**EXPOSÉ FLEX BOX**

**SLIDE 1 : Debut**

**SLIDE 2 : Historically**

Date : 23 July 2009

Native name : CSS Flexible Box Layout

Abbreviation : Flexbox

CSS Flexible Box Layout, commonly known as Flexbox

In the 2000s the intensive use of the Web by [mobile agents](https://en.wikipedia.org/wiki/Mobile_agent) motivated "liquid layouts" and [responsive elements](https://en.wikipedia.org/wiki/Responsive_web_design#History) for the growing variety of [screen sizes](https://en.wikipedia.org/wiki/Display_size" \o "Display size).[[5]](https://en.wikipedia.org/wiki/CSS_Flexible_Box_Layout#cite_note-5) In the 2010s, the intensive use of popular [JavaScript](https://en.wikipedia.org/wiki/JavaScript) layout [frameworks](https://en.wikipedia.org/wiki/Software_framework" \o "Software framework), such as [Bootstrap](https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)" \o "Bootstrap (front-end framework)), inspired CSS flex-box and grid layout specifications

As of September 2020, 99.69% of installed browsers (99.29% of desktop browsers and 100% of mobile browsers) support CSS Flexible Box Layout.

**SLIDE 3 : Introduction**

The Flexbox Layout (Flexible Box aims at providing a more efficient way to lay out, align and distribute space among items in a container, even when their size is unknown and/or dynamic.

The main idea behind the flex layout is to give the container the ability to alter its items’ width, height and order to best fill the available space (mostly to accommodate to all kind of display devices and screen sizes). A flex container expands items to fill available free space or shrinks them to prevent overflow.

Most importantly, the flexbox layout is direction-agnostic as opposed to the regular layouts (block which is vertically-based and inline which is horizontally-based). While those work well for pages, they lack flexibility to support large or complex applications (especially when it comes to orientation changing, resizing, stretching, shrinking, etc.).

Flexbox layout is most appropriate to the components of an application, and small-scale layouts, while the [Grid](https://css-tricks.com/snippets/css/complete-guide-grid/) layout is intended for larger scale layouts.

**SLIDE 4 : Basic**

Since flexbox is a whole module and not a single property, it involves a lot of things including its whole set of properties. Some of them are meant to be set on the container (parent element, known as “flex container”) whereas the others are meant to be set on the children (said “flex items”).

If “regular” layout is based on both block and inline flow directions, the flex layout is based on “flex-flow directions”.

Items will be laid out following either the main axis (from main-start to main-end) or the cross axis (from cross-start to cross-end).

Main axis – The main axis of a flex container is the primary axis along which flex items are laid out. Beware, it is not necessarily horizontal; it depends on the flex-direction property

**SLIDE 5 : Flex properties**

A list of differents properties :

* For the parents (flex container)
* For the children (flex items)

**SLIDE 6 : display**

To turn an HTML element into a Flexbox, you need to use the display property with the values "flex" (flexible container area at block level) or "inline-flex" (defining the flexible container area at embedded level). Otherwise, the browser ignores all Flexbox properties you have used.

This defines a flex container; inline or block depending on the given value. It enables a flex context for all its direct children.

**SLIDE 7 : align-items**

* stretch (default): stretch to fill the container (still respect min-width/max-width)
* flex-start / start / self-start: items are placed at the start of the cross axis. The difference between these is subtle, and is about respecting the flex-direction rules or the writing-mode rules.
* flex-end / end / self-end: items are placed at the end of the cross axis. The difference again is subtle and is about respecting flex-direction rules vs. writing-mode rules.
* center: items are centered in the cross-axis
* baseline: items are aligned such as their baselines align

**SLIDE 8 : Flex-direction**

This establishes the main-axis, thus defining the direction flex items are placed in the flex container. Flexbox is a single-direction layout concept. Think of flex items as primarily laying out either in horizontal rows or vertical columns.

* row (default): left to right
* row-reverse: right to left
* column top to bottom
* column-reverse: bottom to top

**SLIDE 9 : justify-content**

This defines the alignment along the main axis. It helps distribute extra free space leftover when either all the flex items on a line are inflexible, or are flexible but have reached their maximum size. It also exerts some control over the alignment of items when they overflow the line.

* flex-start (default): items are packed toward the start of the flex-direction.
* flex-end: items are packed toward the end of the flex-direction.
* center: items are centered along the line
* space-between: items are evenly distributed in the line; first item is on the start line, last item on the end line
* space-around: items are evenly distributed in the line with equal space around them. Note that visually the spaces aren’t equal, since all the items have equal space on both sides. The first item will have one unit of space against the container edge, but two units of space between the next item because that next item has its own spacing that applies.
* space-evenly: items are distributed so that the spacing between any two items (and the space to the edges) is equal.

**SLIDE 10 : order**

By default, the elements are arranged in the order in which they appear in the source code (starting from the lowest numbered ordinal group and working up). To change the order of the objects in the flex container, the order property is used.

The order property defines the order of flex elements by assigning them to ordinal groups. It takes a single integer value, which defines the ordinal group to which the flex element belongs. The default value is 0.

**SLIDE 11 : Flex-grow**

**FLEX GROW**

The flex-grow property defines the ability of an element to grow if necessary. It defines the flex-growth factor (relative to the rest of the objects in the container), which specifies how much space the element should occupy in the container when distributing positive space.

The property takes a unitless value that serves as a proportion.

The value cannot be a negative number.

If all elements in the container have the same value for flex-grow (for example, 1), all elements have the same size in the container.

If the size of one of the flex elements is different (e.g., 2), that element will occupy twice as much space as the size of other elements (whose size is set to 1).

**FLEX SHRINK**

The flex-shrink property is set to allow a flex element to shrink. It specifies how much space the element should take up in the container when negative space is distributed.

The property takes a unitless value.

The value cannot be a negative number.

By default, all flex elements can be shrunk, but if we set the value to 0 (do not shrink), they will retain the original size.

**SLIDE 11 : Let’s Pratice**

Tp flexblox

**SLIDE ++ : En plus**

#### flex-flow

This is a shorthand for the flex-direction and flex-wrap properties, which together define the flex container’s main and cross axes. The default value is row nowrap.