**Flex Box:**

- Flex box stands for flexible box
- liberty to align and Justify our Elements with just some line of code but you need to think graphically.
- It's a layout model in CSS that makes arranging elements a whole lot easier.
- With flexbox, you have a container that holds your elements, and it's like your shelf. Inside this container, you can use the power of flexbox to control how your elements behave.

Flexbox is just a concept of CSS that makes your life very easy.

Now, lets create some boxes again in HTML to start with flexbox.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Flexbox</title>
    <style>
        .container {
            background-color: tomato;
            border: 2px solid black;
            height: 500px;
            width: auto;
        }

        .box {
            height: 100px;
            width: 100px;
            background-color: dodgerblue;
            margin: 10px;
        }

    </style>
</head>

<body>
    <div class="container">
        <div class="box">1</div>
        <div class="box">2</div>
        <div class="box">3</div>
        <div class="box">4</div>
    </div>
</body>

</html>
```

So this is the basic boxes that we made similar to above.

Now lets use the **display: flex;** property in the parent container

```
.container {
    background-color: tomato;
    border: 2px solid black;
    height: 500px;
    width: auto;
    display: flex;
}
```

When you apply the CSS property display: flex; to an element, you're essentially creating a flex container. This container establishes a flex formatting context for its child elements, or "flex items." This means that the child elements within this container will follow the rules and behaviors defined by the flexbox layout model.

Default property of flex box is to arrange all the elements in the row as you can also see here.

You can also use many flex-direction property to arrange the elements in different ways.

You can also use this important property called **justify-content**.

**justify-content:** Defines how flex items are distributed along the main axis (horizontal for row layout, vertical for column layout).

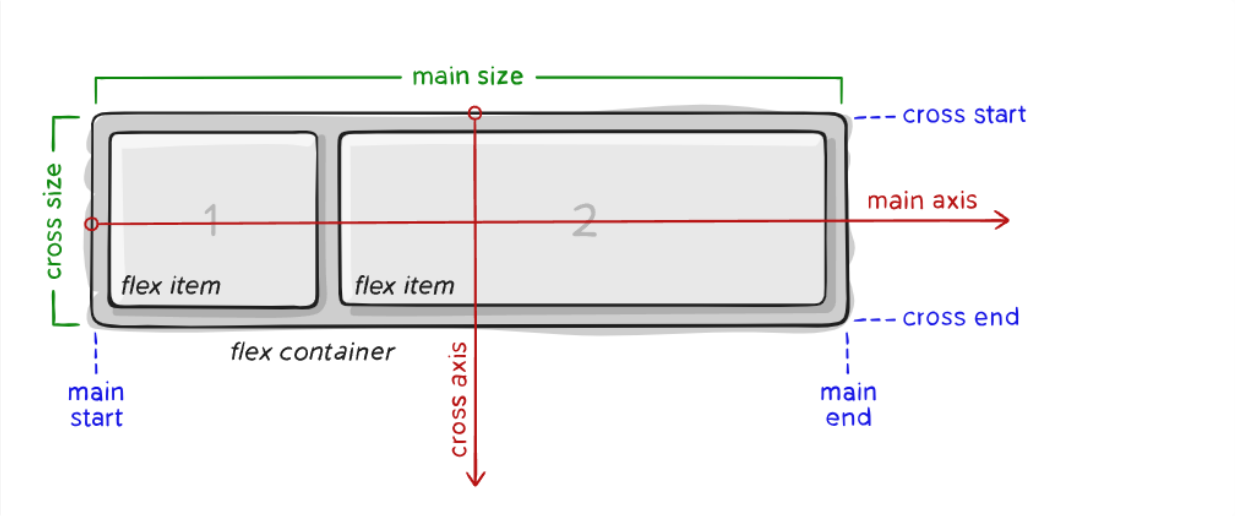
So if we want to center all the row items, we can use **justify-content: center**

```
.container {
    background-color: tomato;
    border: 2px solid black;
    height: 500px;
    width: auto;
    display: flex;
    justify-content: center;
}
```

The justify-content: center; property applied to a flex container in CSS aligns its child elements **horizontally** at the center of the container along the main axis.

But there is 1 more property that is used to align items vertically that is **align-items**.

Explain the axis using the below diagram. diagram refence: https://css-tricks.com/snippets/css/a-guide-to-flexbox/




**[Ask the learners]** If I say, I want my flex direction is row, what will be my main axis?

- X axis

**[Ask the learners]** And If I say, I want my flex direction is column, what will be my main axis?

- Y axis

Now lets take more examples of **justify-content** property.

- **justify-content: flex-start;**
  - Flex items are aligned at the beginning of the container (left for a row layout, top for a column layout).
- **justify-content: flex-end;**
  - Flex items are aligned at the end of the container (right for a row layout, bottom for a column layout).
- **justify-content: center;**
  - Flex items are centered along the container's main axis.
  - Equal space is added before the first item and after the last item, creating a balanced appearance.
- **justify-content: space-between;**
  - Flex items are evenly spaced along the main axis.
  - The first item aligns with the container's start, the last item aligns with the container's end, and equal space is added between the items.
- **justify-content: space-around;**

    - Flex items are evenly spaced along the main axis, with space distributed around them.
    - Space is added before the first item, after the last item, and between each pair of items.
- **justify-content: space-evenly;**
    - Flex items are evenly spaced along the main axis, with equal space added between them.
    - Equal space is added before the first item, between all items, and after the last item.

There are many options for this property, you can choose to play around with all of them.

Now lets understand **align-items** property.

The **align-items property** lets you control how flex items are positioned on the cross axis within the container.

- **align-items: flex-start;**
    - Flex items align at the start of the cross axis (top for row layout, left for column layout).
    - No additional space is added between items and the container's cross axis edge.
- **align-items: flex-end;**
    - Flex items align at the end of the cross axis (bottom for row layout, right for column layout).
    - No additional space is added between items and the container's cross axis edge.
- **align-items: center;**
    - Flex items are vertically centered along the cross axis.
    - Equal space is added above and below the items, creating a balanced appearance.
- **align-items: baseline;**
    - Flex items are aligned along their text baselines.
    - This value can be useful when items have varying font sizes or text content.
- **align-items: stretch;**
    - Flex items are stretched to fill the container's cross axis.
    - If no height is explicitly set on the items, they will take up the full height of the container.

**[Ask the learners]** If flex-direction if set to row and justify-content is center, which direction it will align?

- Horizontally

Now, we have covered two important properties, justify-content and align-items. Now we will move formward with some interesting properties.

- A responsive website is a website that can work of different screen size and can adapt to those different screen sizes. It responds and adjusts its layout, images, and content to fit the screen it's being viewed on, whether that's a desktop computer, tablet, or smartphone.

Lets take an example to scaler's website

Navigate to https://www.scaler.com/

If you minimize the screen, you can see that everything is changing according to the screen size. Design is not breaking and is adapting to the screen size whether it is mobile view or the desktop view.

In inspect tab of our HTML website, we can check how our website will look in different dimensions and screen sizes.

If we decrease the screen size, we can see that the boxes are shrinking. That means it is not responsive, There are various properties in flex box that we can use to make our website responsive.

In the current scenario the boxes are getting shrinked but if I don't want it to shrink and I want them to adjust according to the screen size, for that we can use some CSS properties.

There is 1 property know as **flex-wrap**

**flex-wrap:** The flex-wrap property in flexbox controls whether flex items should wrap onto multiple lines within the flex container when there's not enough space to fit them all on a single line.

So if we use the above code with flex-wrap property, we can see that the boxes are now responsive and not shrinking.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Flexbox</title>
    <style>
        .container {
            background-color: tomato;
            border: 2px solid black;
            height: 500px;
```

```
            display: flex;
            flex-direction: row;
            justify-content: center;
            align-items: center;
            flex-wrap: wrap;

        }

        .box {
            height: 200px;
            width: 200px;
            background-color: dodgerblue;
            margin: 10px;
        }

    </style>
</head>

<body>
    <div class="container">
        <div class="box">1</div>
        <div class="box">2</div>
        <div class="box">3</div>
        <div class="box">4</div>
    </div>
</body>

</html>
```

The flex-wrap property is especially useful when dealing with responsive layouts. It allows you to control how flex items reorganize when the available space changes. For example, if you have a row of cards and they don't fit on a smaller screen, setting flex-wrap: wrap; would cause them to stack vertically, creating a more readable layout.

Here's what each value of flex-wrap does:

- flex-wrap: nowrap;
  - This is the default value.
  - Flex items will stay on a single line, even if they cause overflow. They won't wrap to the next line.
- flex-wrap: wrap;
  - When there's not enough space for all items on a single line, flex items will wrap to the next line. They will stack vertically if needed.
- flex-wrap: wrap-reverse;
  - Similar to wrap, but the wrapping happens in reverse order. The last flex item becomes the first item on the new line, and so on.

Now we can discuss about item wise flex box,For now we were using flex box in container but now we will use flex box within the container items

Navigate to https://css-tricks.com/snippets/css/a-guide-to-flexbox/#aa-properties-for-the-childrenflex-items and explain that we can use flex properties for children items.

There is 1 property called **order** that is used for ordering the items.

The **order** property in flexbox allows you to control the visual order in which flex items appear within a flex container, regardless of their order in the HTML markup. It's particularly useful for reordering items for different screen sizes or creating unique visual layouts. H

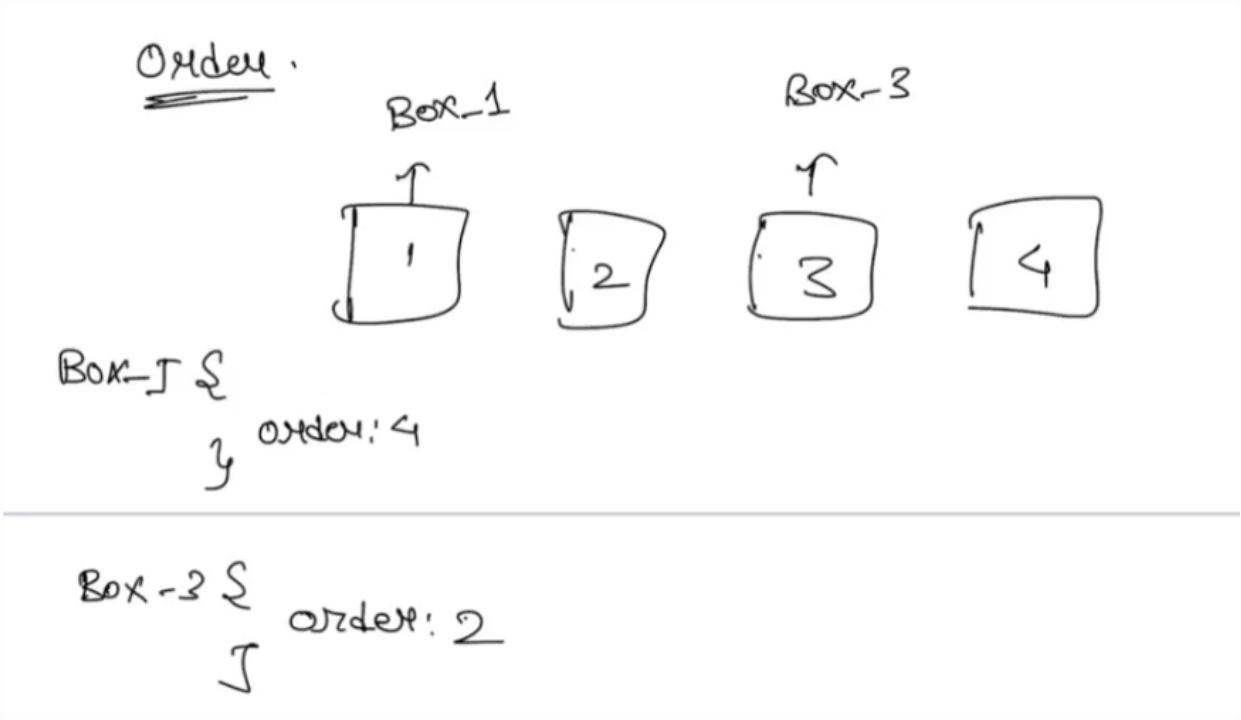
lets give the order property to box1, and see what happens:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Flexbox</title>
    <style>
        .container {
            background-color: tomato;
            border: 2px solid black;
            height: 500px;
            width: auto;
            display: flex;
            flex-direction: row;
            justify-content: center;
            align-items: center;
            flex-wrap: wrap;

        }

        .box {
            height: 200px;
            width: 200px;
            background-color: dodgerblue;
            margin: 10px;
        }

        .box1{
            order: 5;
        }

    </style>
</head>

<body>
    <div class="container">
        <div class="box box1">1</div>
        <div class="box">2</div>
        <div class="box">3</div>
        <div class="box">4</div>
    </div>
</body>
```

```
</html>
```

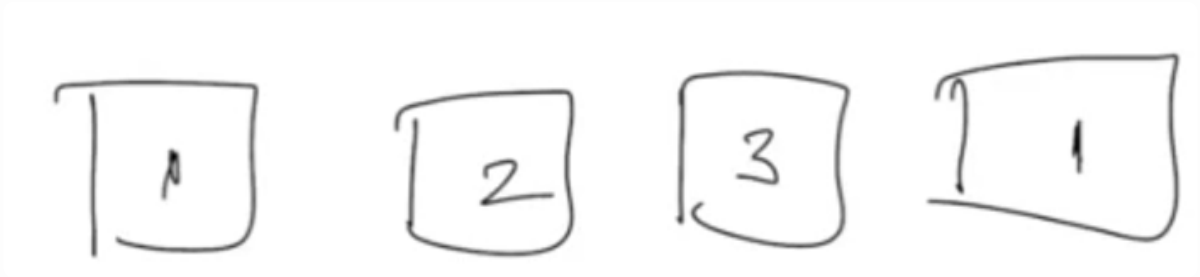
Now you can see that the box1 is ordered at the last.

let's understand this with an example:




Suppose, we gave order: 4 to box1 and order: 2 to box3.

- box1 has the order 4, so it will go at last.
- box3 has order 2, so it will go at third.
- box 2 will be at 2nd position
- box1 will be at first.




Now lets' understand flex-shrink property.

The **flex-shrink** property in flexbox determines how flex items shrink when the container's available space is limited. It defines the ability of an item to shrink in relation to other items in the container when the container's size is reduced.

let's understand this with an example:

We are providing flex-shrink property as 2 to box3.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Flexbox</title>
    <style>
        .container {
            background-color: tomato;
            border: 2px solid black;
            height: 500px;
            display: flex;
            flex-direction: row;
            justify-content: center;
            align-items: center;

        }

        .box {
            height: 200px;
            width: 200px;
            background-color: dodgerblue;
            margin: 10px;
        }

        .box3{
            flex-shrink: 2;
        }

    </style>
</head>

<body>
    <div class="container">
        <div class="box">1</div>
        <div class="box">2</div>
        <div class="box box3">3</div>
        <div class="box">4</div>
    </div>
</body>

</html>
```

Now, whenever we are downsizing the screen box will be the first one to shrink the most.

Similarly there is a property called flex-grow

**flex-grow:**

- This defines the ability for a flex item to grow if necessary. It accepts a unitless value that serves as a proportion. It dictates what amount of the available space inside the flex container the item should take up.

lets see a basic example:

```
.box3{
    flex-shrink: 2;
}
```

Now the box3 will grow the most on increasing the screen size

**Responsiveness:** Responsiveness in web design refers to the ability of a website or web application to adapt and provide an optimal user experience across various devices and screen sizes. A responsive design ensures that the content and layout of a website adjust dynamically based on the device's characteristics, such as screen width, height, and orientation. This approach enhances usability and accessibility, offering a seamless experience for users on desktops, laptops, tablets, and smartphones.

**Media Queries Analogy:** Imagine you're a chef preparing a dish. Your goal is to make the dish appealing to different types of eaters, each with specific preferences. Some like it spicy, some like it mild, and others prefer a balance. To achieve this, you use different recipes (styles) based on the preferences of the eater (device). In the world of web design, media queries are like the recipes that determine how your website looks and behaves based on the characteristics of the device viewing it.

**Media Queries in Code:** Media queries are part of CSS (Cascading Style Sheets) and allow you to apply styles based on certain conditions, typically related to the device's features. Here's a basic example:

Here's a simple HTML document to complement the CSS with media queries:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Responsive Design Example</title>
  <style>
    /* General styles for all devices */
    body {
      font-size: 16px;
      margin: 20px;
    }

    /* Media query for devices with a screen width of 600 pixels or less */
    @media only screen and (max-width: 600px) {
      body {
        font-size: 14px;
        background-color: lightcoral;
      }
    }

    /* Media query for devices with a screen width between 601 and 900 pixels */
```

```
    @media only screen and (min-width: 601px) and (max-width: 900px) {
      body {
        font-size: 18px;
        background-color: lightblue;
      }
    }

    /* Media query for devices with a screen width of 901 pixels or more */
    @media only screen and (min-width: 901px) {
      body {
        font-size: 20px;
        background-color: lightgreen;
      }
    }
  </style>
</head>
<body>
  <h1>Responsive Design Example</h1>
  <p>This is a simple example demonstrating responsiveness using media queries.</p
</body>
</html>
```

This HTML document includes a basic structure with a title, meta tags for viewport settings, and a styled body. The body styles are defined in the embedded CSS, which includes media queries for different screen widths. The background colors are added to make it visually evident when the media queries are applied. Feel free to customize the content and styles according to your preferences and requirements.

## Resources for Further Exploration

To deepen your understanding and gain practical experience, here are some resources you can explore:

### CSS Flexbox:

1. **MDN Web Docs - Flexbox Guide:**
    - Flexbox - MDN Web Docs
2. **CSS-Tricks - A Complete Guide to Flexbox:**
    - A Complete Guide to Flexbox
3. Flexbox Froggy: While focused on Flexbox, it's a great resource to reinforce general layout concepts.

### Media Queries:

1. **MDN Web Docs - Media Queries:**

- Using media queries - MDN Web Docs

2. **CSS-Tricks - Introduction to CSS Media Queries:**
    - Introduction to CSS Media Queries