Layout with Flexbox

CSS Flexbox

The CSS `display: flex` property sets an HTML element as a block level flex container which takes the full width of its parent container. Any child elements that reside within the flex container are called flex items.
Flex items change their size and location in response to the size and position of their parent container.

```
div {
  display: flex;
}
```

`justify-content` Property

The CSS `justify-content` flexbox property defines how the browser distributes space between and around content items along the main-axis of their container. This is when the content items do not use all available space on the major-axis (horizontally).
`justify-content` can have the values of:

- `flex-start`
- `flex-end`
- `center`
- `space-between`
- `space-around`

```
/* Items based at the center of the parent
container: */
div {
  display: flex;
  justify-content: center;
}

/* Items based at the upper-left side of
the parent container: */
div {
  display: flex;
  justify-content: flex-start;
}
```

`flex` Property

The `flex` CSS property specifies how a flex item will grow or shrink so as to fit within the space available in its `flex` container. This is a shorthand property that declares the following properties in order on a single line:

- `flex-grow`
- `flex-shrink`
- `flex-basis`

```css
/* Three properties declared on three lines: */
.first-flex-item {
  flex-grow: 2;
  flex-shrink: 1;
  flex-basis: 150px;
}

/* Same three properties declared on one line: */
.first-flex-item {
  flex: 2 1 150px;
}
```

`flex-direction` Property

The `flex-direction` CSS property specifies how flex items are placed in the flex container - either vertically or horizontally. This property also determines whether those flex items appear in order or in reverse order.

```css
div {
  display: flex;
  flex-direction: row-reverse;
}
```

`align-content` Property

The `align-content` property modifies the behavior of the flex-wrap property. It determines how to space rows from top to bottom (ie. along the cross axis). Multiple rows of items are needed for this property to take effect.

`flex-grow` Property

The CSS `flex-grow` property allows flex items to grow as the parent container increases in size horizontally. This property accepts numerical values and specifies how an element should grow relative to its sibling elements based on this value.
The default value for this property is `0`.

```
.panelA {
  width: 100px;
  flex-grow: 1;
}

/* This panelB element will stretch twice
wider than the panelA element */
.panelB {
  width: 100px;
  flex-grow: 2;
}
```

`flex-shrink` Property

The CSS `flex-shrink` property determines how an element should shrink as the parent container decreases in size horizontally. This property accepts a numerical value which specifies the ratios for the shrinkage of a flex item compared to its other sibling elements within its parent container.
The default value for this property is `1`.

```
.container {
  display: flex;
}

.item-a {
  flex-shrink: 1;
  /* The value 1 indicates that the item
should shrink. */
}

.item-b {
  flex-shrink: 2;
  /* The value 2 indicates that the item
should shrink twice than the element item-
a. */
}
```

Css flex-basis property

The `flex-basis` CSS property sets the initial base size for a flex item before any other space is distributed according to other flex properties.

```
// Default Syntax
flex-basis: auto;
```

The CSS `flex-flow` property

The CSS property `flex-flow` provides a shorthand for the properties `flex-direction` and `flex-wrap` . The value of the `flex-direction` property specifies the direction of the flex items and the value of the `flex-wrap` property allows flex items to move to the next line instead of shrinking to fit inside the flex container. The `flex-flow` property should be declared on the flex container.

```
// In this example code block, "column" is
the value of the property "flex-direction"
and "wrap" is the value of the property
"flex-wrap".

.container {
  display: flex;
  flex-flow: column wrap;
}
```

CSS `display: inline-flex` property

The CSS `display: inline-flex` property sets an HTML element as an inline flex container which takes only the required space for the content. Any child elements that reside within the flex container are called flex items. Flex items change their size and location in response to the size and position of their parent container.

```
.container{
  display: inline-flex;
}
```

Flexbox Properties align-items

When working with CSS flexbox `align-items` is used to align flex items vertically within a parent container.

Css flex-wrap property

The `flex-wrap` property specifies whether flex items should wrap or not. This applies to flex items only. Once you tell your container to `flex-wrap`, wrapping become a priority over shrinking. Flex items will only begin to wrap if their combined `flex-basis` value is `greater` than the current size of their flex container.

```
.container {
  display: flex;
  flex-wrap: wrap;
  width: 200px;
}
```

Print Share

0

Learn CSS: Grid

Grid Template Columns

To specify the number of columns of the grid and the widths of each column, the CSS property `grid-template-columns` is used on the grid container. The number of width values determines the number of columns and each width value can be either in pixels(`px`) or percentages(%).

```css
#grid-container {
  display: grid;
  width: 100px;
  grid-template-columns: 20px 20% 60%;
}
```

fr Relative Unit

The CSS grid relative sizing unit `fr` is used to split rows and/or columns into proportional distances. Each `fr` unit is a fraction of the grid's overall length and width. If a fixed unit is used along with `fr` (like pixels for example), then the `fr` units will only be proportional to the distance left over.

```css
/*
In this example, the second column take
60px of the avaiable 100px so the first
and third columns split the remaining
available 40px into two parts (`1fr` = 50%
or 20px)
*/

.grid {
  display: grid;
  width: 100px;
  grid-template-columns: 1fr 60px 1fr;
}
```

Grid Gap

The CSS `grid-gap` property is a shorthand way of setting the two properties `grid-row-gap` and `grid-column-gap`. It is used to determine the size of the gap between each row and each column. The first value sets the size of the gap between rows and while the second value sets the size of the gap between columns.

```
// The distance between rows is 20px
// The distance between columns is 10px

#grid-container {
  display: grid;
  grid-gap: 20px 10px;
}
```

CSS Block Level Grid

CSS Grid is a two-dimensional CSS layout system. To set an HTML element into a block-level *grid container* use `display: grid` property/value. The nested elements inside this element are called *grid items*.

```
#grid-container {
  display: grid;
}
```

CSS grid-row

The CSS `grid-row` property is shorthand for the `grid-row-start` and `grid-row-end` properties specifying a grid item's size and location within the grid row. The starting and ending row values are separated by a `/`. There is a corresponding `grid-column` property shorthand that implements the same behavior for columns.

```
/*CSS Syntax */
grid-row: grid-row-start / grid-row-end;

/*Example*/
.item {
  grid-row: 1 / span 2;
}
```

CSS Inline Level Grid

CSS Grid is a two-dimensional CSS layout system. To set an HTML element into a inline-level *grid container* use `display: inline-grid` property/value. The nested elements inside this element are called *grid items*.
The difference between the values `inline-grid` and `grid` is that the `inline-grid` will make the element inline while `grid` will make it a block-level element.

```
#grid-container {
  display: inline-grid;
}
```

`minmax()` Function

The CSS Grid `minmax()` function accepts two parameters:

- The first parameter is the minimum size of a row or column.
- The second parameter is the maximum size.

The grid must have a variable width for the `minmax()` function.
If the maximum value is less than the minimum, then the maximum value is ignored and only the minimum value is used.
The function can be used in the values of the `grid-template-rows` , `grid-template-columns` and `grid-template` properties.

```
/* In this example, the second column will
vary in size between 100px and 500px
depending on the size of the web browser"
*/

.grid {
  display: grid;
  grid-template-columns: 100px
minmax(100px, 500px) 100px;
}
```

grid-row-start & grid-row-end

The CSS `grid-row-start` and `grid-row-end` properties allow single grid items to take up multiple rows. The `grid-row-start` property defines on which row-line the item will start. The `grid-row-end` property defines how many rows an item will span, or on which row-line the item will end. The keyword `span` can be used with either property to automatically calculate the ending value from the starting value or vice versa. There are complementary `grid-column-start` and `grid-column-end` properties that apply the same behavior to columns.

```
/* CSS syntax:
grid-row-start: auto|row-line;
grid-row-end: auto|row-line|span n;
*/
grid-row-start: 2;
grid-row-end: span 2;
```

CSS grid-row-gap

The CSS `grid-row-gap` property determines the amount of blank space between each row in a CSS grid layout or in other words, sets the size of the gap (gutter) between an element's grid rows. The `grid-column-gap` provides the same functionality for space between grid columns.

```
/*CSS Syntax */
grid-row-gap: length; /*Any legal length
value, like px or %. 0 is the default
value*/
```

CSS grid-area

The CSS `grid-area` property specifies a grid item's size and location in a grid layout and is a shorthand property for the `grid-row-start`, `grid-column-start`, `grid-row-end`, and `grid-column-end` in that order. Each value is separated by a `/`.
In the included example, `Item1` will start on row 2 and column 1, and span 2 rows and 3 columns

```
.item1 {
  grid-area: 2 / 1 / span 2 / span 3;
}
```

Justify Items

The `justify-items` property is used on a grid container. It's used to determine how the grid items are spread out along a row by setting the default `justify-self` property for all child boxes.
The value `start` aligns grid items to the left side of the grid area.
The value `end` aligns grid items to the right side of the grid area.
The value `center` aligns grid items to the center of the grid area.
The value `stretch` stretches all items to fill the grid area.

```
#container {
  display: grid;
  justify-items: center;
  grid-template-columns: 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}
```

Align Self

The CSS `align-self` property is used to set how an individual grid item positions itself along the column or block axis. By default grid items inherit the value of the `align-items` property on the container. So if the `align-self` value is set, it would over-ride the inherited `align-items` value.
The value `start` positions grid items on the top of the grid area.
The value `end` aligns the grid on the bottom of the grid area.
The value `center` positions grid items on the center of the grid area.
The value `stretch` positions grid items to fill the grid area (default).

CSS grid-template-areas

The CSS `grid-template-areas` property allows the naming of sections of a webpage to use as values in the `grid-row-start`, `grid-row-end`, `grid-column-start`, `grid-column-end`, and `grid-area` properties. They specify named grid areas within a CSS grid.

```css
/* Specify two rows, where "item" spans
the first two columns in the first two
rows (in a four column grid layout)*/
.item {
  grid-area: nav;
}
.grid-container {
  display: grid;
  grid-template-areas:
    'nav nav . .'
    'nav nav . .';
}
```

CSS grid-auto-flow

The CSS `grid-auto-flow` property specifies whether implicity-added elements should be added as rows or columns within a grid or, in other words, it controls how auto-placed items get inserted in the grid and this property is declared on the grid container.
The value `row` specifies the new elements should fill rows from left to right and create new rows when there are too many elements (default).
The value `column` specifies the new elements should fill columns from top to bottom and create new columns when there are too many elements.
The value `dense` invokes an algorithm that attempts to fill holes earlier in the grid layout if smaller elements are added.

```css
/*CSS Syntax */
grid-auto-flow: row|column|dense|row
dense|column dense;
```

Justify Content

Sometimes the total size of the grid items can be smaller than the grid container. If this is the case, the CSS property `justify-content` can be used to position the entire grid along the row or inline axis of the grid container.
The value `start` aligns the grid to the left side of the grid container.
The value `end` aligns the grid to the right side of the grid container.
The value `center` centers the grid horizontally in the grid container.
The value `stretch` stretches the grid items to increase the size of the grid to expand horizontally across the container.
The value `space-around` includes an equal amount of space on each side of a grid element, resulting in double the amount of space between elements as there is before the first and after the last element.
The value `space-between` includes an equal amount of space between grid items and no space at either end.
The value `space-evenly` places an even amount of space between grid items and at either end.

Align Content

Some times the total size of the grid items can be smaller than the grid container. If this is the case, the CSS property `align-content` can be used to position the entire grid along the column axis of the grid container.
The property is declared on the grid container.
The value `start` aligns the grid to the top of the grid container.
The value `end` aligns the grid to the bottom of the grid container.
The value `center` centers the grid vertically in the grid container.
The value `stretch` stretches the grid items to increase the size of the grid to expand vertically across the container.
The value `space-around` includes an equal amount of space on each side of a grid element, resulting in double the amount of space between elements as there is before the first and after the last element.
The value `space-between` includes an equal amount of space between grid items and no space at either end.
The value `space-evenly` places an even amount of space between grid items and at either end.

CSS grid-auto-rows

The CSS `grid-auto-rows` property specifies the height of implicitly added grid rows or it sets a size for the rows in a grid container. This property is declared on the grid container. `grid-auto-columns` provides the same functionality for columns. Implicitly-added rows or columns occur when there are more grid items than cells available.

Justify Self

The CSS `justify-self` property is used to set how an individual grid item positions itself along the row or inline axis. By default grid items inherit the value of the `justify-items` property on the container. So if the `justify-self` value is set, it would over-ride the inherited `justify-items` value.
The value `start` positions grid items on the left side of the grid area.
The value `end` positions the grid items on the right side of the grid area.
The value `center` positions grid items on the center of the grid area.
The value `stretch` positions grid items to fill the grid area (default).

```
// The grid items are positioned to the
right (end) of the row.

#grid-container {
  display: grid;
  justify-items: start;
}

.grid-items {
  justify-self: end;
}
```

CSS grid-area

The CSS `grid-area` property allows for elements to overlap each other by using the `z-index` property on a particular element which tells the browser to render that element on top of the other elements.

Align Items

The `align-items` property is used on a grid container. It's used to determine how the grid items are spread out along the column by setting the default `align-self` property for all child grid items.
The value `start` aligns grid items to the top side of the grid area.
The value `end` aligns grid items to the bottom side of the grid area.
The value `center` aligns grid items to the center of the grid area.
The value `stretch` stretches all items to fill the grid area.

```
#container {
  display: grid;
  align-items: start;
  grid-template-columns: 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}
```

Print   Share

Learn Responsive Design

Media Type in Media Queries

When writing media queries in CSS we use the `only` keyword to specify the type of device. Initially, CSS incorporated a variety of media types such as `screen`, `print` and `handheld`. In order to ensure responsive design and to accommodate a variety of screen sizes the keyword `screen` is now always used for displaying content.

```
@media only screen and (min-width: 600px)
{
    /* ruleset for >= 600px */
}
```

And Operator Media Queries

Through using the `and` operator when writing media queries in CSS, we can specify that multiple media features are required. For example we could require a certain width as well as a specific resolution for a CSS ruleset to apply. The `and` operator when chaining together multiple media features allows us to be more specific with the screens that we are targeting as we build responsive designs.

```
@media only screen and (max-width: 480px)
and (min-resolution: 300dpi) {
    /* CSS ruleset */
}
```

css media query

A CSS *media query* can be used to adapt a website's display and layout to different screen sizes. A media query begins with the `@media` keyword and is followed by one or more conditions that check screen size, orientation, resolution, and/or other properties. If these conditions are met, all CSS rules within the media query will be applied to the page. Media queries are used for responsive web design by tailoring specific stylesheets to different types of devices such as laptops, tablets, mobiles, and more.

```css
/* For screen sizes less than or equal to
480px (most likely a mobile device), the
body element's font size will be set to
12px and the #photo element's width will
be set to 100% */
@media only screen and (max-width: 480px)
{
  body {
    font-size: 12px;
  }

  #photo {
    width: 100%;
  }
}
```

CSS Media Features in Media Queries

Media queries in web development ensure that a website design is responsive. It does so through creating tailored style sheets based on the resolution of the device that it is displayed upon. The browser will detect the screen size and apply the CSS styles for that screen size based on specified media features. Media queries can set rules based on features like the screen width, the screen resolution and device orientation. These media rules can be separated by commas. When one of the media rules is true then the accompanying CSS will apply.

Ranges in Media Queries

Media queries can use CSS to target screen sizes within a certain range through using multiple widths and/or heights. This is an effective tool for responsive design that will address a variety of screen sizes in one CSS media query. In order to set a range for width or the height, set the minimum screen size through using `min-width` and/or `min-height` and then set the maximum through using `max-width` or `max-height`. These properties are used in combination with the `and` operator.

```
@media only screen and (min-width: 480px)
and (max-width: 600px) {
    /* ruleset for 480px - 600px */
}
```

CSS unit `em`

`em` is a CSS unit used to create relatively-sized content. 1 `em` unit represents the base font size of the current element.
In the example code block, `<span>` elements nested inside of elements with class `nav-container` will have a font size of `15px`.

```
.nav-container {
  font-size: 10px;
}

.nav-container span {
  font-size: 1.5em; /* 10 x 1.5 = 15px */
}
```

Usage of Percentages in CSS

Instead of defining a fixed width using units like `px`, or `cm`, percentages can be used in CSS to set the height and width of child elements relative to the dimensions of their parent. The child elements will grow or shrink according to the set percentage values if the parent element changes in size.

```
.news-row {
  height: 300px;
  width: 500px;
}

.news-row .news-column {
  height: 80%; /* 240px */
  width: 50%; /* 250px */
}
```

background-size: cover;

The CSS `background-size` property is used to specify the size of the background image. When given the value `cover`, like `background-size:cover`, the image covers the complete background of the container in which it is being displayed.
The proportions of the image are maintained, so if the dimensions exceed the container's, then some parts of the image will be left out.

CSS unit `rem`

The CSS unit `rem` can be used to set the font size of HTML elements relative to the font size of the root element. 1 `rem` represents the size of the base font within the root element - the `<html>` tag.
In the example code block, the font size for all the `<span>` elements will be twice the size of the font size declared for the root element.

```
html {
  font-size: 18px;
}

span {
  font-size: 2rem;
}
```

Print   Share

0