

Responsive CSS Grid

Setting up Functions and Mixins

Define Grid Defaults

```
$grid-width: 550px;  
$total-columns: 12;  
$gutter-width: 20px;
```

Define a function to calculate a single column's width. The function will need to know:

- **\$grid-width**: The grid width e.g. 960px
- **\$gutter-width**: The gutter width for one column e.g. 20px (10px on either side)

These are passed in as parameters (see highlighted) to the function so they can be used for different grid width and gutter combinations.

The formula to work out the width of one column is:

(Total Grid width – Total Gutter for *all* columns) / Number of columns in Grid

```
@function grid-column-width($grid-width, $gutter-width) {  
  $total-gutter: $gutter-width * $total-columns;  
  $total-column-width: $grid-width - $total-gutter;  
  @return $total-column-width / $total-columns;  
}
```

Step 3: Create a function to work out the responsive width using the formula:

Responsive Width = target / context * 100%

- **\$target**: Target is the width to be calculated e.g. the gutter (padding) between columns
- **\$context**: Context is the width of the parent element e.g. the grid's width

```
@function rwd-width($target, $context) {  
  @return $target / $context * 100%;  
}
```

Define a mixin (reusable block of CSS styles) to generate the CSS common for each of the span column options. This mixin will need to know:

- **\$column-span**: The number of columns to span e.g. 1, 2, ..., 12
- **\$grid-width**: The grid width e.g. 960px
- **\$gutter-width**: The gutter width for one column e.g. 20px (10px on either side)

The width of a span column is calculated as follows

- Total gutter between the columns spanned + width of the columns spanned
e.g.
 - o if it spans 1 column, no gutter + 1 column width
 - o if it spans 2 columns, width of 1 gutter + 2 column widths
 - o ...
 - o If it spans 12 columns, width of 11 gutters + 12 column widths
- Convert the width to a responsive width using the rwd-width function created previously.



```
@mixin span-columns($column-span, $grid-width, $gutter-width) {  
  $total-gutter: $gutter-width * ($column-span - 1);  
  $total-col-width: grid-column-width($grid-width, $gutter-width) * $column-span;  
  width: rwd-width($total-gutter + $total-col-width, $grid-width);  
}
```

Create a mixin to generate the CSS styles for the padding placed on either side of each column. The mixin will need to know:

- **\$grid-width**: The grid width (the context)
- **\$gutter-width**: The gutter width (the target). This will need to be divided by two to get the padding on each side of the column.

Remember to convert it to a responsive width.



```
@mixin col-padding($grid-width, $gutter-width) {  
  $rwd-padding: rwd-width($gutter-width, $grid-width) / 2;  
  padding-left: $rwd-padding;  
  padding-right: $rwd-padding;  
}
```

Defining General Styles

Define style for the grid container (contains all the rows and columns)

- Position the grid centrally.
- Constrain the maximum width of the grid – set to defined grid width variable
- Constrain the minimum width of the grid.

```
.container {  
  margin: 0 auto;  
  max-width: $grid-width;  
  min-width: 300px;  
}
```

Define styles for the row class.

Set overflow to auto to fix the height problem of floating content.

```
.row {  
  max-width: 100%;  
  overflow: auto;  
}
```

Define styles for the column class, which is reused by the individual span column classes.

Set overflow to auto to fix the height problem of floating content.

```
.col {  
  float: left;  
  position: relative;  
}
```

Defining the Column Classes for the Default Media Query “xs”

Here you want to generate styles for when the content:

- Spans 1 column: col-xs-1
- Spans 2 columns: col-xs-2
- ...
- Spans 12 columns: col-xs-12

Use the @for directive to loop through the number of columns: 1 to 12

\$i is the looping variable that changes value from 1, 2, ... 12. To use a dynamic variable in Sass, you enclose it in #{...} – see highlighted (will be replaced by the current value of \$i)

```
@for $i from 1 through $total-columns {  
  .col-xs-#{ $i } {  
    @extend .col;  
    @include col-padding($grid-width, $gutter-width);  
    @include span-columns($i);  
  }  
}
```

Create the styles for col-xs-1 ... col-xs-12 using the mixins defined earlier to generate the respective padding and widths.

```
.col-xs-pull-#{ $i }, .col-pull-#{ $i } {  
  right: rwd-width($i, $total-columns);  
  position: relative;  
}  
  
.col-xs-push-#{ $i }, .col-push-#{ $i } {  
  left: rwd-width($i, $total-columns);  
  position: relative;  
}  
}
```

Create a set of pull and push styles for changing the order of column content for all the grid columns col-xs-push-1 ... col-xs-push-12 and col-xs-pull-1 ... col-xs-pull-12.

Pull moves a column to the left by pushing its right side ←

→ Push moves a column to the right by pushing its left side

Just some test styles used to test the grid

```
.test-grid-col {  
  min-height: 50px;  
  border: 1px dotted #808080;  
  background-color: antiquewhite;  
}  
  
.test-grid > div.row {  
  margin-top: 10px;  
  margin-bottom: 10px;  
}
```

Defining the other Media Query Classes – sm, md and lg

Create a reusable mixin to create the styles for the different media queries. This mixin will need to know the following:

- **\$viewport-min-width:** The minimum viewport to target (used to set up the media query)
- **\$grid-width:** The grid width for the specified viewport
- **\$gutter-width:** The gutter width for the specified viewport (you may want smaller gutters for smaller grids and larger ones for larger grids).
- **\$grid-size:** The media query class, i.e. sm, md, lg

```
@mixin rwd-grid($viewport-min-width, $grid-width, $gutter-width, $grid-size) {
```

```
  @media(min-width: $viewport-min-width) {
```

Set up the media query based on the specified minimum viewport width parameter

```
    .container {
      max-width: $grid-width;
    }
```

Set the grid maximum width for this media query

Here you want to generate styles for when the content:

- Spans 1 column: col-sm-1, col-md-1, col-lg-1
- Spans 2 columns: col-sm-2, col-md-2, col-lg-2
- ...
- Spans 12 columns: col-sm-12, col-md-12, col-lg-12

Use the @for directive to loop through the number of columns: 1 to 12

\$i is the looping variable that changes value from 1, 2, ... 12. This time there are two dynamic

- #{ \$i } will be replaced by the current value of \$i, the looping variable
- #{ \$grid-size } will be replaced by the \$grid-size parameter passed into the mixin.

```
@for $i from 1 through $total-columns {
  .col-#{ $grid-size }-#{ $i } {
    @include col-padding($grid-width, $gutter-width);
    float: left;
    position: relative;
    @include span-columns($i, $grid-width, $gutter-width);
  }
}
```

Create the styles for col-sm-1, ... col-xs-12, col-md-1 ... col-md-12 and col-lg-1 ... col-lg-12 respectively using the mixins defined earlier to generate the respective padding and widths.

```

.col-#{$grid-size}-pull-#{$i} {
  right: rwd-width($i, $total-columns);
}

.col-#{$grid-size}-push-#{$i} {
  left: rwd-width($i, $total-columns);
}
}
}

```

Create a set of pull and push styles for changing the order of column content for all the grid columns and media query classes (sm, md, lg) e.g. col-sm-push-1 ... col-sm-push-12 and col-sm-pull-1 ... col-sm-pull-12.

Pull moves a column to the left by pushing its right side ←

→ Push moves a column to the right by pushing its left side

Finally, use the responsive grid mixin, rwd-grid, to generate the grid classes for the different media query classes:

- sm: Minimum viewport (770px) Grid width (750px) Gutter width (20px)
- md: Minimum viewport (980px) Grid width (960px) Gutter width (20px)
- lg: Minimum viewport (1200px) Grid width (1170px) Gutter width (30px)
-

↓

```

@include rwd-grid(770px, 750px, 20px, "sm");
@include rwd-grid(980px, 960px, 20px, "md");
@include rwd-grid(1200px, 1170px, 30px, "lg");

```