CSS Grid Introduction

CSS Grid is designed to simplify laying out webpages using CSS. This is a dramatic improvement over what we've used in the past.

CSS Grids:

- designed to accomodate complex layouts
- are language agnostic (work with right-to-left)
- can be nested

There are three basic steps for creating a CSS Grid layout:

1. Establish a grid container
2. Set up columns and rows
3. Place page elements on the grid

The Grid container

Just like with flexbox, a grid is created by changing the display property of a parent element. This element becomes the *grid container* and defines the context for grid formatting.

To create a grid container change the `display` property to `grid`:

```
.grid-container {
   display: grid
}
```

I'm using a class of 'grid-container' in this example, but we could use any element, class or id for the grid container.

Once we declare a grid container, all immediate children of that element become *grid items*. Only children, not descendents, are grid items.

Grid Terms

The primary parts of a grid are:

- Grid lines
- grid cell and grid areas
- grid tracks (rows or columns)

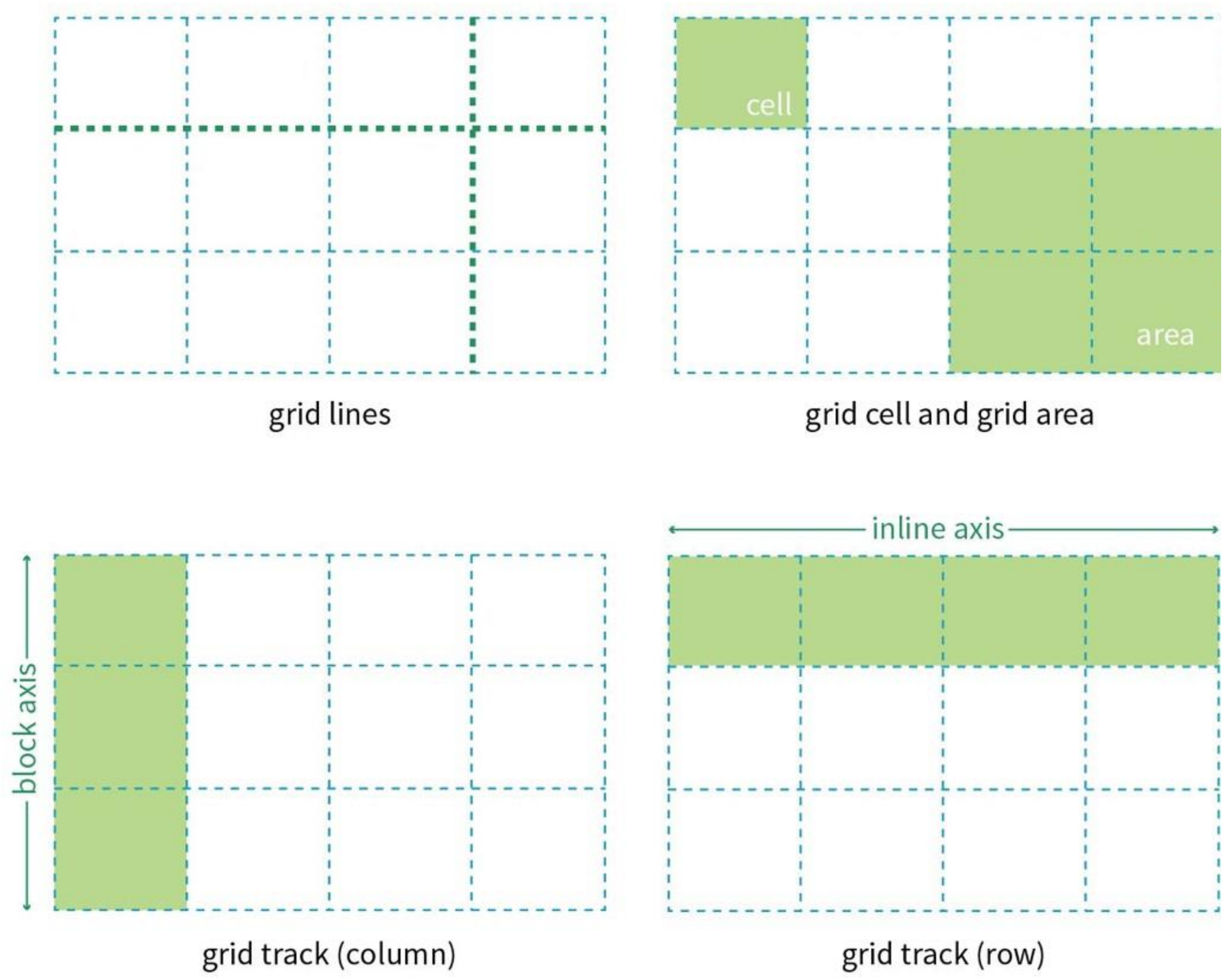


FIGURE 16-30. The parts of a CSS grid.

Figure 16-30 from *Learning Web Design*

Definitions from *Learning Web Design*

Grid line
: The horizontal and vertical dividing lines of the grid are called grid lines.

Grid cell
: The smallest unit of a grid is a grid cell, which is bordered by four adjacent grid lines with no grid lines running through it.

Grid area
: A grid area is a rectangular area made up of one or more adjacent grid cells.

Grid track
: The space between two adjacent grid lines is a grid track, which is a generic name for a grid column or a grid row. Grid columns are said to go along the block axis, which is vertical (as block elements are stacked) for languages written horizontally. Grid rows follow the inline (horizontal) axis.

Create Grid tracks (rows and columns)

Grid rows and columns are known as tracks. In order for our grid to work like a grid, we have to establish sizing rules for our rows and columns.

Track sizes (rows and columns) can be specified using any of the following values:

- Fixed measurements (such as px or em)

- Percentage values (%)
- Fractional units (fr)
- auto
- min-content, max-content
- minmax()
- fit-content()

The values can be applied to either of the grid-template-* properties:

- `grid-template-rows`
- `grid-template-columns`

Fractional Units (fr)

Fractional units are a new unit designed specifically for CSS grids. These units can be considered a ratio.

The following example will set the middle column to be twice the size of the first and third columns:

```
.grid-container {
   display: grid
   grid-template-columns: 1fr 2fr 1fr;
}
```

This example makes each column proportionally larger:

```
.grid-container {
   display: grid
   grid-template-columns: 1fr 2fr 3fr;
}
```

Best of all, fr units can be mixed with fixed units such as pixels and ems. This is great for maintaining fixed content areas for things like ads.

The browser first assigns space for the fixed measurements and then calculates the fr units.

```
.grid-container {
   display: grid
   grid-template-columns: 1fr 2fr 250px;
}
```

minmax()

The minmax() function can be used in place of a value to define both the minimum size and maximum size of a grid track. *You cannot use fr units inside the minmax() function.*

```
.grid-container {
   display: grid
   grid-template-columns: min-content minmax(20em, 50em) max-content;
}
```

min-content, max-content and auto

min-content
    Sets the track size to the smallest size possible without breaking content. Ex: longest word
max-content
    Sets the track size to the length needed for the longest content **without wrapping text**
auto
    Let the browser decide how big things should be
minmax(auto, auto)
    Works similarly to min-content and max-content except is smarter about wrapping and sizing

```
.grid-container {
   display: grid
   grid-template-columns: 1fr minmax(20em, 50em) 250px;
}
```

fit-content()

fit-content() will use a formula to allow a track to reflow to the minium size when needed but not display larger than the fit-content value.

In this example, the second column will get smaller as needed, but not expand greater than 300px.

```
.grid-container {
   display: grid
   grid-template-columns: 1fr fit-content(300px) 2fr;
}
```

repeat(), auto-fill, and auto-fit

The repeat function allows you to specify repeating patterns of columns.

For example, imagine you wanted 10 columns that alternated between 1fr and 2fr. You could write this:

```
.grid-container {
   display: grid
   grid-template-columns: 1fr 2fr 1fr 2fr 1fr 2fr 1fr 2fr 1fr 2fr;
}
```

Using repeat() simplifies this. The first value is the number of times to repeat the pattern while the values after comma demonstrate the patter.

```
.grid-container {
   display: grid
   grid-template-columns: repeat(5, 1fr 2fr);
}
```

auto-fill and auto-fit

The auto-fill and auto-fit values are quite similar to each other. Both tell the browser to adjust the tracks so that the pattern fits based on the specified height or width. auto-fit will discard tracks without content.

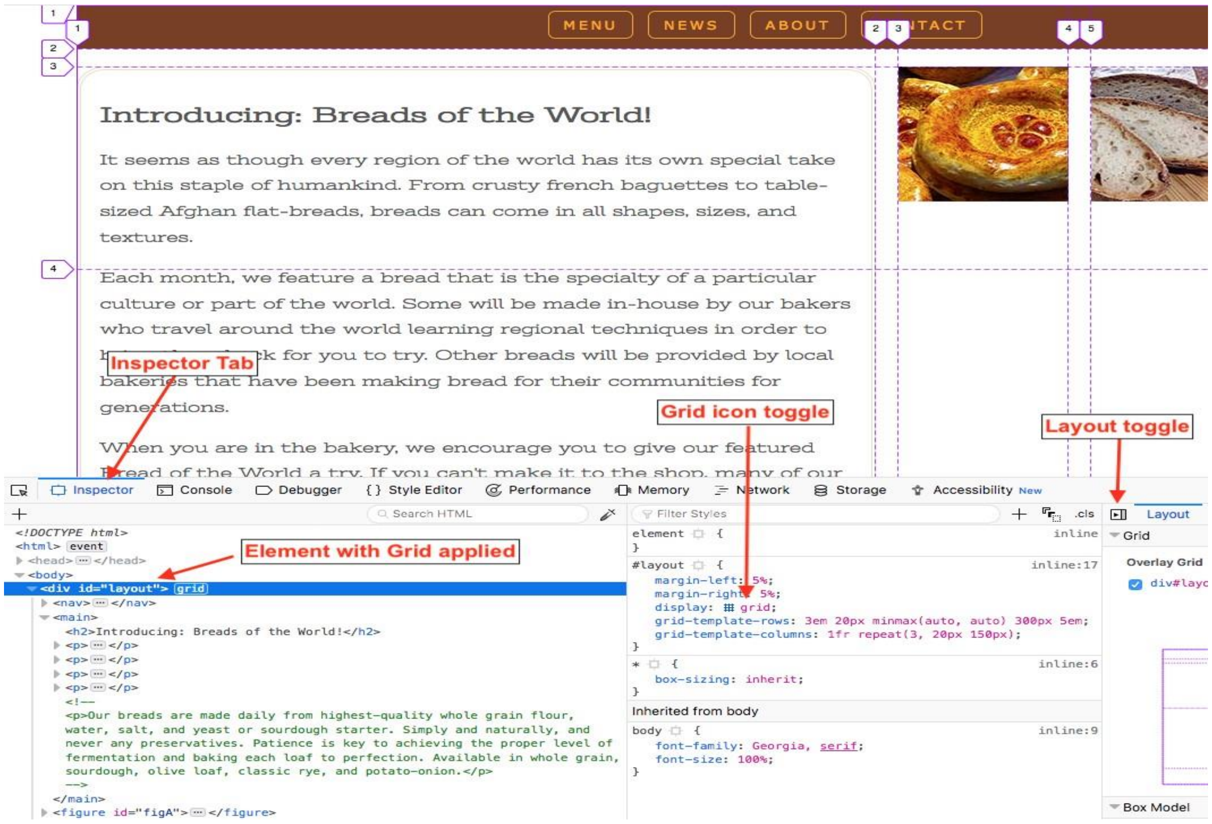
Placing Elements in the Grid

So far we've just seen how to define grids and the browser slots each grid item in one by one. To make grids more useful, we also need to define where and how to place items in the grid.

Firefox Grid Developer Tools

Using Firefox, you can see a visual overlay of any grid by using the developer tools in Firefox. Right now, Firefox has the best developer tools for CSS Grid.

1. In Firefox open a page using a grid layout.
2. Go to Tools > Web Developer > Toggle Tools to turn on the tools.
3. Select the Inspector tab
4. With the element using the grid selected you can click the grid icon in the rules pane to toggle the grid lines on and off
5. More features are available by expanding the layout pane into a 2 column view



Introducing: Breads of the World!

It seems as though every region of the world has its own special take on this staple of humankind. From crusty french baguettes to table-sized Afghan flat-breads, breads can come in all shapes, sizes, and textures.

Each month, we feature a bread that is the specialty of a particular culture or part of the world. Some will be made in-house by our bakers who travel around the world learning regional techniques in order to ck for you to try. Other breads will be provided by local bakeries that have been making bread for their communities for generations.

When you are in the bakery, we encourage you to give our featured Bread of the World a try. If you can't make it to the shop, many of our

```
<!DOCTYPE html>
<html> event
  <head>⋯</head>
  <body>
    <div id="layout"> grid
      <nav>⋯</nav>
      <main>
        <h2>Introducing: Breads of the World!</h2>
        <p>⋯</p>
        <p>⋯</p>
        <p>⋯</p>
        <p>⋯</p>
        <!--
        <p>Our breads are made daily from highest-quality whole grain flour,
        water, salt, and yeast or sourdough starter. Simply and naturally, and
        never any preservatives. Patience is key to achieving the proper level of
        fermentation and baking each loaf to perfection. Available in whole grain,
        sourdough, olive loaf, classic rye, and potato-onion.</p>
        -->
      </main>
      <figure id="figA">⋯</figure>
```

```
element {                                                   inline
}
#layout {                                                   inline:17
   margin-left: 5%;
   margin-right: 5%;
   display: grid;
   grid-template-rows: 3em 20px minmax(auto, auto) 300px 5em;
   grid-template-columns: 1fr repeat(3, 20px 150px);
}
* {                                                         inline:6
   box-sizing: inherit;
}
Inherited from body
body {                                                      inline:9
   font-family: Georgia, serif;
   font-size: 100%;
}
```

Firefox Grid Developer Tools

Place elements using grid lines

Elements can be placed on the grid with the following properties:

- `grid-row-start`
- `grid-row-end`
- `grid-column-start`
- `grid-column-end`

The start and end values are identified by counting the grid lines in the grid.

Use Firefox to see a visual overlay of the grid lines.

These values should be applied to individual grid items.

In this example:

- the header should stretch across all the columns and fit into 1 row
- the header should stretch across all the columns and fit into 1 row
- each aside should fill one grid column
- main should fill the two right columns and the middle row

Using the criteria above, we can lay out the `header` with the following CSS:

```
header {
   grid-column-start: 1;


   grid-column-end: 4;
   grid-row-start: 1;
   grid-row-end: 2;
}
```

Grid lines can also be counted backwards, using negative numbers. The last line is -1, the second to last line is -2, etc.

When specifying elements to span across all columns or rows, it's easier and more versatile to use -1 for the end value. This avoids counting errors and will work even if the grid lines change.

```
header {
   grid-column-start: 1;
   grid-column-end: -1;
   grid-row-start: 1;
   grid-row-end: 2;
}
```

Shorthand Properties

In additional to using the start and end properties for rows and columns, you can also use shorthand values instead to save typing.

- `grid-row`
- `grid-column`

We can rewrite the header properties above using this shorthand:

```
header {
   grid-column: 1 / -1;
   grid-row: 1 / 2;
}
```

Lastly, there is one more shorthand to apply all of the properties.

The order is: "row-start" "column-start" "row-end" "column-end".

This still follows the general CSS rule for listing values in a clockwise order.

```
header {
   grid-area: 1 / 1 / 2 / -1;
   /* row-start / column-start / row-end / column-end */
}
```

Place elements using named areas

CSS Grid allows you to name both grid lines and areas.

Naming grid lines is covered in Chapter 16 of *Learning Web Design*

We could define a grid like so:

```
.grid-container {
   display: grid;
   grid-template-columns: 200px 1fr 200px;
   grid-template-rows: 100px 400px 100px;
}
```

The CSS property `grid-template-areas` allows you to name individual grid cells or groups of cells called areas in your grid. These names can be referenced later for placing elements.

Drawing a quick grid helps to visualize this.

We'll use the figure below to set up our names.

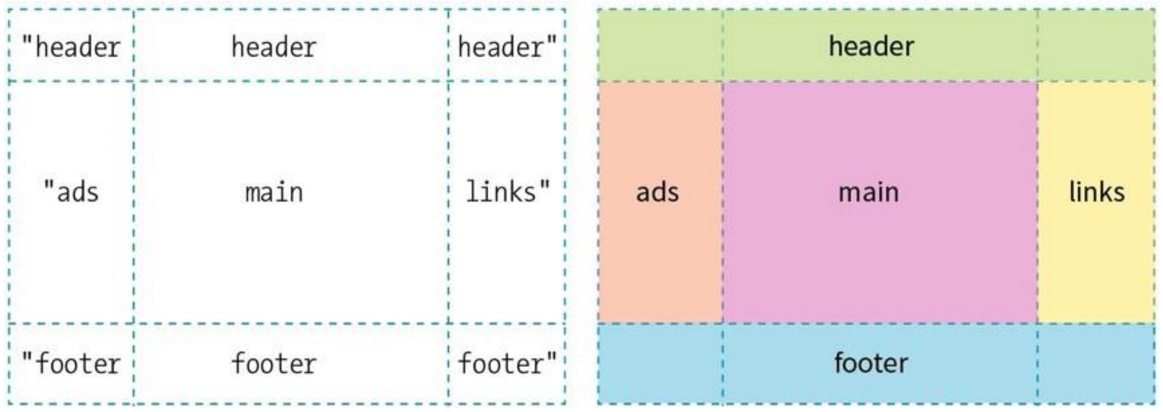


FIGURE 16-36. When neighboring cells have the same name, they form a named area that can be referenced later.

With `grid-template-areas`:

- each row is a string of text in quotes
- each column is separated by 1 or more spaces
- every cell must be accounted for
- to leave a cell unnamed, use a period instead of a name
- grid areas must be rectangular
- multiple adjacent cells with the same name will form a named area with that name

The CSS to create the named areas above would be this:

```
.grid-container {
   display: grid;
   grid-template-columns: 200px 1fr 200px;
   grid-template-rows: 100px 400px 100px;

   grid-template-areas:
      "header header header"
      "ads     main    links"
      "footer  footer  footer"
   ;
}
```

It helps to align the values on different lines to make the grid more clear

By itself, the page should change. You can hover use the Firefox developer tools to view the grid names.

-

Placing elements using area names

The named areas can be laid out using `grid-area`

```
header {
   grid-area: header;
}

main {
   grid-area: main;
}

aside:first-of-type {
   grid-area: links;
}

aside:last-of-type {
   grid-area: ads;
}

footer {
   grid-area: footer;
}
```