CHAPTER 14
FLEXBOX

- This is a 1D layout method for arranging items in rows or columns.

- Items either *flex* to fill additional space or *shrink* to occupy smaller space.

- When the **display** property is set to **flex**, then the container to which it is applied becomes a **flex container**. The container is then displayed as a **block-level element**. This is the basis for how it interacts with the rest of the page. When an element is converted to a **flex container**, then all its children are **flex items** and are laid out as such.

- The **Flex Model** is described over here. **Flex elements** are laid out across 2 axes. The first of these axes is the **Main Axis**. This is the axis that runs in the direction in which the **flex items** are laid out in. The start of this axis is called the **main start** and the end of this axis is called the **main end**. The length from the start edge of a flex item to its end edge is called the **main size**.

- When the **Flex Model** is applied to a container, then the elements in the container expand to the minimum dimensions set by the container.

- The **cross axis** is the axis running perpendicular to the direction the flex items are laid out in. The start and end of this axis are called the **cross start** and **cross end**. The length from the cross-start edge to the cross-end edge is the **cross size**.

- The **flex-direction** property determines the direction in which the axis runs. The default value is **row** as the flex items are laid out across a row. Another value is **column**, this results in the flex items being laid out across a column.

- The **justify-content** property defines how the browser distributes space between and around content items along the main axis of a **flex container**. It has several values. The values that we will focus on over here are: **start**, **end**, **flex-start**, **flex-end** and **center.**

- In the **start** setting, the items are packed flush to each other toward the start edge of the alignment container in the main axis.

- In the **end** setting, the items are packed flush to each other toward the end edge of the alignment container in the main axis.

- In the **flex-start** setting, the items are packed flush to each other toward the edge of the alignment container depending on the flex container's main-start side. This only applies to flex layout items. For items that are not children of a flex container, this value is treated like **start**. *In the example in the video, the 6 divs shifted to the left end of the container, leaving some space at the right end of the container.*

- In the **flex-end** setting, the items are packed flush to each other toward the edge of the alignment container depending on the flex container's main-end side. This only applies to flex layout items. For items that are not children of a flex container, this value is treated like **end**. *In the example in the video, the 6 divs shifted to the right end of the container, leaving some space at the left end of the container.*

- In the **center** setting, the items are packed flush to each other toward the center of the alignment container along the main axis. *In the example in the video, the 6 divs shifted to the center of the container, leaving some space at the left end and the right end of the container.*

- In the **space-around** setting, the items are evenly distributed within the alignment container along the main axis. The spacing between each pair of adjacent items is the same. The empty space before the first and after the last item equals half of the space between each pair of adjacent items. *In the example in the video, we saw that there was an equal amount of space between each of the 6 divs with some space being there between the beginning of the container and the first div and the end of the container and the last div*.

- In the **space-between** setting, the items are evenly distributed within the alignment container along the main axis. The spacing between each pair of adjacent items is the same. The first item is flush with the main-start edge, and the last item is flush with the main-end edge. *In the example in the video, we saw that there was an equal amount of space between each of the 6 divs with no space there between the beginning of the container and the first div and the end of the container and the last div*.

- In the **space-evenly** setting, the items are evenly distributed within the alignment container along the main axis. The spacing between each pair of adjacent items, the main-start edge and the first item, and the main-end edge and the last item, are all exactly the same. *In the example in the video there is an equal amount of space between each of the 6 divs and also between the beginning of the container and the first div and the end of the container and the last div*.

- The **gap** property sets an equal amount of gap between each flex item in the flex container. It can be set in **rem**, **px**, etc.

- The **align-items** property deals with the **cross-axis**, i.e., the axis that runs perpendicular to the direction in which the flex items are laid out in. We will focus on the **flex-start**, **flex-end** and **center** settings here. **In short, this property aligns the columns.**

- In the **flex-start** setting, the items are aligned flush against the **main-start** or **cross-start** side when used in **flex layout**. If used without the flex layout, it behaves as **start**. *In the example in the video, all the flex items are aligned at the top (main-start / cross-start) side.*

- In the **flex-end** setting, the items are aligned flush against the **main-end** or **cross-end** side when used in **flex layout**. If used without the flex layout, it behaves as **end**. *In the example in the video, all the flex items are aligned at the bottom (main-end / cross-end) side.*

- In the **center** setting, the flex items' margin boxes are centered within the line on the cross-axis. *In the example in the video, all the flex items are aligned at the center of the flex container.*

- The **flex-direction** property determines how flex items are placed in the container defining the main-axis and the direction. We will focus on 4 of its values: **row**, **row-reverse**, **column** and **column-reverse**.

- In the **row** setting, the flex container's main-axis is defined to be the same as the text direction. The **main-start** and the **main-end** points are the same as the content direction. *In the example shown in the video, the flex items are arranged from left to right.*

- In the **column** setting**,** the flex container's main-axis is the same as the block-axis. The **main-start** and **main-end** points are the same as the **before** and **after** points of the writing-mode. *In the example shown in the video, the flex items are arranged from top to bottom.*

- The **row-reverse** setting behaves the same as the **row** setting. However, the **main-start** and **main-end** points are opposite to the content direction.

- The **column-reverse** setting behaves the same as the **column** setting. However, the **main-start** and **main-end** are opposite to the content direction.

- By default, if we were to shrink the size of the browser window, then even though the flex container will shrink with it, the **flex items** in it will overflow through its edges. In order to prevent this from happening, we can set the **flex-wrap** property to the **wrap** setting. When the flex items wrap in the flex container, then rows become shorted and split into multiple rows. This will continue as the browser window shrinks and will end when the flex items are all arranged in a single column.

- The values of the **flex-direction** and the **flex-wrap** properties can be specified together in this order in the **flex-flow** property.

- The **align-content** property sets the distribution of space in & around content items along a flexbox's **cross-axis**. **In short, this property aligns the rows in a flex container.** This property has the same values as the **align-items** property.

- In order to be able to center some content using the **flexbox Model**, one must apply the following rules {**display: flex; align-items: center; align-content: center;**}.

- An element can be a **flex item** and also a **flex container** simultaneously.

- The **flex-basis** property sets the initial main size of a flex item. It works unless the initial main size of the said flex item has already been set using the **box-sizing** property. *In the example in the video, the flex-basis value is described as the main size of the flex items.*

- A simplified description of the **flex-grow** property is to describe it as determining the extent to which a flex item can grow. *In the example in the video, when* **flex-grow** *is set to* **1** *then both the flex items grow to the same size in the flex container.* The more correct explanation is to say that the **flex-grow** property sets the amount of **positive free space** that can be assigned to the said flex item's main size. *In the example in the video, when the flex-grow for the second flex item is set to 2, then the second flex item becomes bigger than the first flex item since it gets to use twice as much of the remaining free space in the flex container to grow than what the first flex item can use.* So, the **positive free space** is the amount of remaining free space inside a flex container that a flex item can use to grow in size.

- The **flex-shrink** property can be simply described as the extent to which a flex item can grow smaller when it needs to wrap inside the flex container (like for eg - When we shrink the browser window.). A flex item with a higher **flex-shrink** value can grow smaller.

- The **flex** property can be used to specify the **flex-basis**, **flex-grow** and **flex-shrink** in that order.

- The **order** property is used to determine the **order of flex items** in a flex container. It takes both positive and negative integer values. The **order** property divides the flex items into **ordinal groups**. This means that each flex item has a number assigned to it. The flex items are placed in the ascending order of the ordinal value. If any two flex items **in a group have the same value**, then those items are laid out according to **source order**. When the **flex-direction** is switched to **row-reverse**, then the ordering is reversed. A useful trick is to use {**order: -1;**} on a flex item that we want to place first while keeping the order of all the other flex items unchanged. The default value of order is **0**. **The order property has no effect on elements that aren't flex items.**

- A website that allows us to learn the FLexbox Model by completing challenges is: **https://www.flexboxfroggy.com/**