.

In order to apply formatting to an element it must first be located.

## Element Selectors

An element selector targets all matching HTML elements on the page. This is done by specifying the name of the HTML element before the curly braces as follows:

```
/* this will target all <p></p> elements */  
p {  
  
}  
  
/* this fill target all <a></a> elements (hyperlinks) */  
a {  
  
}
```

```
<body>  
  <h1>Welcome to Introduction to CSS</h1>  
  
  <p>  
  This course will help you learn how to format your web pages  
  using <a href="https://www.w3schools.com/css/default.asp">CSS</a>  
  </p>  
</body>
```



## Class Selectors

A class selector allows developers to mark HTML elements on the page with a special class name, and then target those elements using CSS. This is done by specifying the name of the class before the curly braces. In the CSS file the class name must be preceded by a dot (**.**) as follows:

```
/* this will target all elements with a class named highlight */  
.highlight {  
  
}
```

```
<body>  
  <h1>Welcome to Introduction to CSS</h1>  
  
  <p>  
    This course will help you learn how to format your web pages  
    using <a class="highlight" href="...">CSS</a>  
  </p>  
  
  <p class="highlight">  
    You will learn through projects and practical exercises.  
  </p>  
</body>
```

Notice that the `highlight` class can be added to different types of elements. This allows you to control how different elements are formatted just by marking them with a class.

## Id Selectors

An Id selector allows developers to mark HTML elements on the page with a unique id, and then target that element by id using CSS. This is done by specifying the name of the class before the curly braces. In the CSS file the class name must be preceded by a hashtag (**#**) as follows:

```
/* this will target all elements with a class named highlight */  
.first-paragraph {  
  
}
```

```
<body>  
  <h1>Welcome to Introduction to CSS</h1>  
  
  <p id="first-paragraph">  
    This course will help you learn how to format your web pages  
    using <a class="highlight" href="...">CSS</a>  
  </p>  
  
  <p class="highlight">  
    You will learn through projects and practical exercises.  
  </p>  
</body>
```

Think of an id like you would a primary key in a database. There should only be one element on the page with the id of `first-paragraph`. This allows you to pinpoint exactly which element you want to target.

## Combining Selectors

Often you want to apply the same formatting to multiple types of elements. You can target multiple elements by adding a comma separated list in your CSS.

```
/* this will target all h1 though h6 tags on the page */  
h1, h2, h3, h4, h5, h6  
{  
  
}
```

```
<body>
  <h1>Welcome to Introduction to CSS</h1>

  <p id="first-paragraph">
    This course will help you learn how to format your web pages
    using <a class="highlight" href="...">CSS</a>
  </p>

  <h2>What you will learn</h2>

  <p class="highlight">
    You will learn through projects and practical exercises.
  </p>
</body>
```

## CSS Formatting

Once you have located your target elements you can apply formatting to those elements. The type of formatting that you can apply may depend on the element you have targeted.

Common formatting options include:

```
p {
  background-color: lightyellow;
  color: #990000; /* text color maroon */
  border: solid 1px black;
  font-family: arial, sans-serif;
  width: 200px;
  height: 50px;
}
```

## Changing the Display Type

We noted earlier that HTML includes 3 main display types: block, inline and inline-block. The default display type dictates whether an element can be on the same line as other elements, or even if the width and height can be adjusted.

Using CSS you can alter the display type of any element.

```
div {  
    display: inline-block;  
    /* divs will no longer occupy a full row */  
  
    width: 300px;  
}  
  
a {  
    display: inline-block;  
    /* the width and height of hyperlinks can now be altered */  
  
    width: 150px;  
    height: 25px;  
}
```

## Developer Tools

---

Styling your pages can be both a frustrating and a rewarding task. Often when you try to style a page things just don't look the way you want them to, and you need to **debug** your styles.

All modern browsers have built-in developer tools which allow you to see the details of a page and how CSS rules and formatting are applied. You should become familiar with the Developer Tools and use them frequently as you design your pages. They also provide robust debugging tools for JavaScript, which you will see in Sprint 2.

To open the Developer Tools you can click the **F12** key or you can right-click on an element and select **Inspect**.

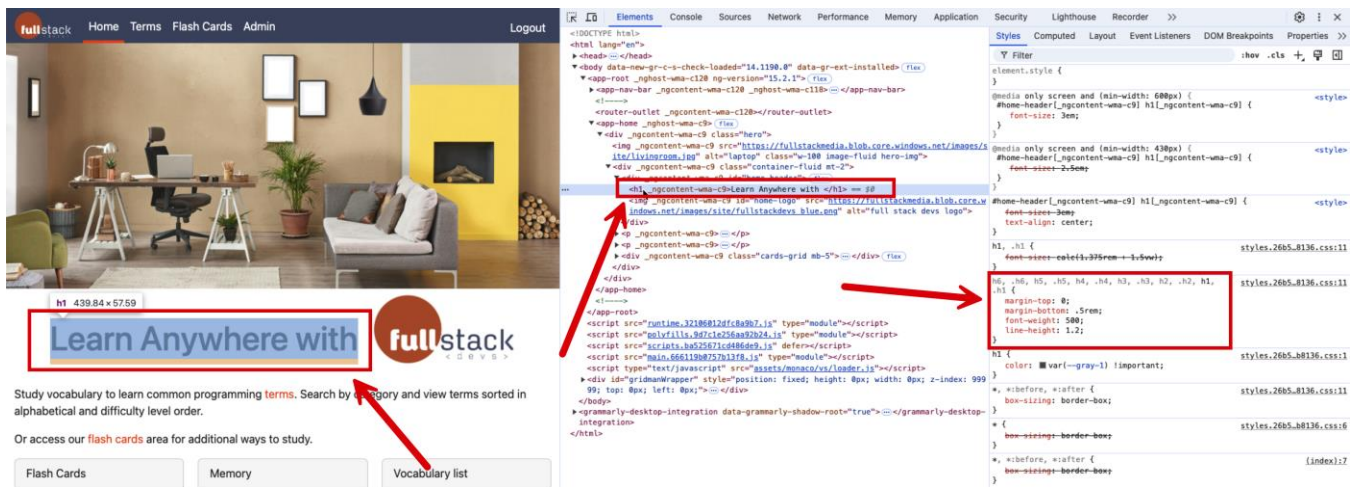


Image: FullStackDevs.net homepage – Developer Tools

With the developer tools you can highlight specific elements and visually see which element on the page is being referenced. You can also see which CSS styles are applied to that element.

As you complete your exercises throughout this week, you should form a habit of opening your developer tools and using them to help you understand your page layout and css.



# Chapter 2: Layout

---

Page layout is a critical part of website design. CSS offers several options for dealing with page layout.

## Topic: CSS Position

---

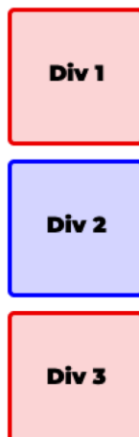
Before CSS3, objects could be placed on the screen using one of five possible positioning methods: static, relative, absolute, fixed, or sticky.

### Static

---

`static` is the default positioning used when a browser lays out the page. In other words, block elements are on their own line. The space used by the height of each element is occupied by the element. Additional elements are placed on the next line. `left` and `top` properties have no effect on the element.

```
.div-2 {  
  left: 30px;  
  top: 30px;  
}
```



**Image:** Div 2 uses the default `position: static`



As the page is scrolled, all elements scroll together.



**Image:** Div 2 scrolls with all other divs

## Relative

---

**relative** position aligns elements relative to the previous element.

When top or left properties are set, the element's position shifts in relation to the previous element. The space that the element would have occupied is still reserved by the browser.

```
.div-2 {  
  position: relative;  
  left: 30px;  
  top: 30px;  
}
```

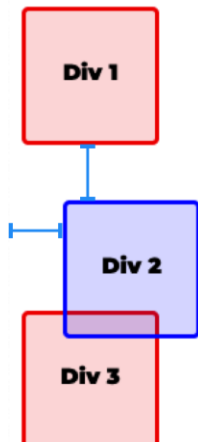


Image: Div 2 uses `position: relative`

As the page is scrolled, all elements scroll together.

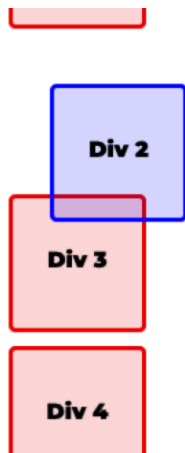


Image: Div 2 scrolls with all other divs

## Absolute

`absolute` position aligns elements relative to the body element. The element's position shifts in relation to the `top left` corner of the **parent** container. The space that the element would have occupied is no longer reserved by the browser.

```
.div-2 {  
  position: absolute;  
  left: 30px;  
  top: 30px;  
}
```

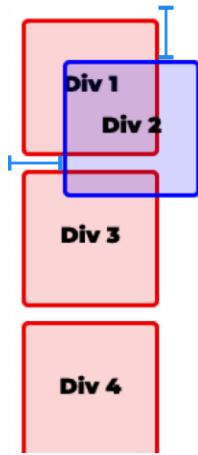


Image: Div 2 uses `position: absolute`

As the page is scrolled, all elements scroll together.

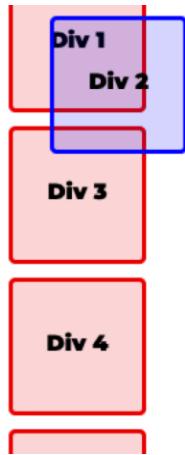


Image: Div 2 scrolls with all other divs

## Fixed

`fixed` position aligns elements relative to the page. The element's position shifts in relation to the `top left` corner of the **page**. The space that the element would have occupied is not reserved by the browser.

```
.div-2 {  
  position: fixed;  
  left: 30px;  
  top: 30px;  
}
```

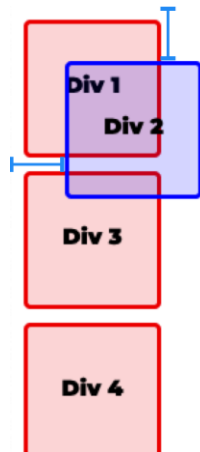


Image: Div 2 uses `position: fixed`

As the page is scrolled, fixed divs do not scroll.

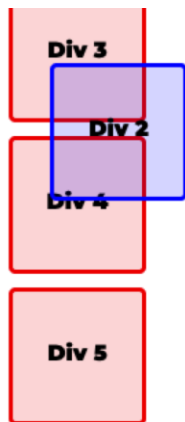
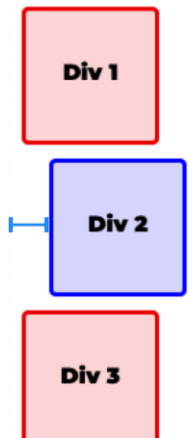


Image: Div 2 does not scroll

## Sticky

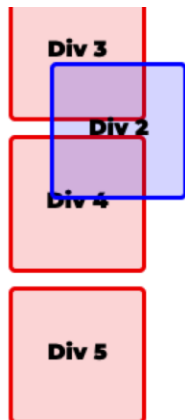
`sticky` initially the position aligns the **top** of an element relative to the bottom of the previous element, and the **left** of the element is aligned to the left side of the **page**.

```
.div-2 {  
  position: sticky;  
  left: 30px;  
  top: 30px;  
}
```



**Image:** Div 2 uses `position: sticky`

As the page is scrolled, the sticky div scrolls until it gets to the top of the page. Once it reaches the top of the page it no longer scrolls with the page.



**Image:** Div 2 stops scrolling at the top of the page

# Chapter 3: CSS FlexBox and Grid

---

The CSS FlexBox and Grid layout styles were added to CSS3 to simplify web page layout for designers.

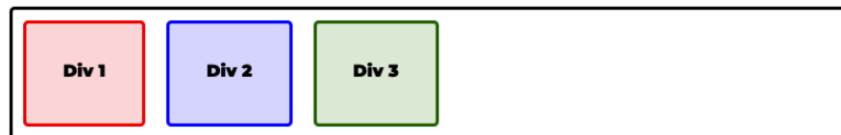
## Topic: FlexBox

---

CSS3 introduced two new tools that are used to assist in page layout: FlexBox and Grid. Unlike the other layout options we discussed, FlexBox is not a CSS position. Rather, it is a new `display` option.

An element with `display` set to `flex` controls the display of all child elements. The child elements' default display mode no longer applies.

```
#parent-div {  
    display: flex;  
}
```



**Image:** FlexBox container with 3 divs

The width of the above divs is **not** set. Notice that the child elements do not take up the full width of the container.

A FlexBox creates an container that acts as a stack panel. All elements are stacked either horizontally or vertically.

## Direction

---

The default direction of a flexbox is `row`. This means that child elements are stacked horizontally. The direction can be changed by setting the `flex-direction` property.

```
#parent-div {  
  display: flex;  
  flex-direction: column;  
}
```

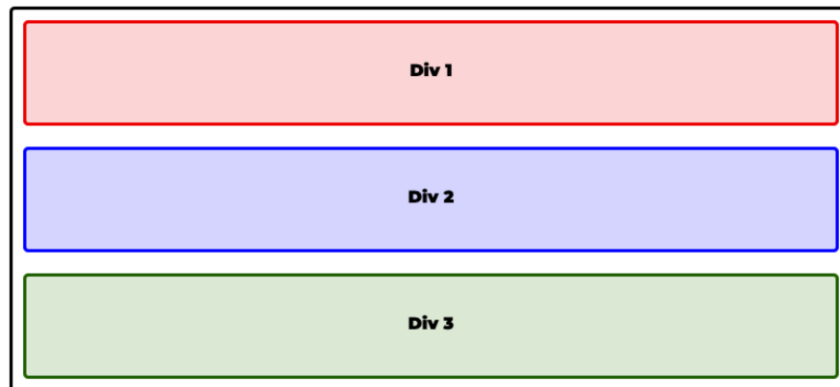


Image: FlexBox set to `flex-direction: column`

## Justify Content

---

Once the FlexBox has been established you have the ability to adjust how the child elements are distributed. The `justify-content` property is used to manage the distribution.

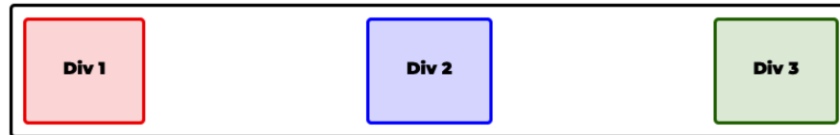
### Row distribution

If `flex-direction` is set to `row`, `justify-content` affects the `left-to-right` distribution.

```
#parent-div {  
  display: flex;  
  flex-direction: row;  
}
```



```
}
```

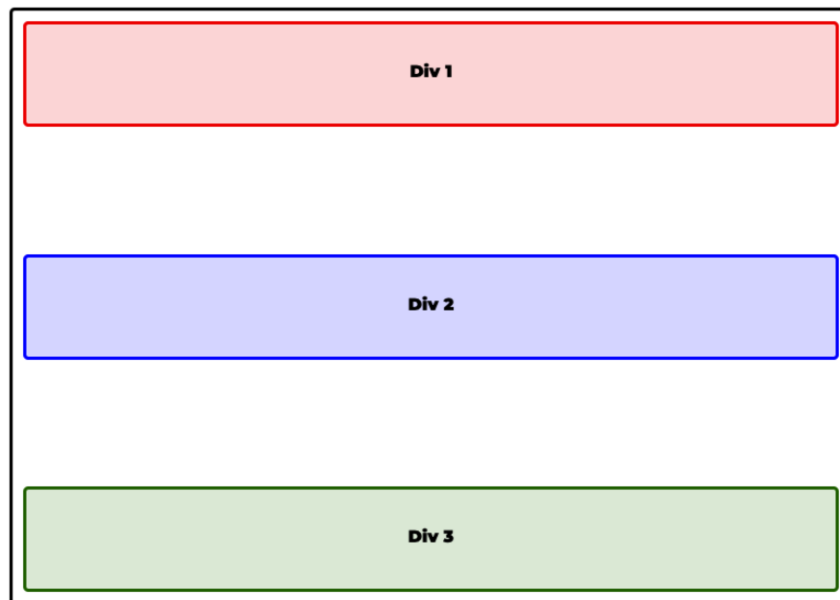


**Image:** Divs distributed across the row

## Column distribution

If `flex-direction` is set to column, `justify-content` affects the `top-to-bottom` distribution.

```
#parent-div {  
  display: flex;  
  flex-direction: column;  
}
```



**Image:** Divs distributed across a column

## `justify-content: space-between`

All extra space is evenly distributed between the elements. No extra space is added outside the first and last elements.

```
#parent-div {  
  display: flex;  
  flex-direction: row;  
  justify-content: space-between;  
}
```

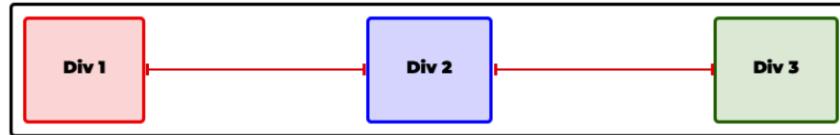


Image: space-between

## justify-content: space-around

All extra space is evenly distributed to the left and right of each element.

```
#parent-div {  
  display: flex;  
  flex-direction: row;  
  justify-content: space-around;  
}
```

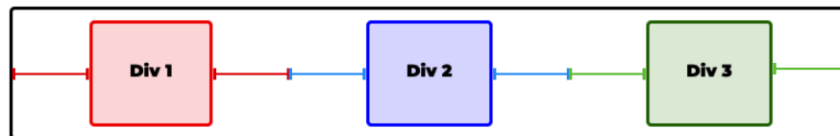


Image: space-around

## justify-content: center

All extra space is evenly distributed to the left and right of the first and last elements.

```
#parent-div {  
  display: flex;  
  flex-direction: row;  
  justify-content: center;  
}
```

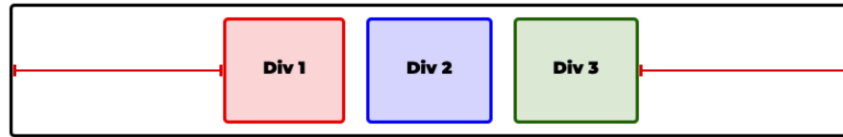


Image: center

## justify-content: flex-end

All extra space is placed at the beginning of the flexbox.

```
#parent-div {  
  display: flex;  
  flex-direction: row;  
  justify-content: flex-end;  
}
```



Image: flex-end

## FlexBox Summary

There are several other properties that can be used to define page layout using FlexBox. A great resource is available at <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>.

<http://flexboxfroggy.com> is an online game to help learn how to use FlexBox.

## Topic: Grid

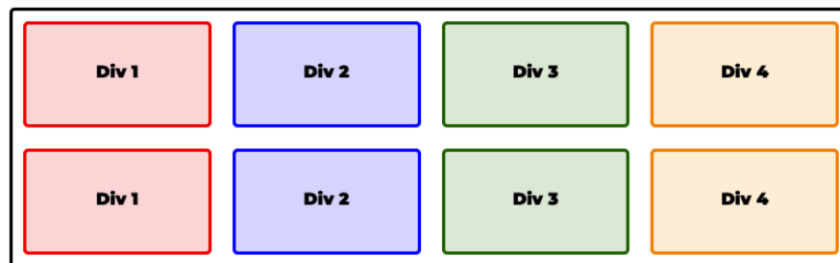
A CSS Grid is the other new layout option introduced in CSS3. A FlexBox allows you to layout pages in linear fashion, using **either** rows **OR** columns. But, a CSS Grid allows you to define a full 2-dimensional grid using both rows **AND** columns.

## Defining a Grid

A grid is defined by specifying the `display` property. By default a grid only has one column, so you must also define the number of columns.

A `1fr` unit is a fractional unit that defines a percentage of the available width. In this example since there are 4 `1fr` units, each takes 25% of available width. 8 elements would create 2 rows in this 4 column grid.

```
#parent-div {  
  display: grid;  
  grid-template-columns: 1fr 1fr 1fr 1fr;  
}
```



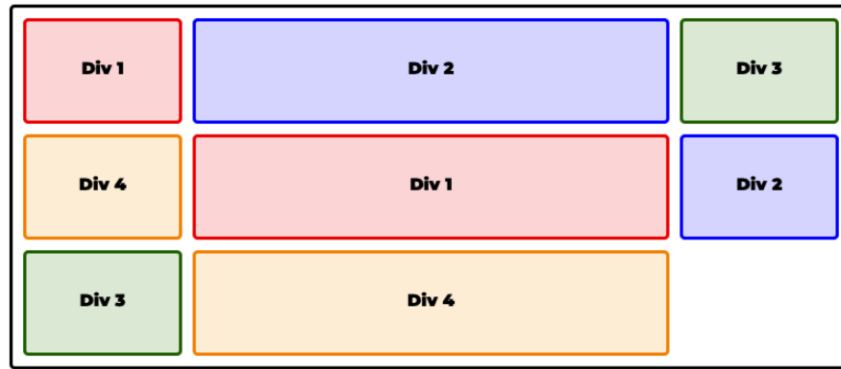
**Image:** Grid with 8 child elements

## Column Width

The width of columns can be modified by defining `px`, `%` or using multiple `fr` units.

### Width using `fr`

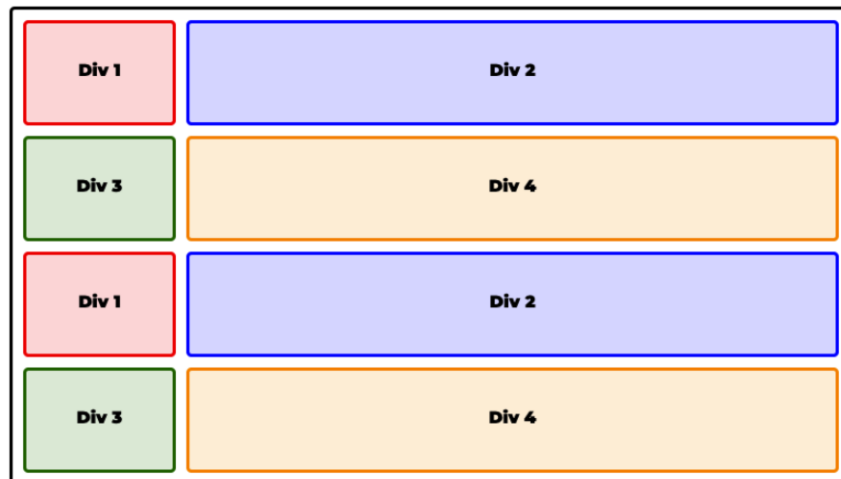
```
#parent-div {  
  display: grid;  
  grid-template-columns: 1fr 3fr 1fr;  
}
```



**Image:** 3 columns, middle 3x the size of the left and right

## Width using %

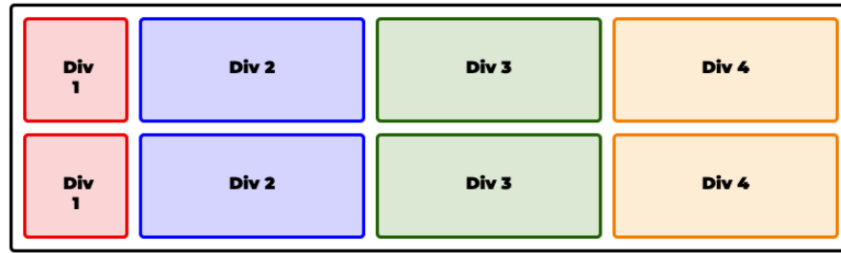
```
#parent-div {
  display: grid;
  grid-template-columns: 1fr 80%;
}
```



**Image:** 2 columns , right column at 80%

## Width using px

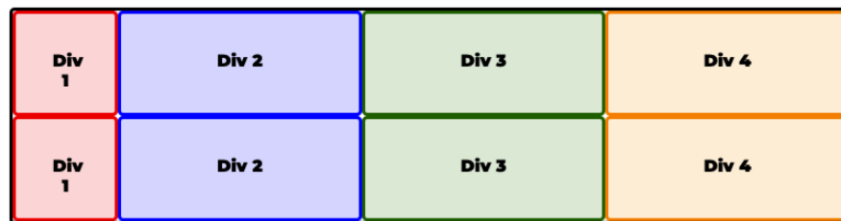
```
#parent-div {
  display: grid;
  grid-template-columns: 100px 1fr 1fr 1fr;
}
```



**Image:** 4 columns , left column at 100px

## Spacing Between Cells

By default, all space between cells (rows and columns) is 0.

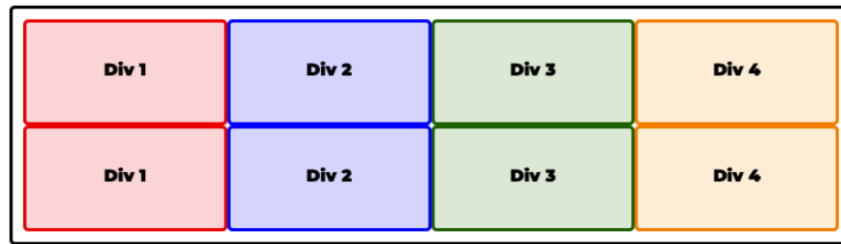


**Image:** Grid with no padding or gap

# Padding

Padding is used to define space outside of all columns and rows.

```
#parent-div {  
  display: grid;  
  grid-template-columns: 1fr 1fr 1fr 1fr;  
  padding: 10px;  
}
```

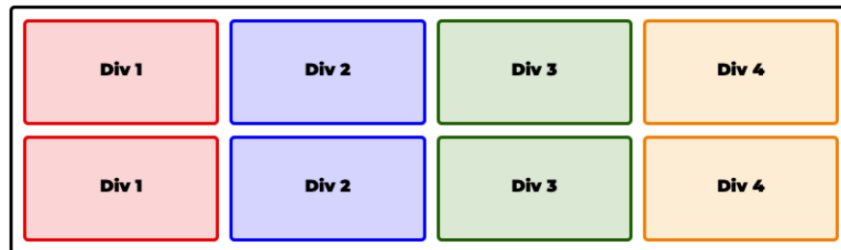


**Image:** Grid with 10px padding

# Gap

Gap is used to define the space between rows and columns.

```
#parent-div {  
  display: grid;  
  grid-template-columns: 1fr 1fr 1fr 1fr;  
  padding: 10px;  
  gap: 10px;  
}
```



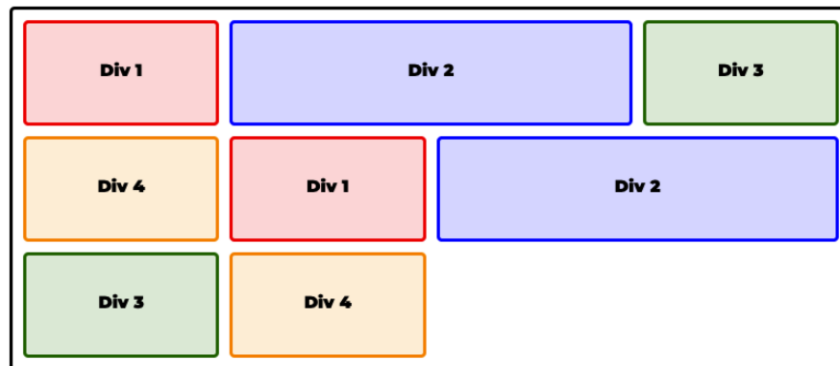
**Image:** Grid with 10px padding and gap

## Column / Row Span

Column and Row spans are used to take up space of more than one column or row.

### Column

```
.div-2 {  
  grid-column: span 2;  
}  
  
#parent-div {  
  display: grid;  
  grid-template-columns: 1fr 1fr 1fr 1fr;  
  padding: 10px;  
  gap: 10px;  
}
```

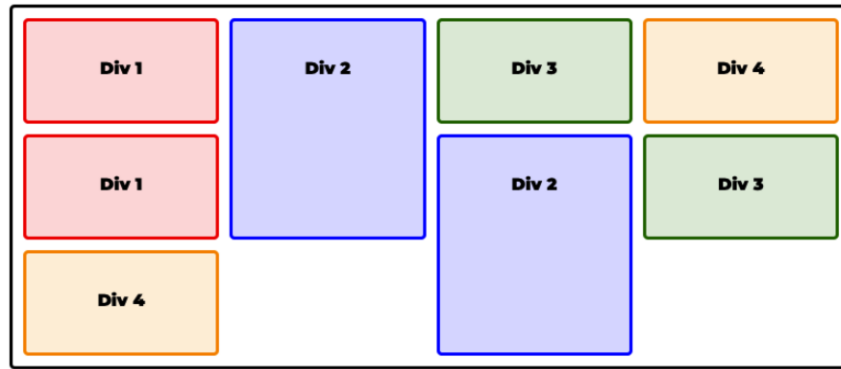


**Image:** Grid with div-2 column span

### Row

```
.div-2 {  
  grid-row: span 2;  
}  
  
#parent-div {  
  display: grid;  
  grid-template-columns: 1fr 1fr 1fr 1fr;  
  padding: 10px;  
  gap: 10px;  
}
```





**Image:** Grid with div-2 row span

## Grid Summary

A CSS Grid can also be defined by specifying named grid areas, and placing elements in each of those areas. There are also other properties that can be used to define page layout using Grid. A great resource is available <https://css-tricks.com/snippets/css/complete-guide-grid/>.

<http://cssgridgarden.com> is an online game to help learn how to use CSS Grid.