CSS Grid Layout

Mohamed Emary

December 1, 2023

1 Grid Layout

1.1 Units & CSS Functions

Grid layout is used to create page layouts through a series of rows and columns. It is a newer layout system that is not supported by older browsers.

first you should use `display: grid;` to make the element a grid container. Then you can use `grid-template-columns` and `grid-template-rows` to define the columns and rows of the grid. You can also use `grid-template-areas` to define areas of the grid.

This is the syntax for defining the columns and rows:

```
/* you can use px, em, rem, %, fr, auto */

/* This defines 3 columns with 100px width each */
grid-template-columns: 100px 100px 100px;

/* This defines 3 columns with 100px width each and 2 rows with 50px height each */
grid-template-columns: 100px 100px 100px;
grid-template-rows: 50px 50px;

/* This creates 3 columns with 30% 50% and 20% width each and 3 rows with auto height each */
grid-template-columns: 30% 50% 20%;
grid-template-rows: auto auto auto;

/* This creates 2 columns with percentages 2 and 1 */
grid-template-columns: 1fr 1fr; /* you can't do that with auto */

/* you can achieve the same result as before using % but sometimes it's hard to calculate the percentages like below so fr becomes handy and it's still not 100% accurate as fr */
grid-template-columns: 66.6666% 33.3333%;

/* you can also combine units */
grid-template-columns: 100px 1fr 2fr;

/* auto fits the content of the column or row unless the free space is not occupied by other columns or rows */
/* Here the first column with auto will fit the content of the text inside it and the second column will take the rest of the space */
grid-template-columns: auto 1fr;
/* Here the first column will take the whole space except for the space occupied by the second column content width only */
grid-template-columns: 1fr auto;
/* Here the first column will have 100px width and the second column will take the rest of the space */
/* both of the examples below do the exact same thing */
grid-template-columns: 100px auto;
grid-template-columns: 100px 1fr;
```

```
32
33  /* you can also use repeat to repeat a value */
34  /* This will create 3 columns with 100px width each */
35  grid-template-columns: repeat(3, 100px);
36
37
38  /* You can also specify the height of each row alone */
39  grid-template-rows: 100px 150px 50px;
40
41  /* you may also have many rows say 5 but you just specify the height of 3 of them
    ↪  and the rest will be the default which is auto so they will fit the height of
    ↪  the content */
42  grid-template-rows: 100px 150px 50px; /* we-have-five-columns */
43
44
45  /* you can also use minmax */
46  /* This will create 2 columns one with 200px width and the other will expand to
    ↪  fit the remaining space */
47  grid-template-columns: minmax(100px, 200px) auto;
48
49  /* this will create 2 columns one with 100px width and the other will expand to
    ↪  fit the remaining space */
50  grid-template-columns: minmax(100px, 200px) 1fr;
51
52  /* if we have a grid container with 250px width this will overflow the grid
    ↪  container by 50px because the minimum width of the column is 200px and the
    ↪  other column has a width of 100px so the total width is 300px */
53  grid-template-columns: minmax(200px, 300px) 100px;
54
55  /* this will make a row with minimum height of 100px and maximum height of the
    ↪  content height so if the content need a height greater than 100px it will
    ↪  expand to fit the content height. */
56  grid-template-rows: minmax(100px, auto)
57
58  /* This will create as many columns with 100px width as it can fit in the
    ↪  available space */
59  grid-template-columns: minmax(100px, auto-fill)
60
61
62  /* This will create as many columns with 100px width as it can fit in the
    ↪  available space */
63  grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
64  /* does the same as above */
65  grid-template-columns: repeat(auto-fill, minmax(100px, auto));
66
67  /* the gap between the columns and rows can be specified using the gap property
    ↪  */
68  /* row-gap is the vertical gap between rows */
69  row-gap: 10px;
70  /* column-gap is the horizontal gap between columns */
```

```
column-gap: 10px;
/* with gap you can specify both row-gap and column-gap in one line */
/* 10px for row gap and 15px for column gap */
gap: 10px 15px;
/* with only one value the row gap and column gap will be the same */
gap: 10px;

/* ================================= */

/* The shorthand property for grid-template-columns and grid-template-rows is
   grid-template */
/* This will create three rows, each with a height of 100 pixels, and two
   columns, each with a width of 50 pixels. The values before and after the "/"
   represent the rows and columns, respectively. */
grid-template: 100px 100px 100px / 50px 50px;
```

`auto` will automatically size the column or row to fit the content.

`fr` will divide the space into parts that have width or height depending on the fraction specified

`repeat(number_of_times, value)` will repeat the value the number of times specified

`minmax(minimum, maximum)` will specify the minimum and maximum width of the column or height of the row (they can be in the range between the minimum or maximum too not only take the minimum or maximum values). So in summary `minmax` will allow a column or row to expand to the maximum point if there is enough space and it will shrink to the minimum point if there is not enough space.

you can even combine functions for example `grid-template-columns: repeat(3, minmax(100px, 1fr));`

`auto-fill` will create as many columns or rows as it can fit in the container without overflowing it.

`gap: row-gap column-gap;` will specify the gap between the rows and columns respectively.

1.2 Items Alignment

`justify-content` will align the items horizontally, the values for `justify-content` are:

1. `start` will align the items to the start of the row. `start` is the default value.
2. `end` will align the items to the end of the row
3. `center` will align the items to the center of the row
4. `space-around` will align the items with space around them
5. `space-between` will align the items with space between them
6. `space-evenly` will align the items with space evenly between them

```
/* justify-content will align the items horizontally */
/* unlike flexbox justify-items will always work on the horizontal and not the
   main axis */
/* the default value is stretch */
justify-content: start;

/* the code above will not have an effect if the width of the columns is
   specified by fr units for example */
/* because the items will take the whole width of the column so you will not
   notice the effect of the justify-items property */
```

```
grid-template-columns: 1fr 1fr 1fr;
```

`align-content` will align the items vertically, the values for `align-content` are:

1. `stretch` will stretch the items to fill the whole height of the row. `stretch` is the default value.
2. `start` will align the items to the start of the row.
3. `end` will align the items to the end of the row
4. `center` will align the items to the center of the row
5. `space-around` will align the items with space around them
6. `space-between` will align the items with space between them
7. `space-evenly` will align the items with space evenly between them

// review this code

```
/* align-content will align the items vertically */
/* unlike flexbox align-items will always work on the vertical and not the cross
↪  axis */
/* the default value is stretch */
align-content: start;

/* the code above will not have an effect if the height of the rows is specified
↪  by fr units for example */
/* because the items will take the whole height of the row so you will not notice
↪  the effect of the align-items property */
grid-template-rows: 1fr 1fr 1fr;
```

1.3 Merging Cells

In CSS Grid Layout you can merge cells using `grid-column-start`, `grid-column-end`, `grid-row-start`, `grid-row-end` properties.

We also have `grid-column` and `grid-row` properties that are shorthand for `grid-column-start`, `grid-column-end`, and `grid-row-start`, `grid-row-end` respectively.

```
/* this will merge the first cell with the second cell */
grid-column-start: 1;
grid-column-end: 3;

/* if you don't know the number of columns you can use -1 to specify the last
↪  column */
grid-column-start: 1;
grid-column-end: -1;

/* you can also use the shorthand property */
/* this will merge the first cell with the second cell */
grid-column: 1 / 3;

/* you can even apply both properties to the same cell */
/* this will merge the first cell with the second cell on the first row */
grid-column: 1 / 3;
/* This will merge the first cell with the second cell on the first column */
grid-row: 1 / 2;

```

```
/* all the above applies to rows too */
```

You can also use Grid Areas to merge cells.

```

/* A parent container with 3x3 cells 3 rows and 3 columns */
.parent {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-template-areas:
    "header header header"
    "nav main sidebar"
    "nav main sidebar"
    "footer footer footer";
}

.header {
  grid-area: header;
}

.nav {
  grid-area: nav;
}

.main {
  grid-area: main;
}

.sidebar {
  grid-area: sidebar;
}

.footer {
  grid-area: footer;
}


/* you can eve have an empty cell by specifying a dot */
.parent {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-template-areas:
    "header header ."
    "nav main sidebar"
    "nav . sidebar"
    "footer footer footer";
}
```

2 Notes From Odin Lessons

only the direct child elements will become grid items here. If we had another element as a child under one of these child elements it would not be a grid item.

But just as you learned in the flexbox lessons, grid items can also be grid containers. So you could make grids inside of grids if you wanted.

If we have 4 items inside a 2x2 grid and we added a 5th item it will be placed in the 3rd row even if the layout we specified have only 2 rows. This is because of the implicit grid concept and it's how CSS Grid is able to automatically place grid items when we haven't explicitly defined the layout for them.

When we use the `grid-template-columns` and `grid-template-rows` properties, we are explicitly defining grid tracks to lay out our grid items. But when the grid needs more tracks for extra content, it will implicitly define new grid tracks. Additionally, the size values established from our `grid-template-columns` or `grid-template-rows` properties are not carried over into these implicit grid tracks. But we can define values for the implicit grid tracks.

We can set the implicit grid track sizes using the `grid-auto-rows` and `grid-auto-columns` properties. In this way we can ensure any new tracks the implicit grid makes for extra content are set at values that we defined. For ex:

```
.container {
  display: grid;
  grid-template-columns: 50px 50px;
  grid-template-rows: 50px 50px;
  grid-auto-rows: 50px;
  /* by default the CSS Grid will add additional content with implicit rows and
   ↪  if we want columns instead we can use: */
  grid-auto-flow: column;
  /* To set the implicit track sizes for columns we can use: */
  grid-auto-columns: 50px;
}
```

By default, CSS Grid will add additional content with implicit rows. This means the extra elements would keep being added further down the grid in a vertical fashion. It would be much less common to want extra content added horizontally along the grid, but that can be set using the `grid-auto-flow: column` property and those implicit track sizes can be defined with the `grid-auto-columns` property.

Lesson Assignment

The main difference between `display: inline-grid` and `display: grid` is how they behave in the document flow:

`display: grid` - Makes the element a block-level grid container, meaning it starts on a new line and takes up the full width available (stretches out to the edge of its parent container).

`display: inline-grid` - Makes the element an inline-level grid container, meaning it can flow inline with text and other inline elements. The width is only as large as necessary for its content (like any inline element).

Some key differences:

- `inline-grid` can sit next to text or other inline boxes on a line, `grid` cannot. It always starts a new line.
- `inline-grid` cannot have a width and height set, as it sizes to its content like inline elements. `grid` can have an explicit width/height set.
- Margins, padding, etc. behave differently on `inline-grid` vs `grid` due to the former being inline-level.
- Line breaks and alignment function differently with `inline-grid` vs `grid`.