`flex-wrap: wrap;`

when you set:

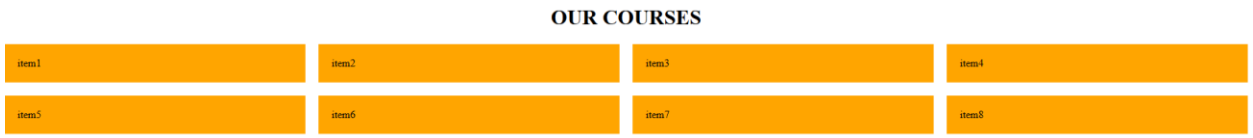
```
flex-wrap: wrap;
```

You are **allowing flex items to wrap onto multiple lines** when they don't fit on one row.

With `flex-wrap: wrap;`

- When items don’t fit, they **automatically move to the next row**.
- The layout becomes **responsive** and adapts to different screen sizes.

  As shown below :-



.

□ Summary

Property	Effect
`flex-wrap: wrap`	Items wrap to new lines when necessary — **responsive layout**. □

Let's go through this CSS code **line by line** and explain what it's doing.

□ Overview:

This code defines styles for a **container** and **item** class using **Flexbox layout**, which is a powerful layout system in CSS used for arranging items in rows or columns efficiently, even when their size is dynamic.

□ `.container` Class

```
.container {
    display: flex;
    flex-wrap: wrap;
    gap: 20px;
}
```

□ Explanation:

1. **`display: flex;`**
    - This turns the `.container` into a **flex container**, meaning its children (with class `.item`) become **flex items**.
    - They are laid out in a row by default (`flex-direction: row` is default).
2. **`flex-wrap: wrap;`**
    - Normally, flex items try to fit in a single line. With `wrap`, if they don't fit, they **wrap onto the next line**.
    - This makes the layout **responsive** — items flow to the next row on smaller screens.
3. **`gap: 20px;`**
    - Adds **20px of space** between the items **both horizontally and vertically**.
    - It's a modern, cleaner alternative to using `margin`.

□ `.item` Class

```
.item {
    flex: 1 1 calc(25% - 20px);
    padding: 20px;
    background-color: orange;
    box-sizing: border-box;
}
```

□ Explanation:

`flex: 1 1 calc(25% - 20px);`
This shorthand is for:

```
flex-grow: 1;
flex-shrink: 1;
flex-basis: calc(25% - 20px);
```

Let's break it down:

- **`flex-basis: calc(25% - 20px);`**
  - The **initial size** of each item is `25%` of the container's width **minus 20px**, accounting for the `gap`.
  - This means ideally, **4 items per row**, but space for the gap is also subtracted.
- **`flex-grow: 1;`**
  - If there's **extra space**, the item **can grow** to fill it proportionally with other items.
- **`flex-shrink: 1;`**
  - If there's **not enough space**, the item can **shrink** to fit.

2. **`padding: 20px;`**
   - Adds **inner space** of 20px inside the item — between content and border.
   - Makes the content not touch the edges.
3. **`background-color: orange;`**
   - Gives each item an orange background.
4. **`box-sizing: border-box;`**
   - Ensures that `padding` is **included** in the element's total width/height.
   - So if the width is `calc(25% - 20px)`, padding won't make the item wider — it's already accounted for.

□ How it works together

Visual Layout:

- `.container` has **flex layout** that wraps rows and has **20px spacing**.
- Each `.item` is approximately **25% of the container width**, minus spacing.
- So, in one row, **4 items** will fit nicely (on wide screens).
- On smaller screens, items will **wrap to new lines**, keeping the layout responsive.

Step 1:-

Create a external css file main.css :-

```
.container {
        display: flex;
        flex-wrap: wrap;
        gap: 20px;
    }

.item {
        flex: 1 1 calc(25% - 20px);
        padding: 20px;
        background-color: orange;
        box-sizing: border-box;
        text-align: center;
        font-weight: bold;
        color: white;
    }
```

Step 2:-

And link this css file into main.html file:-

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Flexbox Grid Example</title>
   <link href="main.css" rel="stylesheet">
</head>
<body>

   <div class="container">
      <div class="item">Item 1</div>
      <div class="item">Item 2</div>
      <div class="item">Item 3</div>
      <div class="item">Item 4</div>
      <div class="item">Item 5</div>
```

```
        <div class="item">Item 6</div>
        <div class="item">Item 7</div>
        <div class="item">Item 8</div>
    </div>

</body>
</html>
```

□ output Example Result:-

On a large screen, the layout might look like:

```
[ Item 1 ][ Item 2 ][ Item 3 ][ Item 4 ]
[ Item 5 ][ Item 6 ][ Item 7 ][ Item 8 ]
```