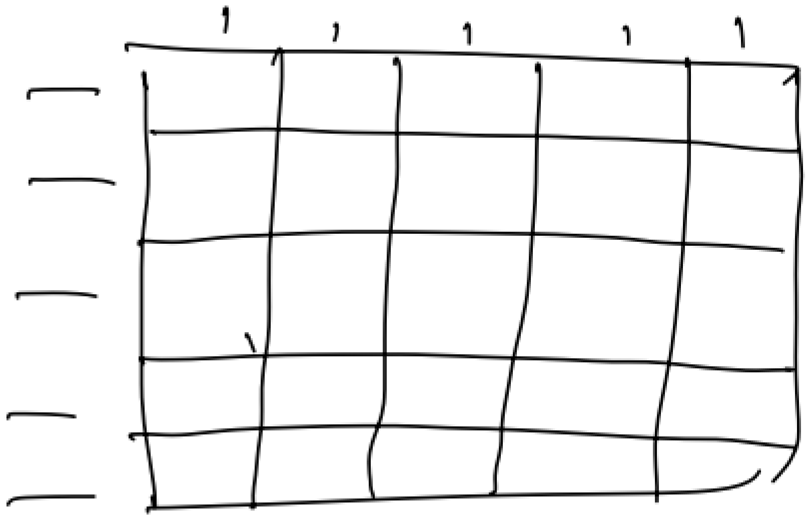
Class 6 - grid

Grid

1. Sometimes, when you have to create a complex layout, you can use a Grid.



This is basically a grid structure.

CSS Grid Layout is a layout in CSS that allows you to create two-dimensional grid layouts for arranging and aligning elements on a web page. It provides a powerful way to structure and design both rows and columns of content, allowing for complex and flexible layouts.

Grid Code

1. Let's create a basic layout of 9 items which we will use to understand Grid.
2. HTML

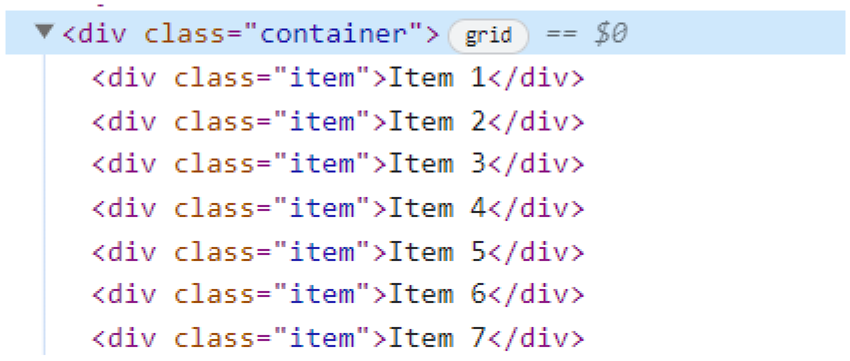
```
<body>
   <div class="container">
       <div class="item">Item 1</div>
       <div class="item">Item 2</div>
       <div class="item">Item 3</div>
       <div class="item">Item 4</div>
       <div class="item">Item 5</div>
       <div class="item">Item 6</div>
       <div class="item">Item 7</div>
       <div class="item">Item 8</div>
       <div class="item">Item 9</div>
   </div>

</body>
```

3. Style tags

```
<title>Grid</title>
   <style>
        *{
           padding: 0;
           margin: 0;
       }

       body{
           font-size: 20px;
           line-height: 1.5;
```

```
            color: dodgerblue;
            background-color: tomato;
        }

        .container{
            width: 1200px;
            margin: 100px auto;
            padding: 10px;
            display : grid;
            /* display: flex; */
            border: 2px solid black;
            width:60%;
        }

        .item{
            background-color: dodgerblue;
            color: white;
            font-size: 20px;
            padding: 20px;
            border: 1px solid lightblue;
        }
    </style>
```

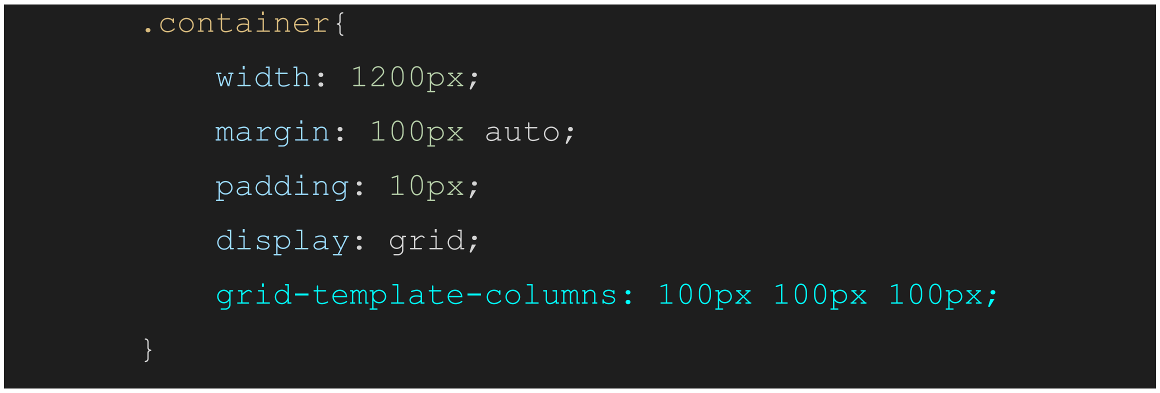
4. Suppose we want to arrange these items in a row order, so what properties can I use?
5. display: flex;. In last lecture we learn't about flex, so giving display as flex will make the items align in row manner.
6. Now, what happens if I give display property as Grid?

7. Nothing will change, they will look the same, but if you go to inspect, you can see a Grid keyword to the container.
8. 

```
▼<div class="container"> grid == $0
    <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>
```

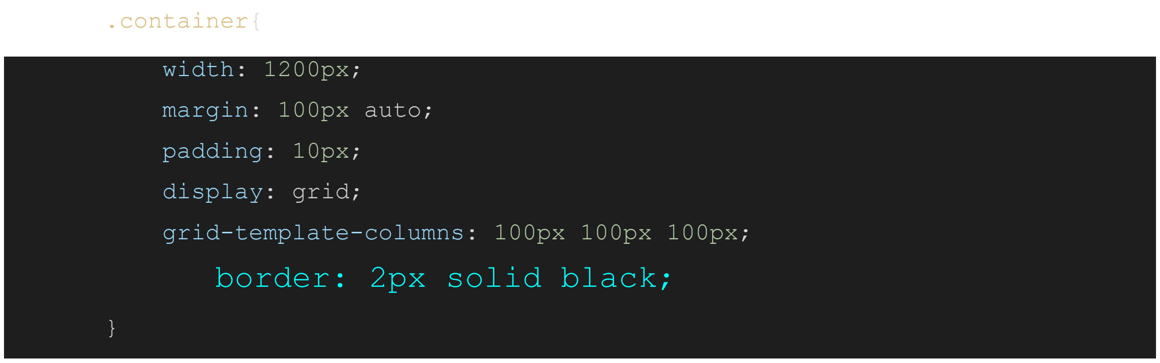
9. It tells that this container is assigned to a grid value, so we are allowed to use any grid property available with the CSS.

Grid-template-columns

1. 

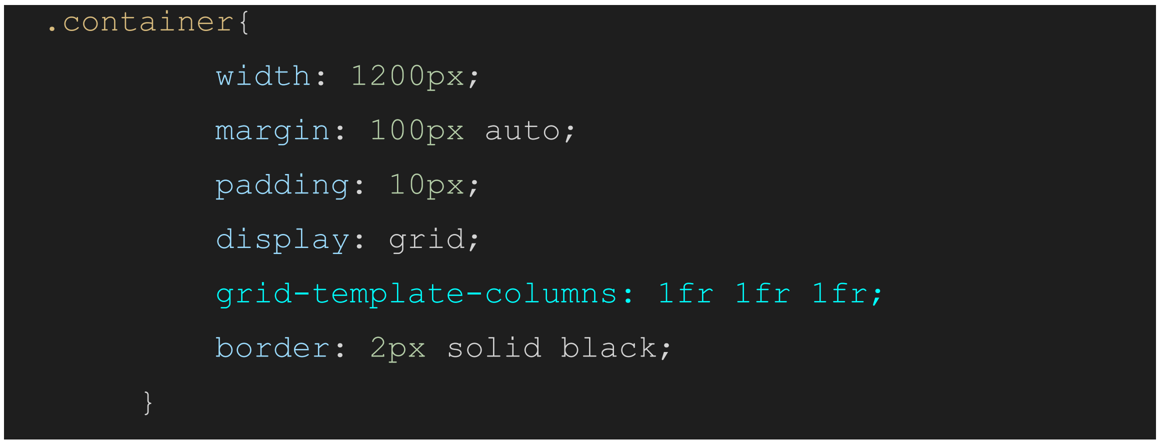
```
        .container{
            width: 1200px;
            margin: 100px auto;
            padding: 10px;
            display: grid;
            grid-template-columns: 100px 100px 100px;
        }
```

2. Here grid will be with three columns, each with a width of 100px. This means that the entire grid will be 300px wide, and each of the nine grid items will occupy a cell that is 100px wide having 3 rows of the grid.

3. Similarly if we remove 1 column of 100px (grid-template-columns: 100px 100px) from the above property, it will create 2 columns of 100px each.
4. 

```
       .container{
          width: 1200px;
          margin: 100px auto;
          padding: 10px;
          display: grid;
          grid-template-columns: 100px 100px 100px;
             border: 2px solid black;
       }
```

5. So here 300px will be taken by the items but the rest 900px will be completely empty. So we are not making any good use of the space. To fix this Grid provides fractional units to handle this.



```
 .container{
          width: 1200px;
          margin: 100px auto;
          padding: 10px;
          display: grid;
          grid-template-columns: 1fr 1fr 1fr;
          border: 2px solid black;
       }
```

6. It will take the entire space and equally divide 3 items in a row. The grid-template-columns: 1fr 1fr 1fr; declaration means that the available space in the container's width will be divided into three equal parts, each represented by 1fr. This is a flexible way to distribute space evenly among the columns.
7. Let's understand it that way:

    a. suppose the container is of length 120px.
    b. grid-template-columns: 1fr 1fr 1fr;
    c. Total frames will be 1+1+1 = 3.
    d. size of each frame will be 120/3 = 40px.
    e. Hence in this case, each column of the three columns will be equal to 40px.

8. Now lets give the units like this:

```
        .container{
            width: 1200px;
            margin: 100px auto;
            padding: 10px;
            display: grid;
            grid-template-columns: 1fr 2fr 1fr;
            border: 2px solid black;
        }
```

9. Now with grid-template-columns: 1fr 2fr 1fr;, you are distributing the available space into three columns with different proportions. The first column will take up 1fr that is 300px of the available space, the second column will take up 2fr that is 600px, and the third column will take up 1fr or 300px.
10.

gap

1. Column gap
2. Now if we want to give some gap between the columns, we can use column-gap property

```
.container{
    width: 1200px;
    margin: 100px auto;
    padding: 10px;
    display: grid;
    grid-template-columns: 1fr 2fr 1fr;
    column-gap: 10px;
}
```

3. Row gap
4. Similarly if we want to give some gap between the rows, we can use row-gap property

```
.container{
    width: 1200px;
    margin: 100px auto;
    padding: 10px;
    display: grid;
    grid-template-columns: 1fr 2fr 1fr;
    column-gap: 10px;
    row-gap:10px;
}
```

5. Gap
6. If you want to give gap to both rows and columns at once. you can give gap property only.

```
.container{
          width: 1200px;
          margin: 100px auto;
          padding: 10px;
```

```
        display: grid;
        grid-template-columns: 1fr 2fr 1fr;
        /* column-gap: 10px;
        row-gap:10px; */
        gap:10px;
    }
```

grid-template-rows

1. Let's now see row-level property of grid.
   ```
       .container{
           width: 1200px;
           margin: 100px auto;
           padding: 10px;
           display: grid;
           grid-template-columns: 1fr 2fr 1fr;
           grid-template-rows: 1fr 2fr 1fr;
       }
   ```
2. The grid-template-rows: 1fr 2fr 1fr; declaration defines the height proportions of the grid's rows. The first row will take up 1fr, the second row will take up 2fr, and the third row will take up 1fr.
3. Similar to how fr units distribute space among columns, here the vertical space is distributed among rows. The second row will be twice as tall as the first and third rows.

grid-template-columns with repeat()

1. In most common use cases, we want our column sizes to be same
2. Writing out each column size can be repetitive, especially for large grids. That's where repeat() comes in. It lets you specify how many times a pattern of columns should repeat.
3. Basic Syntax: repeat(number_of_repetitions, column_width). For example, grid-template-columns: repeat(3, 100px); creates three columns, each 100 pixels wide
4. Flexible Layouts: repeat() is very useful for creating grids with consistent column sizes. It makes your code cleaner and more readable.

```css
        .container{
            width: 800px;
            margin: 100px auto;
            padding: 10px;
            display: grid;
            grid-template-columns: repeat(3, 1fr);
            grid-template-rows: 1fr 2fr 1fr;
        }
```

5. This method can also be applied to row level grid.

Combining Fixed and Flexible Sizes

Mixing Fixed and Auto: You can mix fixed widths with auto to create grids where some columns have a specific size, and others adjust based on content size. For example

   a. **grid-template-columns: 100px auto 200px;**
      i. This sets up a grid with three columns where the first is 100px wide, the second adjusts automatically, and the third is 200px wide.
2. Using fr Unit for Flexible Columns: The fr unit represents a fraction of the available space in the grid container. It's great for creating flexible grids where columns take up a portion of the space. For example:
   a. **grid-template-columns: 1fr 2fr 1fr;**
   b. This creates a three-column grid where the middle column is twice as wide as the side columns.
3. Advanced Patterns with repeat()
   a. Combining repeat() with Fixed and Flexible Sizes: You can use repeat() alongside fixed sizes and the fr unit. For instance:
   b. **grid-template-columns: repeat(2, 1fr) 100px;**
   c. This creates a grid with three columns: two flexible columns (each taking up an equal fraction of the available space) and a third 100px wide column.
4. Creating Complex Grids: You can create more complex patterns by repeating a set of column sizes. For example:

a. **grid-template-columns: repeat(3, 1fr 2fr);**
b. This repeats the pattern 1fr 2fr three times, resulting in a six-column grid with alternating column sizes.

Grid-auto-rows (extra read , do explore)

1. if you know of the structure in advance then you can give row level height and sizes before hand but what if content gets dynamically added to your grid like a new blog post or user comments. then in that case to ensure that your newly generated rows do not look different from the previous ones we can give this grid auto row property
2. This property allows you to control the height of these dynamically added rows.

Pseudo class Item level selection in grid

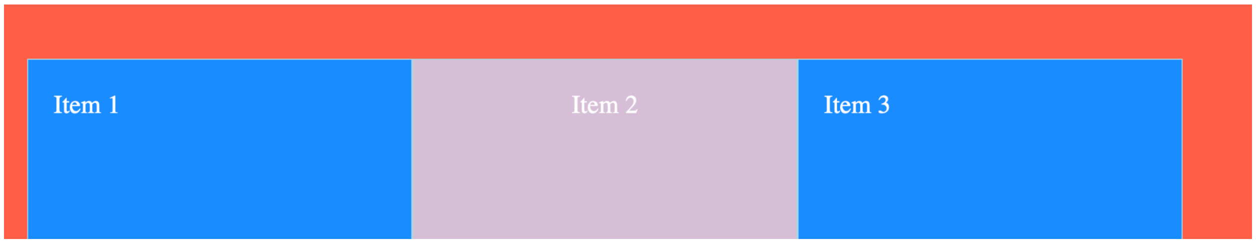
1. In our grid, all the items have the same class

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

2. Suppose we want to select a particular item without adding any additional class or id then how do we do it

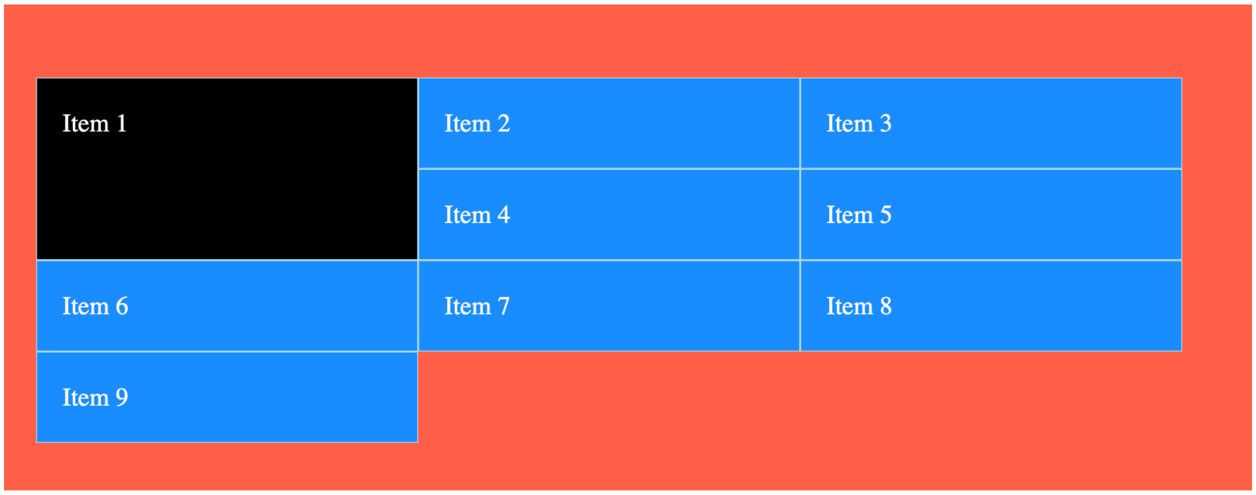
3. In understanding this, we will also learn about pseudo class selectors
4. In CSS, a pseudo-class is a keyword added to selectors that specifies a special state of the selected elements. For example, :hover applies a style when the user hovers over an element.
5. In our case, we will use **item:nth-of-type()** pseudo-selector.
6. :nth-of-type() pseudo-class selector in CSS is a powerful tool for selecting specific elements among a group of similar siblings based on their position
7. :nth-of-type() pseudo-class targets elements based on their position within a group of siblings of the same type.

```
.item:nth-of-type(2){
            background-color: thistle;
            text-align: center;
        }
```



Spanning across multiple rows and columns

1. Now, let's take 1 more example, where you want item 1 to take space of item 4 vertically. So how we can do this?
2. Like below



3.

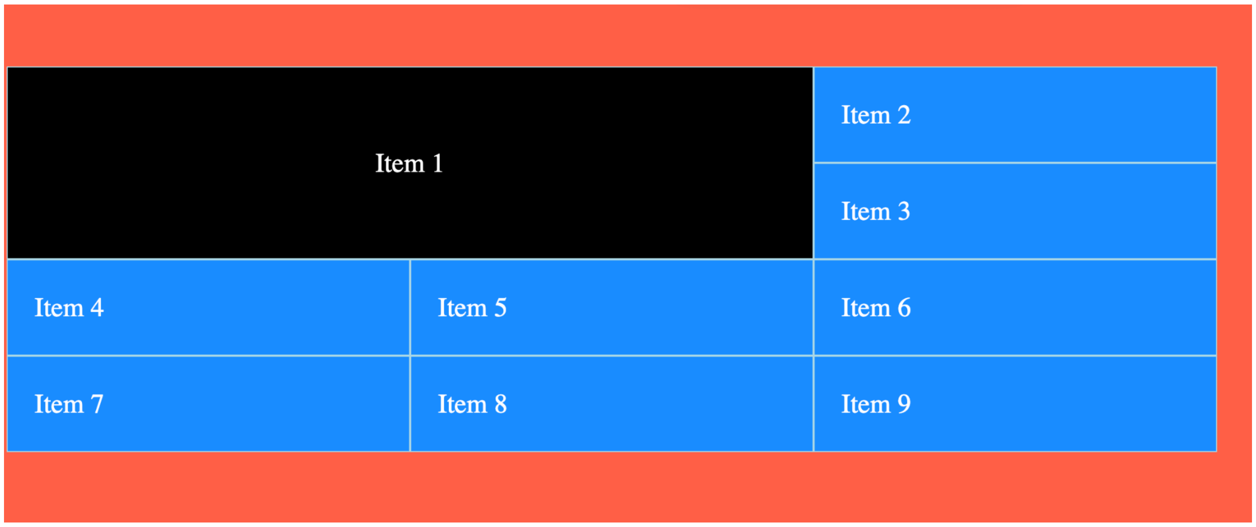
```css
.item:nth-of-type(1){
          background: black;
          grid-row: 1/3;
      }
```

8. Similarly if you want this to span across two cols

```css
     .item:nth-of-type(1){
          background: black;
          grid-row: 1/3;
          grid-column: span 2;
      }
```

9. You can use flex aproperties here as well to center everything

```css
.item:nth-of-type(1){
          background: black;
          grid-row: 1/3;
          grid-column: span 2;
          display: flex;
          justify-content: center;
          align-items: center;
      }
```

10. 

Grid Areas

1. Now lets say our task is to create this layout
2. https://camo.githubusercontent.com/748f25f7c7a1aa6fb12720ff8e4a4b2d5b181792af1aee62effcf9a87f0c2ee0/687474707333a2f2f6432626569716b687139323966302e636c6f756466726f6e742e6e65742f7075626c69635f6173736574732f617373657473 2f3030302f3035302f3037392f6f726967696e616c2f75706c6f61645f306364376562393937343535316237323031356332376265643938633533233362e706e673f3136393533313373393236
3. How many grid items do you see here - 6
4. Lets first define all the components in our HTML page, header, navigation, sidebar, footer.
5. Create a new file gridLayout.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Grid Layout</title>
</head>
<style>
   *{
       margin: 0;
       padding: 0;
   }

   body{
       font-size: 20px;
       height: 100vh;
   }
</style>
<body>
    <header>Header</header>
    <main>Content</main>
    <nav>Navigation Bar</nav>
    <aside>SideBar</aside>
    <footer>footer</footer>
</body>
</html>
```

6. for initializing grid in our application, we can use display: grid; property.

```
   body{
       font-size: 20px;
       height: 100vh;
       display: grid;
   }
```

7. now lets give some borders and colors to all the elements.

```
header, main, nav, aside, footer{
       background-color: tomato;
       color: white;
```

```
      padding: 20px;
      border: skyblue 1px solid;
  }
```

8. Based on what we have learnt, let us approach this and then we ll see a diff way

```
  body{
      font-size: 20px;
      height: 100vh;
      display: grid;
      grid-template-columns: repeat(4, 1fr);
  }
```

```
header{
      grid-column: 1/5;
      text-align: center;
  }
```

9. Similarly , we can give ratios like this for every tag but as you would be feeling that this is not that straightforward

Grid Areas

1. grid areas are named regions of a grid to place items into.
2. grid-template-areas allows you to create a mental picture of the layout.
3. define grid areas using grid-template-areas property.
4. To convert this layout to our desired layout, we can use 1 css property called grid-template-areas, where we can tell CSS what area to give to a specific section

Code

1. Since our design has two sidebars, let us change our markup

```
<body>
   <header>Header</header>
   <main>Content</main>
   <nav>Navigation Bar</nav>
   <aside>SideBar</aside>
   <aside class="sidebar2">Sidebar 2</aside>
   <footer>footer</footer>
 </body>
```

2. Now we create named grid areas

```
header,
   main,
   nav,
   aside,
   footer {
     background-color: tomato;
     color: white;
     padding: 20px;
     border: skyblue 1px solid;
   }
/*
*This means that the <header> element will occupy the
entire area named "header" in the grid.
*/
   header {
     grid-area: header;
   }
```

```
nav {
  grid-area: nav;
}

main {
  grid-area: content;
}

aside {
  grid-area: sidebar;
}
.sidebar2 {
  grid-area: sidebar2;
}
footer {
  grid-area: footer;
}
```

3. Now we define our grid area based on the names that we have given

```
body {
     font-size: 20px;
     height: 100vh;
     display: grid;
     grid-template-areas:
       "header header header header"
       "nav content sidebar sidebar2"
       "nav footer footer footer";
```

```
  }
```

4. You can think of it as a visual map of your layout
5. Each " " (space) represents a cell in the grid, and each line of quotes " " represents a row.
6. Naming areas: within the quotes, we give names to different parts of the grid, like "header", "footer", "sidebar", etc. These names can be anything that makes sense for the layout you're trying to achieve.
7. Now only height and width of the templates is left, so we can use grid-template-columns and grid-template-rows properties:

```
body{
      font-size: 20px;
      height: 100vh;
      display: grid;
      grid-template-areas:
      "header header header header"
      "nav content sidebar sidebar2"
      "nav footer footer footer";

      grid-template-columns: 1fr 4fr 1fr 1fr;
      grid-template-rows: 80px 1fr 70px;
  }
```

8. grid-template-columns: 1fr 4fr 1fr 1fr; defines the widths of the columns in the grid. The first column takes up 1 fraction of the

available space, the second column takes up 4 fractions, and the last two columns take up 1 fraction each.

9. grid-template-rows: 80px 1fr 70px; defines the heights of the rows in the grid. The first row has a fixed height of 80px, the second row takes up 1 fraction of the available space, and the third row has a fixed height of 70px.

Flexbox vs Grid

1. Flexbox One-Dimensional Layout: Flexbox is primarily used for laying out items in a single dimension, either a row or a column.
    a. Content-First Approach: It's ideal when the size of the flex container's content or the number of items is dynamic or unknown.
    b. Alignment and Distribution: Easily align items vertically or horizontally and distribute space between items.
    c. Key Properties:
        i. display: flex defines a flex container.
        ii. flex-direction: Specifies the direction of the main axis (row, row-reverse, column, column-reverse).
        iii. justify-content: Aligns items along the main axis.
        iv. align-items: Aligns items along the cross axis.
        v. flex-wrap: Defines whether items should wrap into multiple lines or not.
    d. Use Flexbox when:

        i. You're aligning a navigation bar, centering an item on the screen, or building a small-scale layout with a linear flow.
        ii. Ideal for one-dimensional layouts, such as navigation bars, headers, footers, and simple component arrangements.
    e. Suited for aligning items within a container along a single axis.
    f. Great for creating responsive designs when dealing with dynamic content.

2. Grid Two-Dimensional Layout: Grid allows for laying out items in both rows and columns simultaneously.
    a. Layout-First Approach: Best for when you have a clear idea of the layout, regardless of the content.
    b. Grid Areas: Enables creating complex layouts by defining areas in a grid.
    c. Consistent Sizing: Offers control over row and column sizing, and the ability to align the entire grid. Complex Layouts: More suitable for larger, more complex layouts.
    d. Key Properties:
        i. display: grid: Defines a grid container.
        ii. grid-template-rows and grid-template-columns: Specifies the size and structure of the grid.
        iii. grid-gap or grid-row-gap and grid-column-gap: Defines the space between grid items.

    iv. grid-template-areas: Assigns names to areas of the grid..

e. Use Grid when:
    i. Creating a complex web page layout with multiple rows and columns, like a photo gallery or a magazine-style layout,image galleries, and responsive dashboards.
    ii. Well-suited for complex layouts with both rows and columns.
    iii. Provides a powerful solution for defining the placement and alignment of items in a grid.