**FLEXBOX PRINCIPLES**

To use the flex box we use the display properties. Display flex to an element turns it into a flex container, and its direct children into flex items. By default, flex items are laid side by side, left to right, all in one row. The flex container fills available width like a block element, but the flex items may not necessarily fill the width of the flex container. The flex items are all of same height, naturally by the contents.

A flex container asserts the control over the layout of its elements within.

(diagram of flexbox)

The items are placed along a line called the main axis, which goes from the main-start (left) to the main-end (right). Perpendicular to the main axis is the cross-axis this goes from cross-start (top) to the cross-end (bottom). The direction of these axes can be changed .Let’s create an example to see the flex box.

**USING THE display: flex PROPERTY**

Let’s build a menu box which has 4 items.

1. Plain HTML leads us to 4 <ul> items.

<ul class="site-nav">

<li><a href="/">Home</a></li>

<li><a href="/features">Features</a></li>

<li><a href="/pricing">Pricing</a></li>

<li><a href="/support">Support</a></li>

<li class="nav-right">

<a href= "/about"> About </a>

</li>

</ul>

1. Let’s apply flex property display: flex to these elements

(show code here)

1. Let’s add padding to give it look and feel

(show code here)

1. Add margins between elements to space the items. We apply a margin between each item, butnot to the outside edges. You can achieve this layout by using the margin-left property and an adjacent sibling combinatory.

**USING THE flex PROPERTY**

The flex property is shorthand for three different sizing properties: flex-grow, flex-shrink, and flex-basis. So to set the defaults we can provide flex: default which can also be written as “flex: 0 1 auto”. In this case the value of flex-grow is 0, so items will not grow larger than their flex-basis size. The value of flex-shrink is 1, so items can shrink if they need to rather than overflowing. The value of flex-basis is auto which means items will either use any size set on the item in the main dimension, or they will get their size from the content size.

Each of the property is explained in detail below.

**Flex-basis:**

The flex basis defines main size for an element.The flex-basis property can be set to any value that would apply to width, including values in px, ems, or percentages. Its initial value is auto, which means the browser will look to see if the element has a width declared. If so, the browser uses that size; if not, it determines the element’s size naturally by the contents.This means that width will be ignored for elements that have any flex basis other than auto.

(show diagram here)

**Flex-grow:**

Once flex-basis is computed for each flex item, they (plus any margins between them) will add up to some width. This width may not necessarily fill the width of the flex container, leaving a remainder. The remainder will be consumed by the flex items based on their flex-grow values, which is always specified as a non-negative integer. If an item has a flex-grow of 0, it won’t grow past its flex basis. If any items have a non-zerogrowth factor, those items will grow until all of the remaining space is used up. This means the flex items will fill the width of the container.

(show diagram here)

**Flex-shrink:**

The flex-shrink value for each item indicates whether it should shrink to prevent overflow. If an item has a value of flex-shrink: 0, it will not shrink. Items with a value greater than 0 will shrink till there is no overflow. An item with a higher value will shrink more than an item with a lower value, proportional to the flex-shrinkvalues.

**Flex-direction:**

Another important option in flexbox is the ability to shift the direction of the axes. The flex-direction property, applied to the flex container, controls this. Its initial value (row) causes the items to flow left-to-right, as we’ve done. Specifying flex-direction: column causes the flex items to stack vertically (top to bottom) instead. Flexbox also supports row-reverse to flow items right to left, and column-reverse toflow items bottom to top.

Let’s use the above property to build a section based content

1. First we apply display: flex to the 3 sections and background color as white. We then see it doesn’t occupy the same width as the navbar.

(show code here line 29 - line 35)

1. use the column-main and column-sidebar classes to target the columns, using flex to apply widths of twothirdsand one-third respectively in the stylesheet

(show code here line 36 – line 42)

1. Let’s split the sections by applying a left margin to all the children of class flex.

(line 44 - line 47). This splits our main section from the side section.

1. What you need is for the two columns to grow if necessary to fill the container’s height. To do this, turn the right column (the column-sidebar) into a flex container with a flex-direction: column. Then, apply a non-zero flex-grow value toboth tiles within.

(line 49 –line 56)

With this we have a 3 section layout with will resize as the main content tile expands. So this is a gentle introduction to flexbox as it has other properties too which can be checked out in the docs.

Now let’s see the other layout which is the CSS grid. The CSS grid lets you define a two-dimensional layout of columns and rows and then place elements within the grid. Some elements may only fill one cell ofthe grid; others can span multiple columns or rows. The size of the grid can be defined precisely, or you can allow it to automatically size itself as needed to fit the contents within. You can place items precisely within the grid, or allow them to flow naturally to fill in the gaps. A grid lets you build complex layouts. All modern browsers conform to the grid specification.

**GRID LAYOUT PRINCIPLES**

Before we use the grid layout here are some few important terminologies

* **Grid line**— These make up the structure of the grid. A grid line can be vertical or horizontal and lie on either side of a row or column. The grid-gap, if defined, lies atop the grid lines.
* **Grid track**— A grid track is the space between two adjacent grid lines. A grid has horizontal tracks (rows) and vertical tracks (columns).
* **Grid cell**— A single space on the grid, where a horizontal grid track and a verticalgrid track overlap.
* **Grid area**— A rectangular area on the grid made up by one or more grid cells.The area is between two vertical grid lines and two horizontal grid lines.

Let’s build the same menu again but using CSS grid. It’s important to note the use of grid here does not render flexbox useless. As we go through the page, you’ll see that flexbox is still an important part of the layout. We’ll point out places on the page where it makes sense to use flexbox.

This version of the HTML has placed each section of the page as a grid item: the header, the menu (nav), the main, and the two sidebars. We’ve also added the tile class to the main and the two sidebars, as this class provides the white background color and the padding that these elements have in common.

(code listing html file V2.0)

Now let’s add CSS to make it look like the one we achieved with flexbox.

1. We set the grid container and define its grid tracks using grid-template-columns and grid-template-rows. The columns are defined using the fraction units 2 fr and 1fr, so the first column will grow twice as much as the second. The rows use the repeat() function. This function provides a shorthand for declaring multiplegrid tracks.The declaration, grid-template-rows: repeat(4, auto); defines four horizontal grid tracks of height auto. It’s equivalent to grid-template-rows: auto auto auto auto. The track size of auto will grow as necessary to the size of its content.
2. With the grid tracks defined, the next portion of the code places each grid item at specific location on the grid. The browser assigns numbers to each grid line in a grid, as shown in figure. The CSS uses these numbers to indicate where each itemshould be placed. For e.g. If you want a grid item to span from grid line1 to grid line 3, you’ll apply grid-column: 1 / 3 to the element. Or, you can apply grid-row: 3 / 5 to a grid item to make it span from the horizontal grid line 3 to gridline 5. These two properties together specify the grid area you want for an element.

Now we see that we have achieved most of the thing with grid then why go for flexbox? The answer is that they are complementary. Use flexbox for 1-D positioning and use grid for 2-D positioning. We will use flexbox to complete the menu bar.

When your design calls for an alignment of items in two dimensions, use grid. When you’re only concerned about one-directional flow, use flexbox. In practice, this will often (but not always) mean grid makes the most sense for a high-level layout of thepage, and flexbox makes more sense for certain elements within each grid area.

To summarize the difference between flexbox and grid

* Flexbox is basically one-dimensional, whereas grid is two-dimensional.
* Flexbox works from the content out, whereas grid works from the layout in.

You may also want to check out other properties of flexbox and grid from Mozilla developer docs <https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox>

And also on Rachel Andrew’s website<https://rachelandrew.co.uk/archives/2016/03/30/should-i-use-grid-or-flexbox/>

where other articles about grid and flex box is explained in depth.