**CSS Basics #4: Grid System and Responsive Design -** [Eric Hu](https://huericnan.medium.com/?source=post_page-----ff211e074387--------------------------------) Feb 7, 2022

**Introduction**

The grid system is very important in CSS, it helps us design web pages without having to use float or positioning. The responsive grid layout is a unique type of CSS grid system because it appears differently on different devices: desktops, tablets, and phones.

*You can download the source code for this tutorial here:*

[*Download the Source Code*](https://www.ericsdevblog.com/index.php/beginners-roadmap-to-web-development/)

**Grid Layout**

A grid system consists of one parent element and several child elements like this:

<div class="grid-container">  
 <div class="grid-item">1</div>  
 <div class="grid-item">2</div>  
 <div class="grid-item">3</div>  
 ...  
</div>

Of course, we should tell the browser that this is a grid by setting the display property to grid:

.grid-container {  
 display: grid;  
}

This will set all elements with class="grid-container" as a grid container, and all direct children of the grid container automatically become grid items.

The vertical lines of grid items are called columns, and the horizontal lines of grid items are called rows. The spaces between each column/row are called gaps.

A picture containing shape

Description automatically generated

The grid-gap can be adjusted using one of the following properties:

* grid-column-gap
* grid-row-gap
* grid-gap

.grid-container {  
 display: grid;  
 grid-template-columns: auto auto auto auto;  
 grid-gap: 50px 100px;  
}

For example, in this case, the column cap is 100px, and the row gap is 50px. And for this demonstration, we have to add grid-template-columns: auto auto auto auto;, I'll explain what this means later in this article.

It is also possible to make one grid item span across multiple columns or rows:

<div class="grid-container">  
 <div class="grid-item1">1</div>  
 <div class="grid-item2">2</div>  
 ...  
</div>.grid-item1 {  
 grid-column-start: 1;  
 grid-column-end: 3;  
}.grid-item2 {  
 grid-column-start: 3;  
 grid-column-end: 5;  
 grid-row-start: 1;  
 grid-row-end: 3;  
}

Calendar

Description automatically generated

Diagram

Description automatically generated with low confidence

Pay attention to the numbers here. For grid-item1, it starts at line 1 and ends **before** line 3.

**Grid Container**

**The grid-template-columns Property**

The grid-template-columns property defines the number of columns in our grid layout, and we can also use it to define the width of each column.

<div class="grid-container">  
 <div class="grid-item">1</div>  
 <div class="grid-item">2</div>  
</div>.grid-container {  
 display: grid;  
 grid-template-columns: 150px 100px;  
}

A picture containing whiteboard

Description automatically generated

In this example, we defined two columns, the first one has a width of 150px, and the second one has a width of 100px. If we have more than two items in this two-column grid, a new row will be automatically added to put the items in.

A picture containing shape

Description automatically generated

We can also set the width of the column to be auto:

.grid-container {  
 display: grid;  
 grid-template-columns: 150px auto;  
}

Diagram

Description automatically generated with medium confidence

If both columns have the width="auto", then they should have the same width:

.grid-container {  
 display: grid;  
 grid-template-columns: auto auto;  
}

Diagram

Description automatically generated with low confidence

**The grid-template-rows Property**

The grid-template-rows property defines the height of each row:

.grid-container {  
 display: grid;  
 grid-template-columns: auto auto;  
 grid-template-rows: 100px 200px;  
}

A picture containing diagram

Description automatically generated

**The justify-content Property**

The justify-content property is used to horizontally align the whole grid inside the container. For this property to have any effect, the total width of the grid (All columns combined) must be less than the container's width, like this:

Calendar

Description automatically generated

With the justify-content property, we can align the columns differently.

.grid-container {  
 display: grid;  
 grid-template-columns: 100px 100px 100px;  
 justify-content: center;  
}

Calendar

Description automatically generated

.grid-container {  
 display: grid;  
 grid-template-columns: 100px 100px 100px;  
 justify-content: space-evenly;  
}

A picture containing text, clock

Description automatically generated

.grid-container {  
 display: grid;  
 grid-template-columns: 100px 100px 100px;  
 justify-content: end;  
}

Diagram

Description automatically generated

**The align-content Property**

The align-content property is used to vertically align the whole grid inside the container, and it works just like the justify-content property.

.grid-container {  
 display: grid;  
 align-content: space-evenly;  
}

A picture containing shape

Description automatically generated

**Grid Items**

**The grid-column Property**

The grid-column property is a shorthand property for the grid-column-start and the grid-column-end properties. It defines which column(s) to place an item.

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

A picture containing polygon

Description automatically generated

In this example, the first item starts from column 1 and ends **before** column 3.

**The grid-row Property**

The grid-row property is a shorthand property for the grid-row-start and the grid-row-end properties. It defines which row(s) to place an item.

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

A picture containing text, clock

Description automatically generated

**The grid-area Property**

The grid-area property is a shorthand property for the grid-row-start, grid-column-start, grid-row-end and the grid-column-end properties.

.grid-item1 {  
 grid-area: 1 / 1 / 3 / 3;  
}

This means that item 1 will start on row 1 and column 1, and end **before** row 3 and column 3.

**Responsive Layout**

As you know, the web pages that we design should be accessible on all kinds of devices, computers, tablets and mobile phones, but they all have different sizes. This means the layout that looks well on one device will be either too big or too small for the others.

The responsive layout is designed to solve this problem. It is a web design concept that allows us to create a single layout that can adapt to different screens. To understand the responsive layout, we need to first talk about the grid view and media query.

**Grid View**

The grid-view means that when we design our web page, we always divide the entire page into columns (usually 12 columns). These columns will have a total width of 100% and will shrink and expand as you resize your browser window.

First, we need to make sure that all HTML elements have the box-sizing property set to border-box. This is to make sure that the padding and borders are included in the total width and height of the elements:

\* {  
 box-sizing: border-box;  
}

We know that the total width of the page is 100%, and the page has 12 columns. Calculate the percentage for one column: 100% / 12 = 8.33%.

Then we can define the layout for this page:

.col-1 {width: 8.33%;}  
.col-2 {width: 16.66%;}  
.col-3 {width: 25%;}  
.col-4 {width: 33.33%;}  
.col-5 {width: 41.66%;}  
.col-6 {width: 50%;}  
.col-7 {width: 58.33%;}  
.col-8 {width: 66.66%;}  
.col-9 {width: 75%;}  
.col-10 {width: 83.33%;}  
.col-11 {width: 91.66%;}  
.col-12 {width: 100%;}

col-1 means this element takes 1 column, col-2 means it take 2 columns, and so on.

All of these columns should be floating to the left (we will discuss float in the next article), and should have padding:

[class\*="col-"] {  
 float: left;  
 padding: 15px;  
}

Let’s look at an example using this layout:

<div class="row">  
 <div class="col-3">Menu</div> <!-- 25% -->  
 <div class="col-9">Main</div> <!-- 75% -->  
</div>

A picture containing graphical user interface

Description automatically generated

**Media Queries**

Media query is another key concept in responsive design. It uses the @media rule to include a block of CSS properties only when certain conditions are met. Now we can design different layouts for desktop, tablet and mobile phones. Let's start with the easiest one, mobile phones:

/\* For mobile phones: \*/  
[class\*="col-"] {  
 width: 100%;  
}

Because the screens on mobile phones are very small, we want all elements to take 100% of the width of the screen. Then we can add the layout for tablet, but this time, we need to set a breakpoint:

/\* For mobile phones: \*/  
[class\*="col-"] {  
 width: 100%;  
}@media only screen and (min-width: 600px) {  
 /\* For tablets: \*/  
 .col-s-1 {width: 8.33%;}  
 .col-s-2 {width: 16.66%;}  
 .col-s-3 {width: 25%;}  
 .col-s-4 {width: 33.33%;}  
 .col-s-5 {width: 41.66%;}  
 .col-s-6 {width: 50%;}  
 .col-s-7 {width: 58.33%;}  
 .col-s-8 {width: 66.66%;}  
 .col-s-9 {width: 75%;}  
 .col-s-10 {width: 83.33%;}  
 .col-s-11 {width: 91.66%;}  
 .col-s-12 {width: 100%;}  
}

This means if the width of the screen is bigger than 600px, the second set of rules is activated.

And finally, we can define the desktop layout using the same method:

/\* For mobile phones: \*/  
[class\*="col-"] {  
 width: 100%;  
}@media only screen and (min-width: 600px) {  
 /\* For tablets: \*/  
 .col-s-1 {width: 8.33%;}  
 .col-s-2 {width: 16.66%;}  
 .col-s-3 {width: 25%;}  
 .col-s-4 {width: 33.33%;}  
 .col-s-5 {width: 41.66%;}  
 .col-s-6 {width: 50%;}  
 .col-s-7 {width: 58.33%;}  
 .col-s-8 {width: 66.66%;}  
 .col-s-9 {width: 75%;}  
 .col-s-10 {width: 83.33%;}  
 .col-s-11 {width: 91.66%;}  
 .col-s-12 {width: 100%;}  
}@media only screen and (min-width: 768px) {  
 /\* For desktop: \*/  
 .col-1 {width: 8.33%;}  
 .col-2 {width: 16.66%;}  
 .col-3 {width: 25%;}  
 .col-4 {width: 33.33%;}  
 .col-5 {width: 41.66%;}  
 .col-6 {width: 50%;}  
 .col-7 {width: 58.33%;}  
 .col-8 {width: 66.66%;}  
 .col-9 {width: 75%;}  
 .col-10 {width: 83.33%;}  
 .col-11 {width: 91.66%;}  
 .col-12 {width: 100%;}  
}

Notice that the last two sets of rules are almost identical, the only difference is the name. One starts with col and the other one starts with col-s. Why do we do this at all if they are the same? Because it gives us the opportunity to decide how many columns each element will take at each breakpoint.

For example:

<div class="row">  
 <div class="col-3 col-s-6">...</div>  
</div>

This <div> element will take 3 columns on desktops, but it will take 6 columns in tablet mode.