CSS flex property

The flex property in CSS is shorthand for **flex-grow, flex-shrink,** and flex-basis. It only works on the flex-items, so if the container's item is not a flex-item, the **flex** property will not affect the corresponding item.

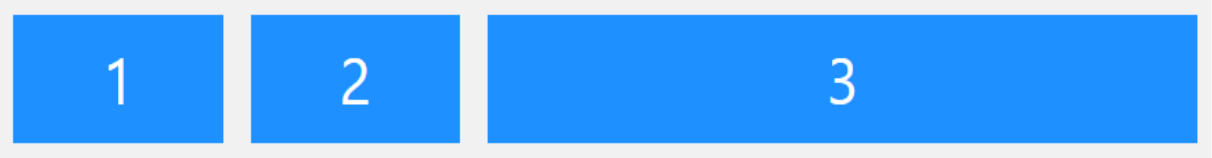
This property is used to set the length of flexible items. The positioning of child elements and the main container is easy with this CSS property. It is used to set how a flex-item will shrink or grow to fit in the space.

The **flex** property can be specified by one, two, or three values.

- When there is the one-value syntax, the value must either be a number or the keywords such as **none, auto,** or **initial**.
- When there is the two-value syntax, the first value must be a number (used as **flex-grow**), the second value can either be a number (used for **flex-shrink**) or a valid width value (used as **flex-basis**).
- When there is three-value syntax, then the values must follow the order: a **number** for the **flex-grow,** a **number** for the **flex-shrink,** and a valid **width** value for **flex-basis**.

The flex-grow Property

The `flex-grow` property specifies how much a flex item will grow relative to the rest of the flex items.



Example

Make the third flex item grow eight times faster than the other flex items:

```
<div class="flex-container">
  <div style="flex-grow: 1">1</div>
  <div style="flex-grow: 1">2</div>
  <div style="flex-grow: 8">3</div>
</div>
```

flex-direction Property

The `flex-direction` property specifies the direction of the flexible items.

Note: If the element is not a flexible item, the `flex-direction` property has no effect.

Example

Set the direction of the flexible items inside the element in reverse order:

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

The flex-shrink Property

The `flex-shrink` property specifies how much a flex item will shrink relative to the rest of the flex items.

The value must be a number, default value is 1.

Example

Do not let the third flex item shrink as much as the other flex items:

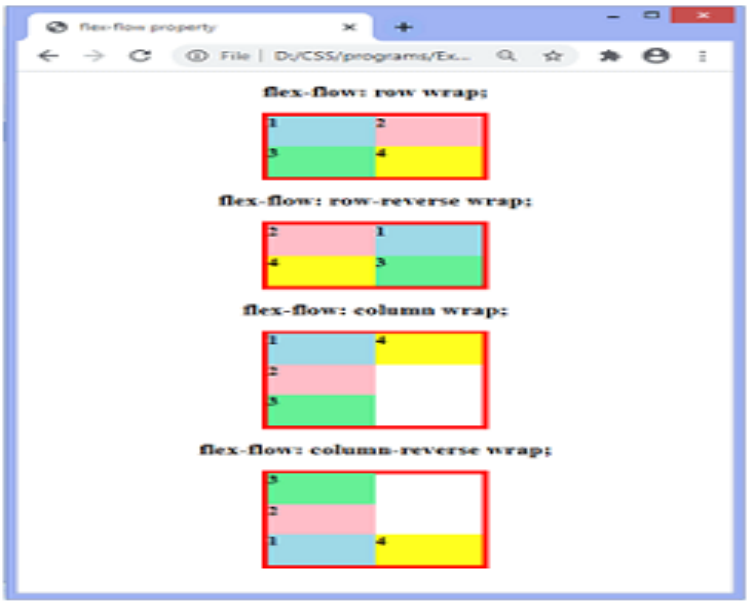
```
<div class="flex-container">
  <div>1</div>
  <div>2</div>
  <div style="flex-shrink: 0">3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
  <div>8</div>
  <div>9</div>
  <div>10</div>
</div>
```

flex-flow property

This CSS property is shorthand for **flex-direction** and **flex-wrap** properties. It only works on the flex-items, so if the container's item is not a flex-item, the **flex-flow** property will not affect the corresponding item.

Syntax

1. flex-flow: flex-direction flex-wrap | initial | inherit; The default value of the flex-flow property is **row nowrap** which is the concatenation of the default values of **flex-direction** (i.e. **row**) and **flex-wrap** (i.e. **nowrap**) properties.

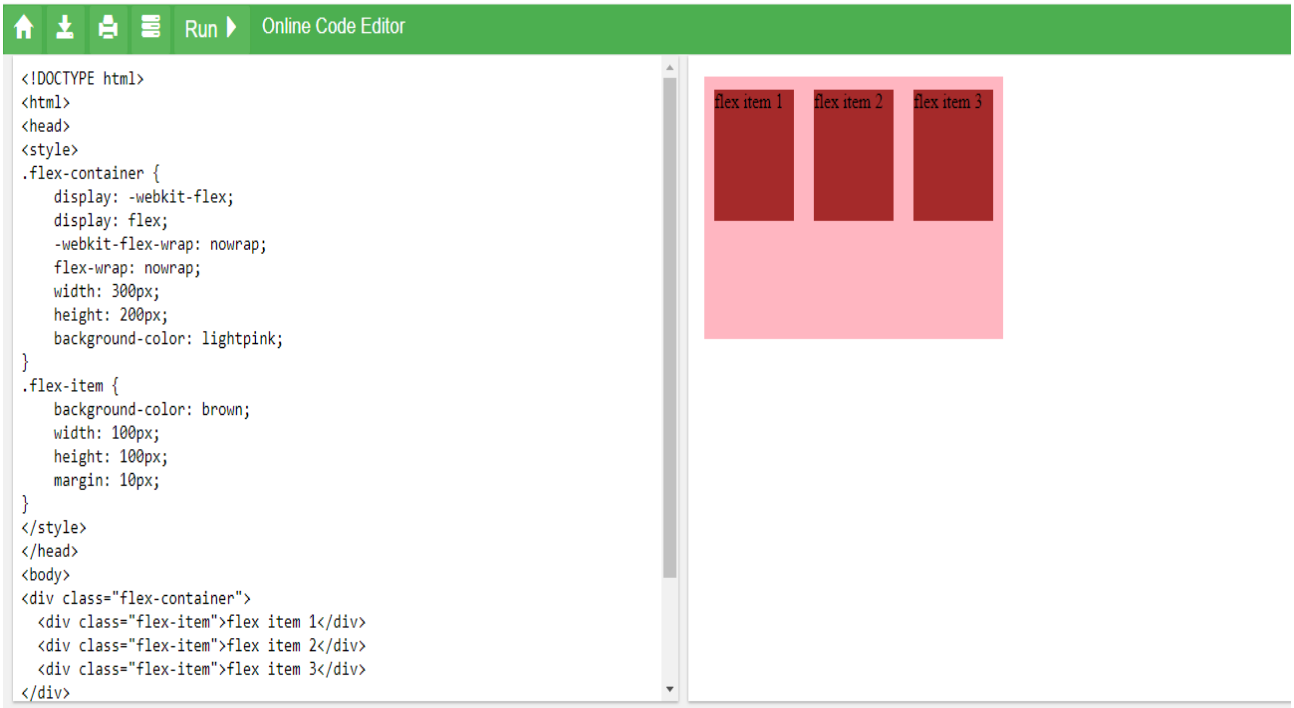


flex-wrap

The CSS3 Flexbox flex-wrap property specifies if the flex-items should wrap or not, in the case of not enough space for them on one flex line.

Its possible values are:

- **nowrap:** It is the default value. The flexible items will not wrap.
- **wrap:** It specifies that the flexible items will wrap if necessary.
- **wrap-reverse:** It specifies that the flexible items will wrap, if necessary, in reverse order.



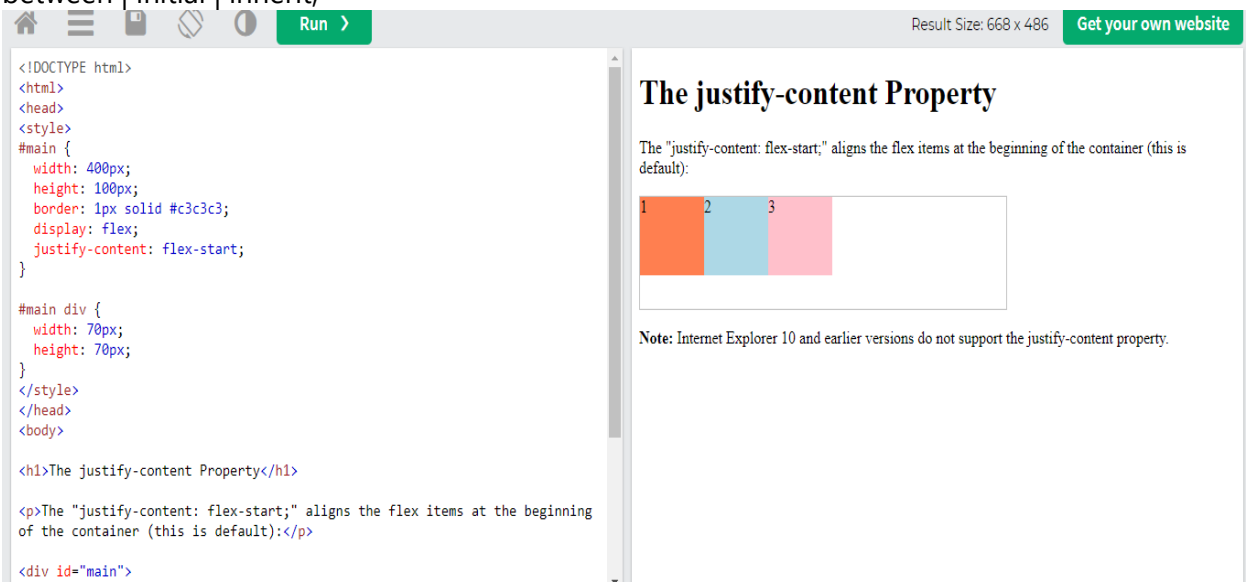
justify-content

This CSS property is used to align the items of the flexible box container when the items do not use all available space on the main-axis (horizontally). It defines how the browser distributes the space around and between the content items.

This CSS property can't be used to describe containers or items along the vertical axis. To align the items vertically, we have to use the **align-items** property.

Syntax

1. justify-content: center | flex-start | flex-end | space-around | space-evenly | space-between | initial | inherit;



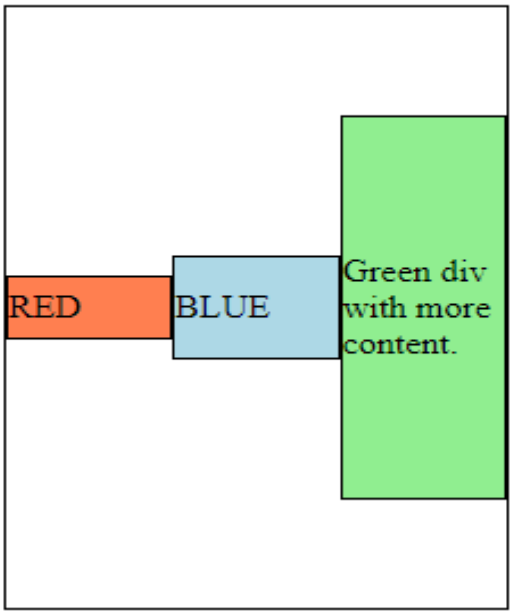
align-items Property

Example

Center the alignments for all the items of the flexible element:

```
div {
  display: flex;
  align-items: center;
}
```

align-items: center



2.)Differences Between Flex and Grid

Dimensionality and Flexibility

Flexbox offers greater control over alignment and space distribution between items. Being one-dimensional, Flexbox only deals with either columns or rows. This system works for smaller layouts, but cannot render complex displays such as text or document-centric properties that enable floats and columns.

Grid has two-dimension layout capabilities which allow flexible widths as a unit of length. This compensates for the limitations in Flex.

Alignment

Flexbox allows fine-tuning of alignments to ensure exact specification sharing. Flex Direction allows developers to align elements vertically or horizontally, which is used when developers create and reverse rows or columns.

For broader alignments in both dimensions simultaneously, CSS Grid deploys fractional measure units for grid fluidity and auto-keyword functionality to automatically adjust columns or rows. The in-built automation saves developers from re-work regimes that may potentially originate in confused calculations.

Item Management

Flex Container is the parent element while Flex Item represents the children. The Flex Container can ensure balanced representation by adjusting item dimensions. This allows developers to design for fluctuating screen sizes.

For fine-tuning this aesthetic, Grid supports both implicit and explicit content placement. Its inbuilt automation allows it to automatically extend line items and copy values into the new creation from the preceding item.

Conclusion

Flexbox and CSS Grid both allow a powerful measure of control over their respective domains of front-end development. However, their capabilities are exponentiated when they are combined, utilizing their respective strengths to create an extremely fluid, customizable, beautiful, smooth, and simple experience.

Combining their code also results in a more lightweight setup where abstraction in both domains spills over into the other. There are vast applications to both options, and even more when they are combined into a powerful setup.

3). The position Property

The `position` property specifies the type of positioning method used for an element.

There are five different position values:

- `static`
- `relative`
- `fixed`
- `absolute`
- `sticky`
- `initial`
- `inherit`

Elements are then positioned using the top, bottom, left, and right properties. However, these properties will not work unless the `position` property is set first. They also work differently depending on the position value.

position: static;

HTML elements are positioned static by default.

Static positioned elements are not affected by the top, bottom, left, and right properties.

An element with `position: static;` is not positioned in any special way; it is always positioned according to the normal flow of the page:

This element has position: static;

Here is the CSS that is used:

Example

```
div.static {
  position: static;
  border: 3px solid #73AD21;
}
```

Try it Yourself »

position: relative;

An element with `position: relative;` is positioned relative to its normal position.

Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

This element has position: relative;

Here is the CSS that is used:

Example

```
div.relative {
  position: relative;
  left: 30px;
  border: 3px solid #73AD21;
}
```

Try it Yourself »

position: fixed;

An element with `position: fixed;` is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

A fixed element does not leave a gap in the page where it would normally have been located.

Notice the fixed element in the lower-right corner of the page. Here is the CSS that is used:

Example

```
div.fixed {
  position: fixed;
```

```
  bottom: 0;
  right: 0;
  width: 300px;
  border: 3px solid #73AD21;
}
```

Try it Yourself »

This element has `position: fixed;`

position: absolute;

An element with `position: absolute;` is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).

However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

Note: Absolute positioned elements are removed from the normal flow, and can overlap elements.

Here is a simple example:

This element has position: relative;

This element has position: absolute;

Here is the CSS that is used:

Example

```
div.relative {
  position: relative;
  width: 400px;
  height: 200px;
  border: 3px solid #73AD21;
}

div.absolute {
  position: absolute;
  top: 80px;
  right: 0;
  width: 200px;
  height: 100px;
```

```
  border: 3px solid #73AD21;
}
```

Try it Yourself »

position: sticky;

An element with `position: sticky;` is positioned based on the user's scroll position.

A sticky element toggles between `relative` and `fixed`, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

Note: Internet Explorer does not support sticky positioning. Safari requires a -webkit- prefix (see example below). You must also specify at least one of `top`, `right`, `bottom` or `left` for sticky positioning to work.

In this example, the sticky element sticks to the top of the page (`top: 0`), when you reach its scroll position.

Example

```
div.sticky {
  position: -webkit-sticky; /* Safari */
  position: sticky;
  top: 0;
  background-color: green;
  border: 2px solid #4CAF50;
}
```

initial Keyword

Example

Set the text color of the element to red, but keep the initial color for <h1> elements:

```css
div {
  color: red;
}

h1 {
  color: initial;
}
```

Inherit Keyword

The `inherit` keyword specifies that a property should inherit its value from its parent element.

The `inherit` keyword can be used for any CSS property, and on any HTML element.

Example

Set the text-color for elements to blue, except those inside elements with class="extra":

```css
span {
  color: blue;
}

.extra span {
  color: inherit;
}
```