Contents

[Scss 2](#_Toc531872052)

[Url’s to follow 2](#_Toc531872053)

[Installation of scss 2](#_Toc531872054)

[Variables 2](#_Toc531872055)

[Nesting 2](#_Toc531872056)

[Partials 2](#_Toc531872057)

[Import 2](#_Toc531872058)

[Mixins 2](#_Toc531872059)

[Extend / inheritance 3](#_Toc531872060)

[Operators 3](#_Toc531872061)

[Flex 3](#_Toc531872062)

[Urls to follow 3](#_Toc531872063)

[Concept 3](#_Toc531872064)

[Flex container 4](#_Toc531872065)

[Flex items 5](#_Toc531872066)

[Grid 6](#_Toc531872067)

[Grid terminology is important to understand: 7](#_Toc531872068)

[Explicit and implicit grid 8](#_Toc531872069)

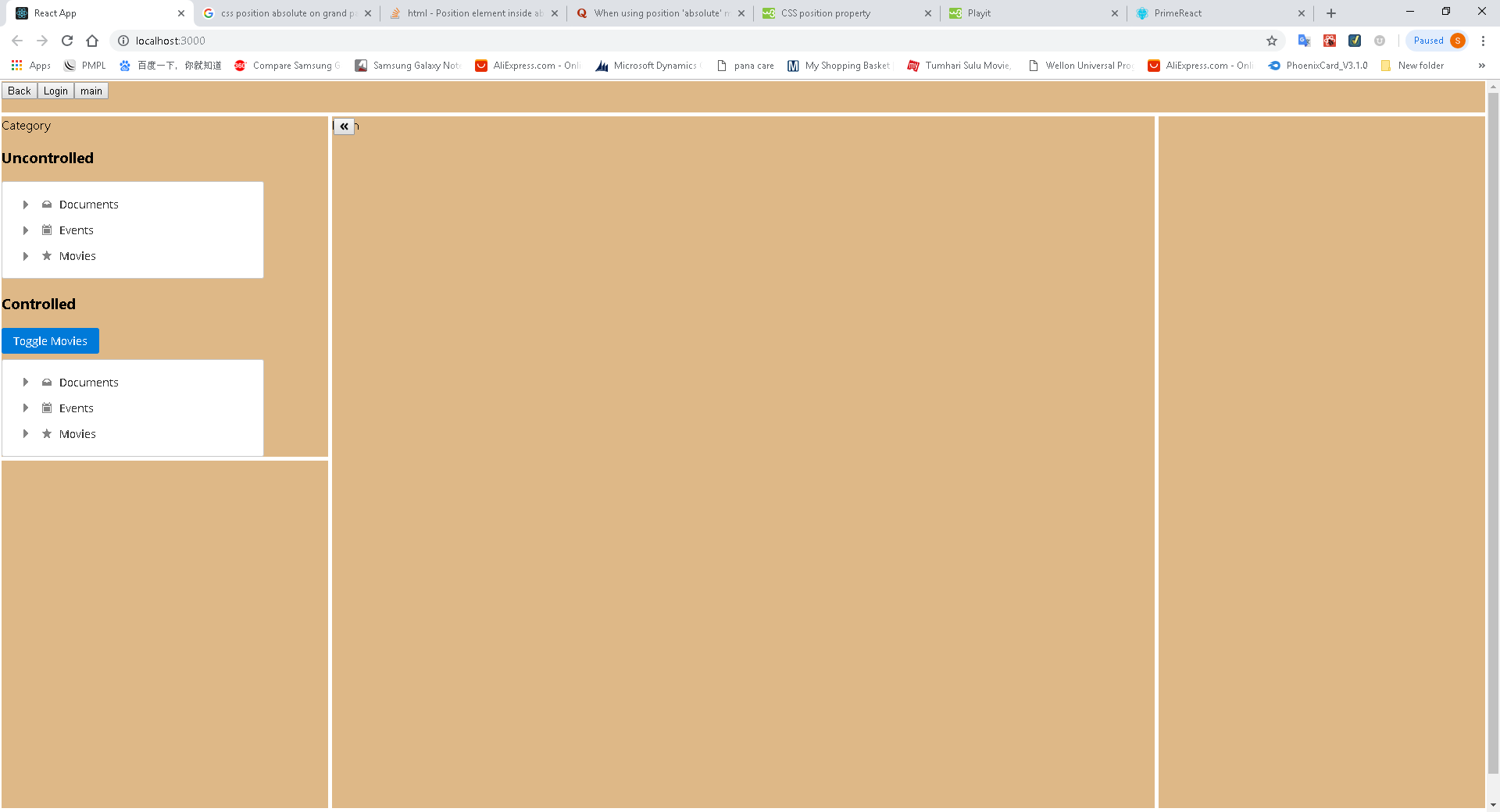
[Auto placement of items 8](#_Toc531872070)

[Properties of Grid container 8](#_Toc531872071)

[Properties of grid items 9](#_Toc531872072)

# Css techniques

* There is one container and many children as header, category, filter, main, left and footer. This is typical web page layout using grid display. The ‘main‘ child is in the middle whereas category is in the left. The layout is as follows:



See the small button in the main area. Its position is rigid. It is placed 2 pixel from top and 2 pixel from left with respect of ‘main’ child element. How to do that:

Css position attribute has possible values as

1. Static: normal flow of element
2. Absolute: placed w.r.o **positioned** parent
3. Fixed: placed in terms of viewport
4. Relative: placed w.r.o current element ‘s original position.

The ‘main’ is in ‘container’ and button is in the ‘main’ element. As above it makes sense that if we make

position:absolute

top 2px

left:2px for the button

then since the button is inside the ‘main’ then the button will be placed 2 pixels w.r.o the ‘main’ which is its immediate parent. But it does not work out.

If we do so then the button is positioned against the ‘container’ and it goes to extreme top left corner. Mind that position is applicable for **positioned parent**. Here the parent is ‘main’ but it is not positioned. So, you need to put following css in the main element to make it positioned:

position:relative; left:0;

This does not alter the placement of ‘main’ element, but it makes it a positioned element. Now the button will be placed as required, as shown in the above figure.

# Scss

## Url’s to follow

//Basics

<https://sass-lang.com/guide>

## Installation of scss

* I used visual studio extension **Live sass compiler**. In HTML file reference is given for .css files. At run time .scss files are converted to .css files. It is live.
* After installation of this VS extension the live server could be started from status bar. Live server started in port 5500.

## Variables

$ is used to denote variables.

$primary-color: #333;

## Nesting

nav {

ul {

**margin**: 0;

**padding**: 0;

**list-style**: none;

}

li { **display**: inline-block; }

a {

**display**: block;

**padding**: 6px 12px;

**text-decoration**: none;

}

}

## Partials

You can create scss partial files with “\_” prefix, like \_myFile.scss. Later on you can import these partial files in main .scss files. The partial files are not compiled to .css files

## Import

***@***import *'reset';*

Above statement imports \_reset.scss file.

## Mixins

Mixins are like functions.

Define a mixin with @mixin and use them with @include as follows:

@mixin transform($property) {

-webkit-transform: $property;

-ms-transform: $property;

transform: $property;

}

.box { @include transform(rotate(30deg)); }

## Extend / inheritance

Define a class with %className and use that call with @extend

%message-shared {

**border**: 1px solid #ccc;

**padding**: 10px;

**color**: #333;

}

**.message** {

**@extend** %message-shared;

}

## Operators

You can use +, -, \*, / and % operators

# Flex

Flex design is one dimensional and grid design is two dimensional. Flex and Grid can co-exist. Flex is much better than floats.

## Urls to follow

<https://internetingishard.com/html-and-css/flexbox/>

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

responsive design

<https://internetingishard.com/html-and-css/responsive-design/>

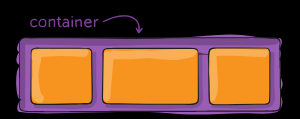
responsive images

<https://internetingishard.com/html-and-css/responsive-images/>

## Concept

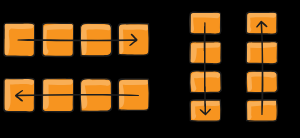
* There are containers and items in container. Any HTML tag can be made container by providing display:flex property in css. A container can in turn have many containers and many items.
* In flex mainly, container decides the layout of items inside it. The items have very little to say. Items also have limited set of properties.
* Flex container is only concerned of laying out items in it. That is containers have control over their children and not on grandchildren.
* You can have flex-direction as row or column. The justify-content and align-items are two properties of container which work on main-axis and cross-axis respectively. When flex-direction is column then justify content will work horizontally and align-items will work vertically. But when flex-direction is row then justify-content will work vertically and align-contents will work horizontally.
* Adding an order property to an item defines its order in the container without affecting surrounding items. Order is an item property and not container property.
* How to provide right and left item fixed with centre item stretchable in a container?
  + Give **width**: say 200px to right and left items and set their flex property to **initial**.
  + Give flex:1 to the middle item.
* Say in header of a web page you want to show signup and login on the right side. You can group the signup and login in a different container/ div and place that in the header container. Another way of doing this without creating an extra div is just put margin-left:auto in the signup item.
* How to provide fixed gapping between all items of a container? For container give padding: 2.5px. For items give margin: 2.5px; This will end up with uniform gapping amongst all items.

## Flex container

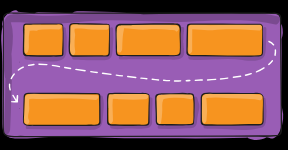


display:flex

Flex-direction: row | column | row-reverse | column-reverse



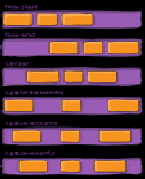
Flex-wrap: wrap | no-wrap | wrap-reverse



Flex-flow: shorthand for flex-direction and flex-wrap

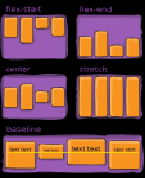
(for main axis alignment)

Justify-content: flex-start | flex-end | center | space-between | space-around | space-evenly



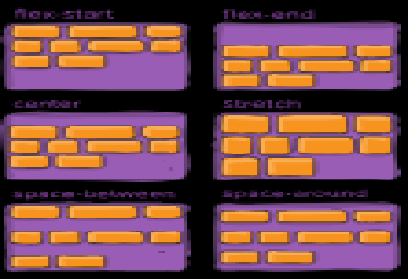
(for cross axis alignment)

Align-items: flex-start | flex-end | center | stretch | baseline

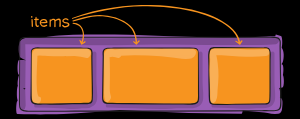


Align-contents: flex-start | flex-end | center | stretch | space-between | space-around

Align-items has no effect when there is only one line of flex items



## Flex items



Order: <integer>

Lower order items say 1 will be placed before higher order items say 2.

Flex-grow: <number>

How much an item should take the available space.

Flex-shrink:<number>

How much an item should shrink. This property overrides align-items of container.

Align-self: flex-start | flex-end …

Align-self : center | flex-start | flex-end | stretch | baseline

This will override the align-items property.

# Grid

### 30/03/2019 revisit

Good url: <https://css-tricks.com/snippets/css/complete-guide-grid/>

//Url’s

<https://gridbyexample.com/learn/>

This is advanced system and two dimensional. While designing grid layout you give display:grid or grid-inline.

You target the grid tracks and not the grid lines. That means you give the width of column tracks and width of row tracks. The line numbers are automatically generated starting from 1,2, 3 etc. Then you place your items referencing the line numbers and not the tracks.

fr unit: Fraction of available space. This is calculated after fixed spaces like pixels are subtracted.

A grid can have another grid by putting display:grid in child item. Placement of items in grid container can be done as

<div class="wrapper">

<div class="box1">One</div>

<div class="box2">Two</div>

<div class="box3">Three</div>

<div class="box4">Four</div>

<div class="box5">Five</div>

</div>

.wrapper {

display: grid;

grid-template-columns: repeat(3, 1fr);

grid-auto-rows: 100px;

}

.box1 {

grid-column-start: 1;

grid-column-end: 4;

grid-row-start: 1;

grid-row-end: 3;

}

.box2 {

grid-column-start: 1;

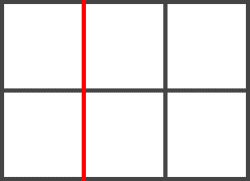
grid-row-start: 3;

grid-row-end: 5;

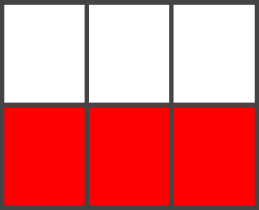
}

## Grid terminology is important to understand:

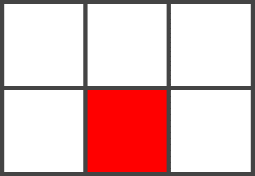
1. Grid lines:Lines that make up the grid



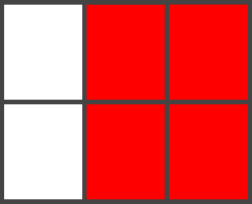
1. Grid tracks: Area between two grid lines. Horizontal or vertical



1. Grid cell: Space between 4 grid lines. This is the smallest unit in thr grid



1. Grid area: An area in grid bound by four grid lines



You can create a grid by display: grid; grid-template-columns, grid-template-rows, grid-gap a shorthand for grid-column-gap and grid-row-gap.

<div class="wrapper">  
 <div class="box a">A</div>  
 <div class="box b">B</div>  
 <div class="box c">C</div>  
 <div class="box d">D</div>  
 <div class="box e">E</div>  
 <div class="box f">F</div>  
</div>

body {  
 margin: 40px;  
}  
  
.wrapper {  
 display: grid;  
 grid-template-columns: 100px 100px 100px;  
 grid-gap: 10px;  
 background-color: #fff;  
 color: #444;  
}  
  
.box {  
 background-color: #444;  
 color: #fff;  
 border-radius: 5px;  
 padding: 20px;  
 font-size: 150%;  
}

## Explicit and implicit grid

We define a grid by defining the tracks through grid-template-rows and grid-template-columns. This automatically define the lines. This is explicit grid. If we place the items using grid-lines which lie outside the tracks or defined grid then the browser creates its own grid lines. The new grid such formed is called implicit grid which is created by browser.

## Auto placement of items

After defining a grid you have placed items in it without defining the position of items. In that case the items will be auto placed by browser by using auto-placement algorithm.

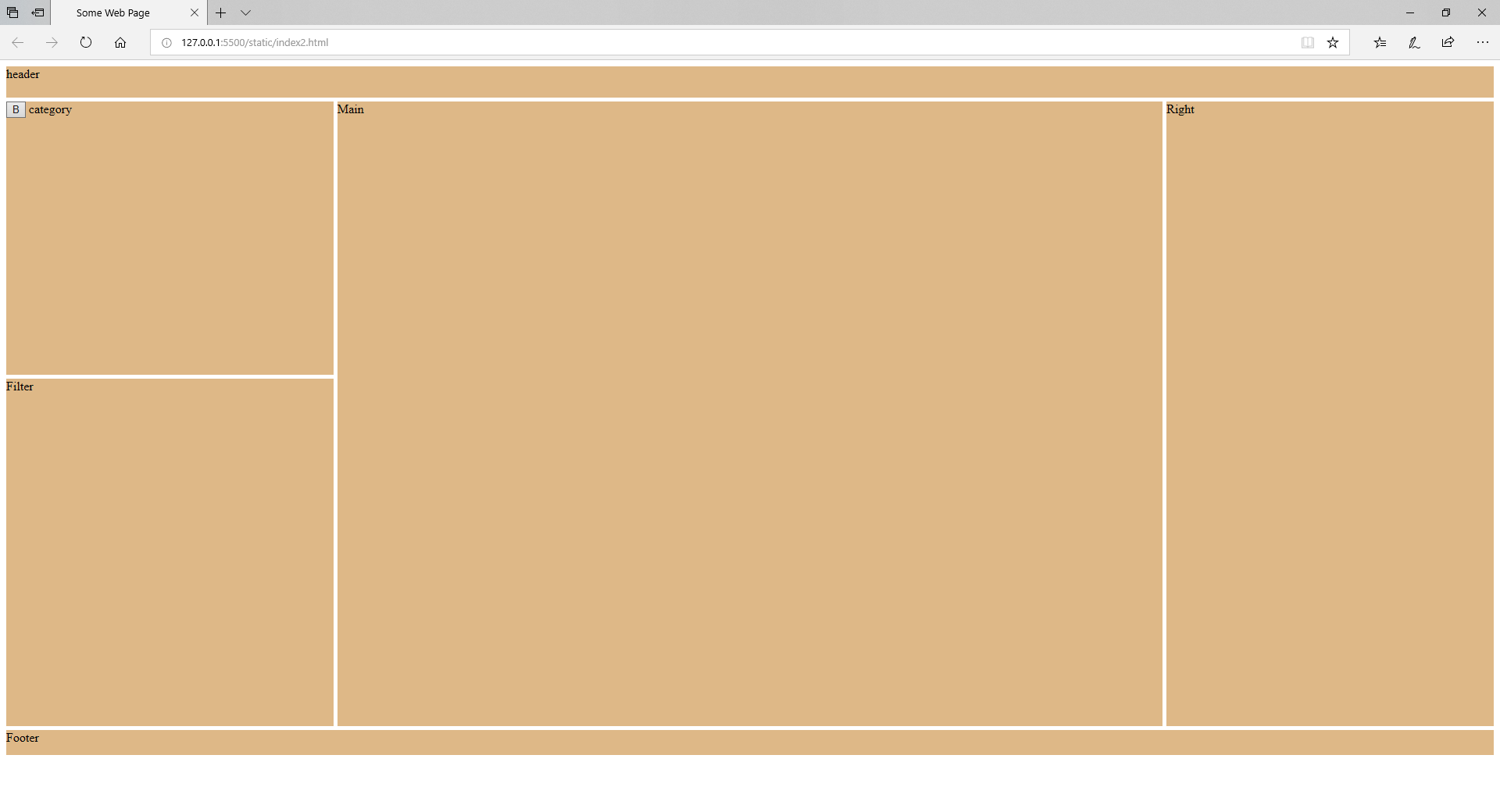
## Properties of Grid container

* [display](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-display)
* [grid-template-columns](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-template-columns-rows)
* [grid-template-rows](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-template-columns-rows)
* [grid-template-areas](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-template-areas)
* [grid-template](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-template)
* [grid-column-gap](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-column-row-gap)
* [grid-row-gap](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-column-row-gap)
* [grid-gap](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-gap)
* [justify-items](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-justify-items)
* [align-items](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-align-items)
* [place-items](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-place-items)
* [justify-content](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-justify-content)
* [align-content](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-align-content)
* [place-content](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-place-content)
* [grid-auto-columns](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-auto-columns-rows)
* [grid-auto-rows](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-auto-columns-rows)
* [grid-auto-flow](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-auto-flow)
* [grid](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid)

## Properties of grid items

* [grid-column-start](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-column-row-start-end)
* [grid-column-end](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-column-row-start-end)
* [grid-row-start](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-column-row-start-end)
* [grid-row-end](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-column-row-start-end)
* [grid-column](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-column-row)
* [grid-row](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-column-row)
* [grid-area](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-grid-area)
* [justify-self](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-justify-self)
* [align-self](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-align-self)
* [place-self](https://css-tricks.com/snippets/css/complete-guide-grid/#prop-place-self)

## divide screen in areas using Grid



I found this very effective. This way you can divide the entire page in separate areas and thereafter place controls in those areas. The above screen consists of header, category, filter main, right and footer areas. Following is the process of designing above screen:

**Step 1: Demark different area in screen and create corresponding classes in scss**

You have header, category, filter, main, right and footer areas. So create one .scss class each for these areas. Give each area a name. I have given the same name as above for each area. Define each area with **grid-area** attribute.

.header {

grid-area: header;

}

.category{

grid-area:category;

}

.filter{

grid-area: filter;

}

.main {

grid-area: main;

min-height: 800px;

}

.right {

grid-area: right;

}

.footer {

grid-area: footer;

}

**Step 2: Define container class**

All above areas reside in a single container. You need to define the container class in .scss which does the actual division of areas.

.container {

display: grid;

grid-template-columns: minmax(200px,22%) auto 22%;

grid-template-rows: 40px minmax(350px,auto) auto 32px;

grid-template-areas: 'header header header'

'category main right'

'filter main right'

'footer footer footer';

grid-gap: 5px;

}

Grid-template-columns, grid-template-rows and grid-template-areas can be taken together as grid-template

You can see in the screen layout that there are four rows (one each for header, category, filter and footer), and three columns (one each for category, main and right). So accordingly set the grid-template-columns for all columns and grid-template-rows for all rows. The minmax function is for min and max values. Attribute auto expands freely the remaining space. Grid-gap defines the distance between two grid lines. This is grid-column-gap and grid-row-gap taken together.

Grid-template-areas is typical. The areas are defined for each column and then for each row. We have 3 columns, so each entry in grid-template-area has three values corresponding to the name of area and there are 4 such entries because there are 4 rows in the layout. That is, it.

## Basics of grid layout

You divide the screen in different areas. You put the items in those areas. The defining of areas can be done in various ways. There is auto placement algorithm available in grid. Also, the placement of those items is done automatically which are not defined. Items can overlap, and their display can be controlled by z-index. Combined with flex, grid can produce outstanding layouts.

Due to auto-layout feature of grid it creates new tracks based on content. By grid-template-columns and grid-template-rows you explicitly provide the track size.

## Simple example for grid area

* grid-template-rows: 100px 200px repeat(2,1fr)

means 4 rows with 100, 200 1fraction and 1 fraction height. Span or track width are defined.

* grid-template-columns: 100px 200px repeat(2,1fr)

means 4 columns with 100, 200, 1 fraction of remaining, 1 fraction of remaining wide.

* grid-auto-rows: 100px, means row heights are 100px. Similarly, grid-auto-columns: 100px means column width are 100 px.
* The grid-auto-flow: row means items are filled in row. Similarly, a column value will indicate that items are filled in column. The default value is row.
* A very simple layout is as follows:

.container3 {

display: grid;

grid-gap: 2px;

grid-template-columns: repeat(3, 1fr);

grid-auto-rows: 100px;

}

.one {

grid-row: 1;

grid-column: 1 / 2;

}

.two {

grid-row: 1/3;

grid-column: 2/4;

}

.three {

grid-row: 2/5;

grid-column: 1;

}

.four {

grid-row: 3;

grid-column: 3;

}

.five {

grid-row: 4;

grid-column: 2;

}

.six {

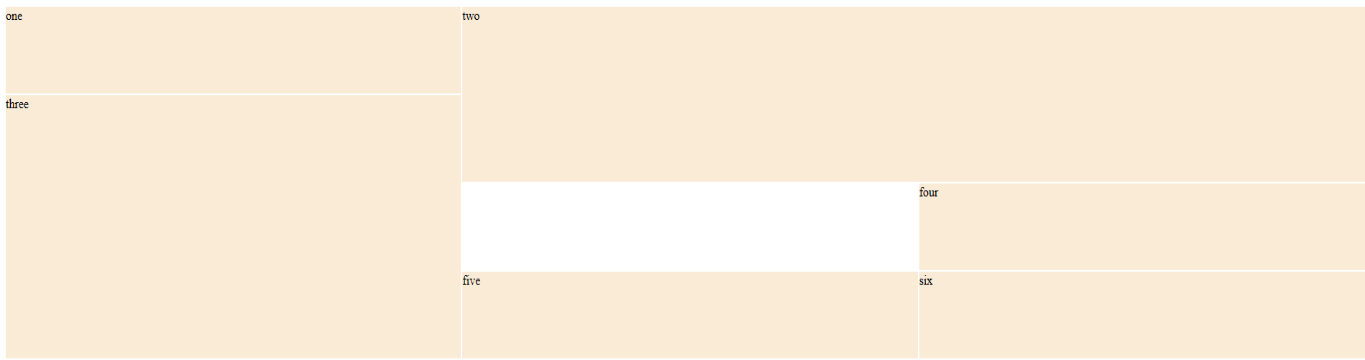
grid-row:4;

grid-column:3;

}

There are three columns from grid-template-columns. Row heights are 100px. The line numbers are automatically generated as 1,2,3,4… for rows and columns. For .one we see grid-row:1. That means row 1 with 1 span wide, since default span width is 1. The grid-column: ½ means column 1 to column 2. Only grid-column: 1 would also do.

.three has grid-row as 2/5. That means span is between row 2 and row 5 having a total span of three. Thus each class above identifies an area comprising of row and column lines. In HTML you can place items with above classes and layout will be reflected accordingly. Above layout draws following screen.



## Sample for auto product filling in grid

.container3 {

display: grid;

grid-gap: 2px;

grid-template-columns: repeat(auto-fill, 500px);

grid-auto-rows: 100px;

}

Since grid-template-rows is not defined it is open or automatic. Grid-template-columns is repeat indefinite with automatically filled with each column width as 500px. So when the browser window width is changed the items are reorganized to fulfil. This free flow product display in e-commerce site. The grid-auto-rows can be minmax(100px, auto) to have minimum height of 100 px or more if content requires more height.

## Defining areas in the grid

There are two ways of defining areas in a grid. First one is by using lines as shown in above example. Second one is by using grid-template-areas.

## Placement of items in grid

The align-items works in vertical (block) axis and justify-items work in horizontal(inline) axis. The align-items and justify-items work on all the items in the area whereas align-self and justify-self work on individual items.