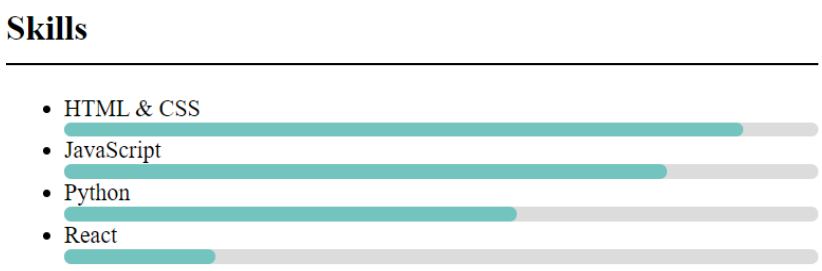
Skill Indicator:

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

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    .box {
      width: 40vw;
      /* border: 2px solid green; */
    }
    h2 {
      border-bottom: 2px solid;
      padding-bottom: 10px;
    }
    .skill-indicator {
      background-color: #e0e0e0;
      border-radius: 5px;
      height: 10px;
     }
    .skill-level {
      height: 100%;
      background-color: #76c7c0;
```

```
      /* Skill level color */
      border-radius: 5px;
     }
  </style>
</head>
<body>
  <div class="box">
    <h2>Skills</h2>
    <ul>
      <li>
        HTML & CSS
        <div class="skill-indicator">
          <div class="skill-level" style="width: 90%;"></div> <!-- Adjust width
for proficiency -->
        </div>
      </li>
      <li>
        JavaScript
        <div class="skill-indicator">
          <div class="skill-level" style="width: 80%;"></div> <!-- Adjust width
for proficiency -->
        </div>
      </li>
      <li>
        Python
        <div class="skill-indicator">
```

```
          <div class="skill-level" style="width: 60%;"></div> <!-- Adjust width
for proficiency -->
        </div>
      </li>
      <li>
        React
        <div class="skill-indicator">
          <div class="skill-level" style="width: 20%;"></div> <!-- Adjust width
for proficiency -->
        </div>
      </li>
    </ul>
  </div>
</body>
</html>
```

Output:



Converting the following Resume Code into the Flex and Skill indicator:

Without Flex:

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Resume Layout</title>
</head>

<body>
    <div class="container">
        <!-- Header Section -->
        <div class="header">
            <div class="info">
                <h1>John Doe</h1>
                <p>Email: john.doe@example.com</p>
                <p>Phone: +123 456 789</p>
            </div>
            <div class="photo">
                <!-- Placeholder for photo -->
                <img src="https://via.placeholder.com/100" alt="Profile Photo">
            </div>
        </div>

        <!-- Main Content Layout -->
```

```
<div class="main-content">
    <!-- Sidebar -->
    <div class="sidebar">
        <!-- Skills Section -->
        <div class="skills">
            <h2 class="section-title">Skills</h2>
            <ul>
                <li>HTML & CSS</li>
                <li>JavaScript</li>
                <li>Python</li>
                <li>React</li>
            </ul>
        </div>

        <!-- Education Section -->
        <div class="education">
            <h2 class="section-title">Education</h2>
            <ul>
                <li>B.Sc. in Computer Science</li>
                <li>M.Sc. in Data Science</li>
            </ul>
        </div>
    </div>

    <!-- Main Content -->
    <div class="content">
```

```
                <!-- Experience Section -->
                <div class="experience">
                    <h2 class="section-title">Experience</h2>

                    <!-- Job 1 -->
                    <div class="job">
                        <div class="job-title">Software Developer at XYZ
Corp</div>
                        <div class="job-duration">Jan 2020 - Present</div>
                        <p>Worked on developing scalable web applications using
JavaScript frameworks such as React and Node.js.</p>
                    </div>

                    <!-- Job 2 -->
                    <div class="job">
                        <div class="job-title">Web Developer at ABC Inc</div>
                        <div class="job-duration">Jun 2018 - Dec 2019</div>
                        <p>Designed and implemented responsive web pages using
HTML, CSS, and JavaScript, enhancing user experience and performance.</p>
                    </div>
                </div>
            </div>
        </div>
    </div>
</body>

</html>
```

Output:

John Doe

Email: john.doe@example.com

Phone: +123 456 789

Skills

- HTML & CSS
- JavaScript
- Python
- React

Education

- B.Sc. in Computer Science
- M.Sc. in Data Science

Experience

Software Developer at XYZ Corp
Jan 2020 - Present

Worked on developing scalable web applications using JavaScript frameworks such as React and Node.js.

Web Developer at ABC Inc
Jun 2018 - Dec 2019

Designed and implemented responsive web pages using HTML, CSS, and JavaScript, enhancing user experience and performance.

With Styling with Flex and Skill Indicator:

```
<!DOCTYPE html>

<html lang="en">


<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Resume Layout</title>
```

```
<style>
  body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
  }

  .container {
    max-width: 900px;
    /* Margin from top and bottom 0, and from left and right auto */
    margin: 0 auto;
    padding: 20px;
  }

  /* Header Section */
  .header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    border-bottom: 2px solid #333;
    padding-bottom: 15px;
    margin-bottom: 20px;
  }
```

```css
/* Sidebar and Main content layout */
.main-content {
  display: flex;
  justify-content: space-between;
}

.sidebar {
  flex: 1;
  margin-right: 20px;
}

.content {
  flex: 2;
}

/* Sidebar Sections */
.section-title {
  font-size: 18px;
  font-weight: bold;
  margin-bottom: 10px;
  border-bottom: 2px solid #333;
}
```

```css
.skills ul,
.education ul {
  list-style-type: none;
  padding: 0;
}

.skills ul li,
.education ul li {
  margin-bottom: 10px;
}

/* Main content Sections */
.job-title {
  font-weight: bold;
}

.job-duration {
  font-style: italic;
  color: gray;
}
```

```
    .skill-indicator {

      background-color: #e0e0e0;
      border-radius: 5px;
      height: 10px;
     }

    .skill-level {
      height: 100%;
      background-color: #76c7c0;
      /* Skill level color */
      border-radius: 5px;
    }
  </style>
</head>



<body>
  <div class="container">
    <!-- Header Section -->
    <div class="header">
      <div class="info">
        <h1>John Doe</h1>
        <p>Email: john.doe@example.com</p>
        <p>Phone: +123 456 789</p>
      </div>
```

```
      <div class="photo">
        <!-- Placeholder for photo -->
        <img src="https://via.placeholder.com/100" alt="Profile Photo">
      </div>
    </div>



    <!-- Main Content Layout -->
    <div class="main-content">
      <!-- Sidebar -->
      <div class="sidebar">
        <!-- Skills Section -->
        <div class="skills">
          <h2 class="section-title">Skills</h2>
          <ul>
            <li>
              HTML & CSS

              <div class="skill-indicator">
                <div class="skill-level" style="width: 90%;"></div> <!--
Adjust width for proficiency -->
              </div>
            </li>
            <li>
              JavaScript

              <div class="skill-indicator">
```

```
                <div class="skill-level" style="width: 80%;"></div> <!--
Adjust width for proficiency -->
              </div>
            </li>
            <li>
              Python

              <div class="skill-indicator">
                <div class="skill-level" style="width: 60%;"></div> <!--
Adjust width for proficiency -->
              </div>
            </li>
            <li>
              React

              <div class="skill-indicator">
                <div class="skill-level" style="width: 20%;"></div> <!--
Adjust width for proficiency -->
              </div>
            </li>
          </ul>
        </div>



        <!-- Education Section -->
        <div class="education">
          <h2 class="section-title">Education</h2>
```

```
            <ul>
              <li>B.Sc. in Computer Science</li>
              <li>M.Sc. in Data Science</li>
            </ul>
          </div>
        </div>


        <!-- Main Content -->
        <div class="content">
          <!-- Experience Section -->
          <div class="experience">
            <h2 class="section-title">Experience</h2>


            <!-- Job 1 -->
            <div class="job">
              <div class="job-title">Software Developer at XYZ Corp</div>
              <div class="job-duration">Jan 2020 - Present</div>
              <p>Worked on developing scalable web applications using
JavaScript frameworks such as React and Node.js.</p>
            </div>


            <!-- Job 2 -->
            <div class="job">
              <div class="job-title">Web Developer at ABC Inc</div>
```

```
            <div class="job-duration">Jun 2018 - Dec 2019</div>

            <p>Designed and implemented responsive web pages using HTML,
CSS, and JavaScript, enhancing user experience

              and performance.</p>

          </div>

        </div>

      </div>

    </div>

  </div>

</body>

</html>
```

Output:

John Doe

Email: john.doe@example.com

Phone: +123 456 789

100 x 100

Skills

HTML & CSS

JavaScript

Python

React

Education

B.Sc. in Computer Science

M.Sc. in Data Science

Experience

Software Developer at XYZ Corp

Jan 2020 - Present

Worked on developing scalable web applications using JavaScript frameworks such as React and Node.js.

Web Developer at ABC Inc

Jun 2018 - Dec 2019

Designed and implemented responsive web pages using HTML, CSS, and JavaScript, enhancing user experience and performance.

CSS Media Queries:

CSS Media Queries are a way for web developers to apply different styles to a webpage based on various factors like the size of the screen, the device being used, or even the orientation of the device.

Think of it like this: Imagine you have a piece of clothing that changes its color depending on the weather. If it's sunny, it turns yellow, if it's raining, it turns blue. Similarly, in web design, media queries let you change how your webpage looks based on different conditions.

For example, you might want your webpage to look one way on a large computer screen, but a bit different on a smaller smartphone screen so that it's easier to read and navigate. With media queries, you can write CSS rules that only apply when certain conditions are met, like when the screen width is less than 600 pixels, indicating a small device.

So, media queries help make websites responsive, adapting to different devices and screen sizes to provide the best user experience possible.

Take a look a the example of @media query

Different type of syntax to use the media query:

```
@media screen and (max-width: 768px) {
   /* CSS styles here */
}
```

or for all the media types:

```
@media  (max-width: 768px) {
   /* CSS styles here */
}
```

Using **@media**, you specify a media query and a block of CSS to apply to the document if and only if the media query matches the device on which the content is being used.

The media type defines what type of media we are targeting:

- **all**: Matches all devices; it is the default type.
- **print**: Matches documents viewed in a print preview or any media that breaks the content up into pages intended to print.
- **screen**: Matches devices with a screen.
- **speech**: Matches devices that read the content audibly, such as a screen reader.

The media feature defines what feature you are trying to match. Some important media features include:

- **width**: Defines the widths of the viewport. This can be a specific number (e.g., 400px) or a range (using min-width and max-width).
- **height**: Defines the height of the viewport. This can be a specific number (e.g., 400px) or a range (using min-height and max-height).
- **aspect-ratio**: Defines the width-to-height aspect ratio of the viewport.
- **orientation**: The way the screen is oriented, such as tall (portrait) or wide (landscape) based on how the device is rotated.

The **@media** rule is itself a logical operator that states "if" the following types and features match, then do some stuff. You can use the and operator to target screens within a range of widths. It's possible to target devices by what they do not support or match.

Using proper breakpoints is advisable as it helps to create a responsive layout. Breakpoints are specific points in CSS where the layout or styling of the website changes to adapt to different screen sizes.

Resolution range	**Layout**
320px – 480px	Mobile devices
481px – 768px	Ipads
769px – 1024px	small screens, laptop
1025px – 1200px	Desktop, large screens
1201px – any	Extra large Screens, TV

Mobile-first responsive design is an approach to website or application design and development where the primary focus is on creating a user experience optimized for mobile devices first, before considering larger screens such as tablets or desktops. This means designing the layout, content, and functionality with smaller screens in mind, and then gradually enhancing the design for larger screens using techniques like responsive grids, flexible images, and media queries.

Syntax:

- A media query consists of a media type and can contain one or more media features, which resolve to either true or false.

```
@media not|only mediatype and (media feature) and (media feature) {
    CSS-Code;
}
```

- The mediatype is optional (if omitted, it will be set to **all**). However, if you use **not** or **only**, you must also specify a mediatype.

  - **not:** This keyword inverts the meaning of an entire media query.

- **only:** This keyword prevents older browsers that do not support media queries from applying the specified styles. It has no effect on modern browsers.
- **and:** This keyword combines a media type and one or more media features.

Using media query with different kinds of operators:

1.
```
@media screen and (max-width: 768px) {
/* CSS styles for screens with a maximum width of 768px */
}
```

2.
```
@media screen and (min-width: 600px) and (max-width: 1024px) {
    /* CSS styles for screens between 600px and 1024px */
}
```

3.
```
@media not screen and (color) {
    /* CSS styles for devices that do not support color */
}
```

Here, the **not** keyword is used to target devices that do not match the specified conditions. In this case, the styles will apply to devices that do not support **color**.

4.
```
@media (orientation: portrait) or (max-width: 600px) {
/* CSS styles for screens in portrait orientation or with a maximum width of 600px */
```

```
    }
5. @media screen, print and (max-width: 768px) {
    /* CSS styles for screens and print media with a maximum width of 768px */
    }
```

Example1:

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>


    <style>
        body {
            background-color: green;
        }

        @media screen and (max-width: 800px) {
            body {
                background-color: yellow;
            }
```

```
        }
        @media screen and (max-width: 600px) {
            body {
                background-color: blue;
            }
        }
        @media screen and (max-width: 400px) {
            body {
                background-color: aqua;
            }
        }
    </style>
</head>
<body>
    <h1>Welcome to Chitkara</h1>
</body>

</html>
```

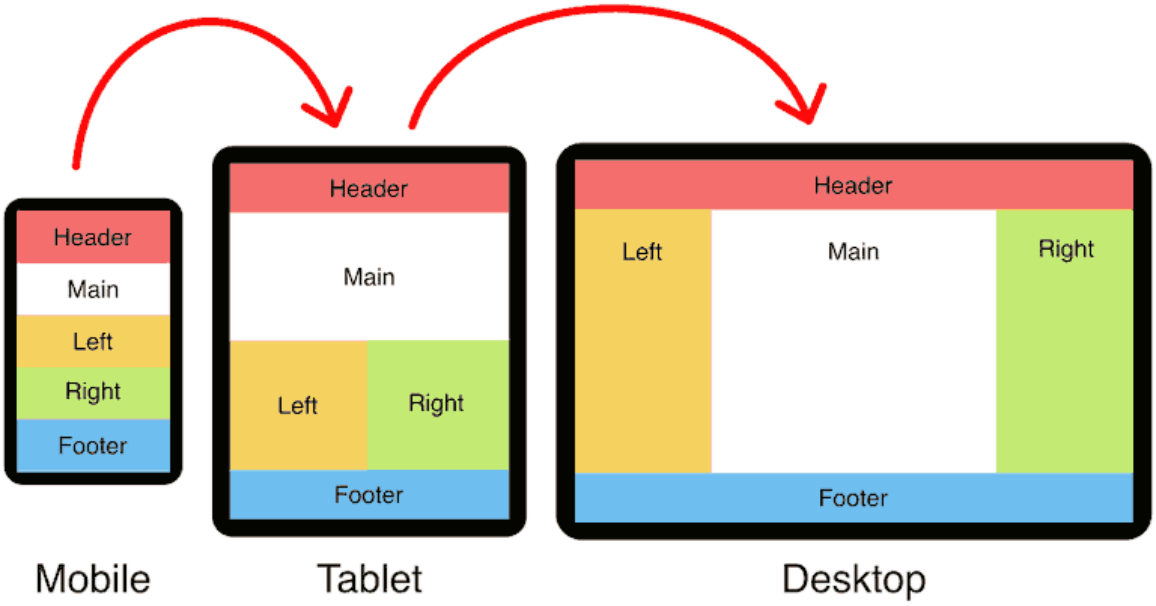
Example2:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Document</title>
    <style>
        body {
            background-color: green;
        }
        @media screen and (min-width: 600px) and (max-width: 800px) {
            body {
                background-color: yellow;
            }
        }
        @media screen and (max-width: 500px) {
            body {
                background-color: red;
            }
        }
    </style>
</head>
<body>
    <h1>Welcome to Chitkara</h1>
</body>
</html>
```

Here till 500 px the color will be red, 500 to 599 the color will be green and 600px to 800px the color will be yellow and after that the color will be again green.

Student Activity for media query:



Solution:

```
<!DOCTYPE html>

<html lang="en">



<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Document</title>

  <style>


    body {

      margin: 0;

      padding: 0;

      height: 100vh;
```

```
  font-size: 25px;
}
header {
  height: 10vh;
  width: 100%;
  background-color: orange;
  font-size: 25px;
  display: flex;
  justify-content: center;
  align-items: center;
}
footer {
  height: 10vh;
  width: 100%;
  background-color: aquamarine;
  display: flex;
  justify-content: center;
  align-items: center;
}
.container {
  height: 80vh;
  display: flex;
}
.left {
  width: 25%;
  background-color: yellow;
```

```
  display: flex;
  justify-content: center;
  align-items: center;
}
.main {
  width: 50%;
  background-color: white;
  display: flex;
  justify-content: center;
  align-items: center;
}
.right {
  background-color: yellowgreen;
  width: 25%;
  display: flex;
  justify-content: center;
  align-items: center;
}

@media (min-width: 769px) and (max-width: 1024px) {
  .main {
    order: -1;
    width: 100%;
  }
  .container {
    flex-wrap: wrap;
```

```
      }
      .left,
      .right {
        width: 50%;
      }
    }
    @media (max-width: 768px) {
      .container {
        flex-direction: column;
        width: 100%;
        /* height: 60vh; */
        height: auto;
      }
      .main {
        order: -1;
      }
      header,
      footer,.left,.right,.main {
        height: 20vh;
        width: 100%;
      }
    }
  </style>
</head>
<body>
  <header>Header</header>
```

```
  <div class="container">
    <div class="left">LeftBar</div>
    <div class="main">Main Area</div>
    <div class="right">RightBar</div>
  </div>
  <footer>Footer</footer>
</body>
</html>
```

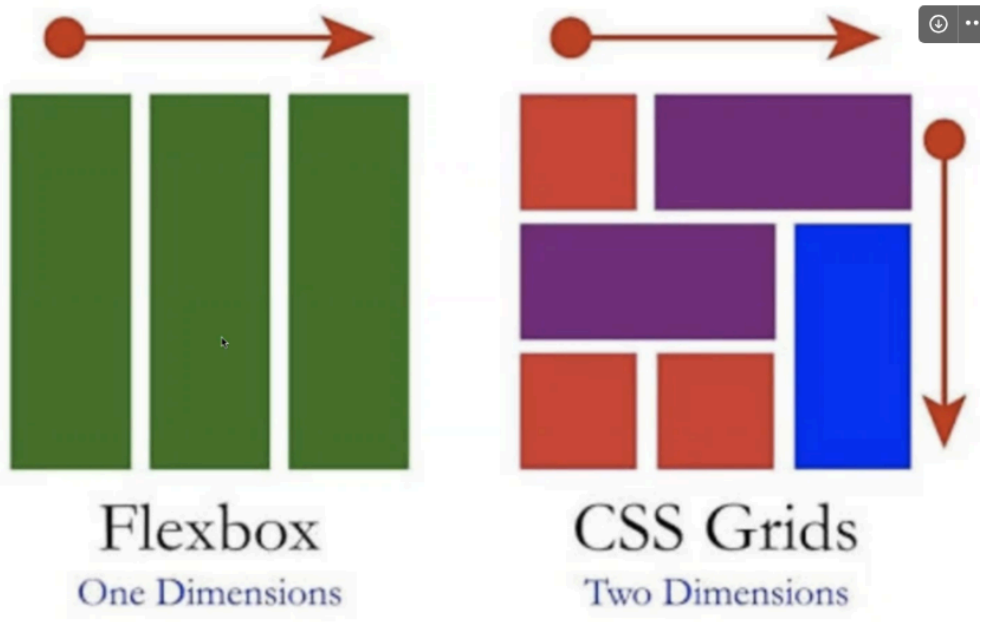
Grid Layout:

CSS Grid Layout, is a two-dimensional grid-based layout system that, compared to any web layout system of the past, completely changes the way we design user interfaces.

CSS has always been used to layout our web pages, but it's never done a very good job of it. First, we used tables, then floats, positioning and inline-block, but all of these methods were essentially hacks and left out a lot of important functionality (vertical centering, for instance).

Flexbox is also a very great layout tool, but its one-directional flow has different use cases and they actually work together quite well!

Grid is the very first CSS module created specifically to solve the layout problems we've all been hacking our way around for as long as we've been making websites.



An HTML element becomes a grid container when its display property is set to **grid**.

Example:

```
.grid-container {
  display: grid;
}
```

All direct children of the grid container automatically become grid items.

Note: The `display: grid` directive only affects a box model and its direct children. It does not affect grandchildren.

To get started you have to define a container element as a grid with **display: grid** set the column and row sizes with **grid-template-columns** and **grid-template-rows**.

Example:

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>



    <style>
        .container {

            border: 2px solid green;
            background-color: antiquewhite;
            padding: 10px;
            display: grid;
            grid-template-columns: auto auto auto;
        }

        .items {
            background-color: aqua;
            border: 1px solid blue;
            font-size: 30px;
            text-align: center;
```

```
        }
    </style>

</head>

<body>

    <h2 align="center">Grid System</h2>

    <div class="container">
        <div class="items">Item1</div>
        <div class="items">Item2</div>
        <div class="items">Item3</div>
        <div class="items">Item4</div>
        <div class="items">Item5</div>
        <div class="items">Item6</div>
        <div class="items">Item7</div>
        <div class="items">Item8</div>
        <div class="items">Item9</div>
    </div>

</body>
</html>
```

Output:

Grid System

Item1	Item2	Item3
Item4	Item5	Item6
Item7	Item8	Item9

Important CSS grid terminologies:

1. **Grid Container:**

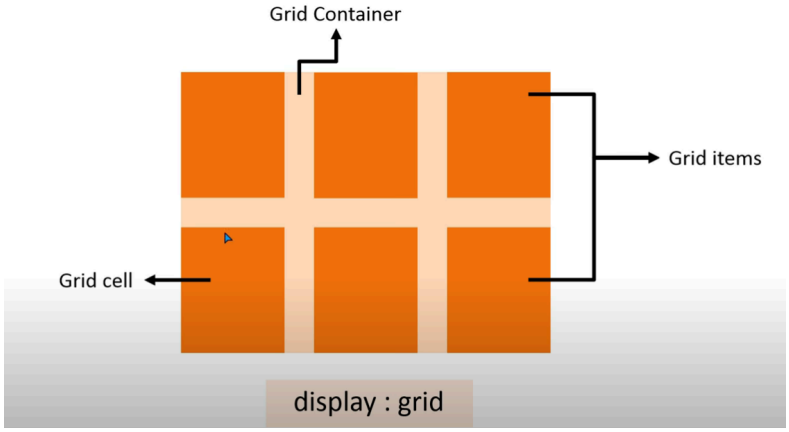
   The element on which display: grid is applied. It's the direct parent of all the grid items.

2. **Grid item:**

   The children (i.e. direct descendants) of the grid container.

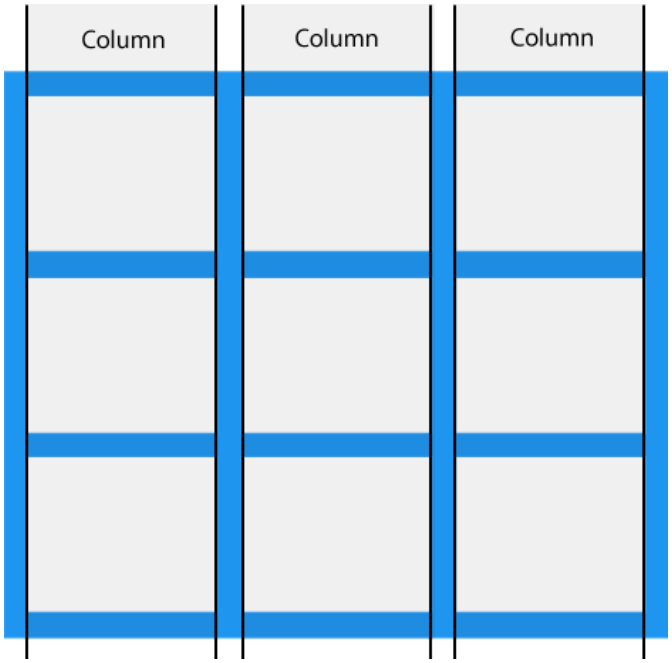
3. **Grid cell:**

   A grid cell is the smallest unit you can have on your CSS grid. It is the space between four intersecting grid lines and conceptually much like a table cell.



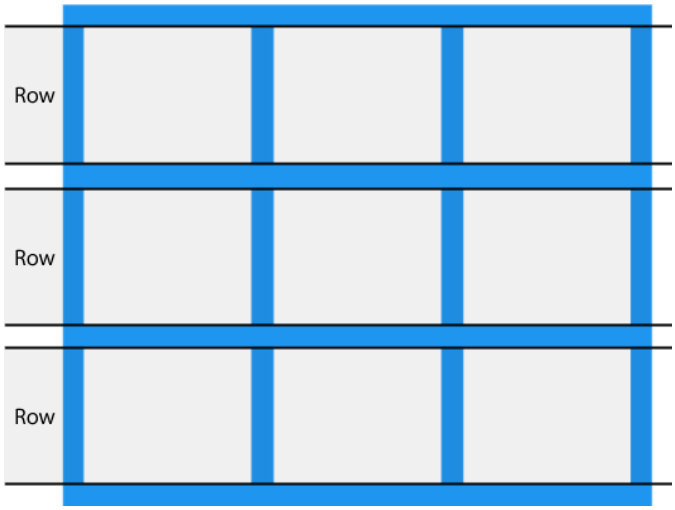
4. **Grid Columns:**

The vertical lines of grid items are called columns.



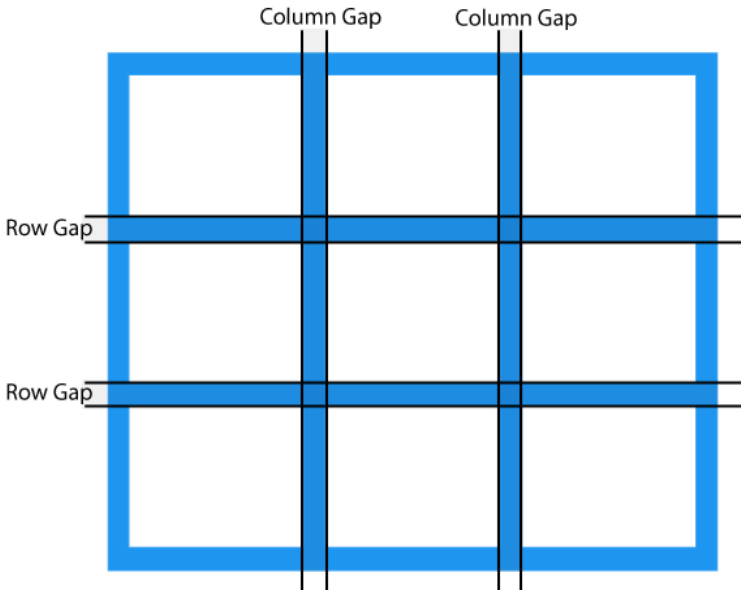
5. **Grid Rows:**

The horizontal lines of grid items are called rows.



6. **Grid Gaps (Gutters):**

The spaces between each column/row are called gaps.



You can adjust the gap size by using one of the following properties:

- column-gap
- row-gap
- gap

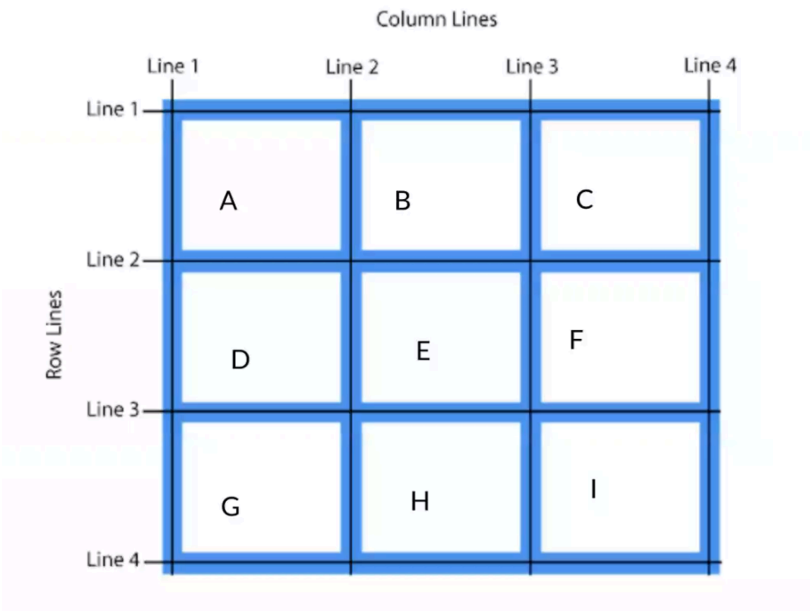
Example:

```
.grid-container {
    display: grid;
    /* for column gap */
    /* column-gap: 50px; */

    /* for row gap */
    /* row-gap: 50px; */

    /* Shorthand property for row-gap and column gap */
    gap: 50px 100px;
```

```
    /* Shorthand property for both row-gap and column gap */
    gap: 20px;

}
```

7. **Grid Lines:**

   The lines between columns are called **column lines** and the lines between rows are called **row lines**.



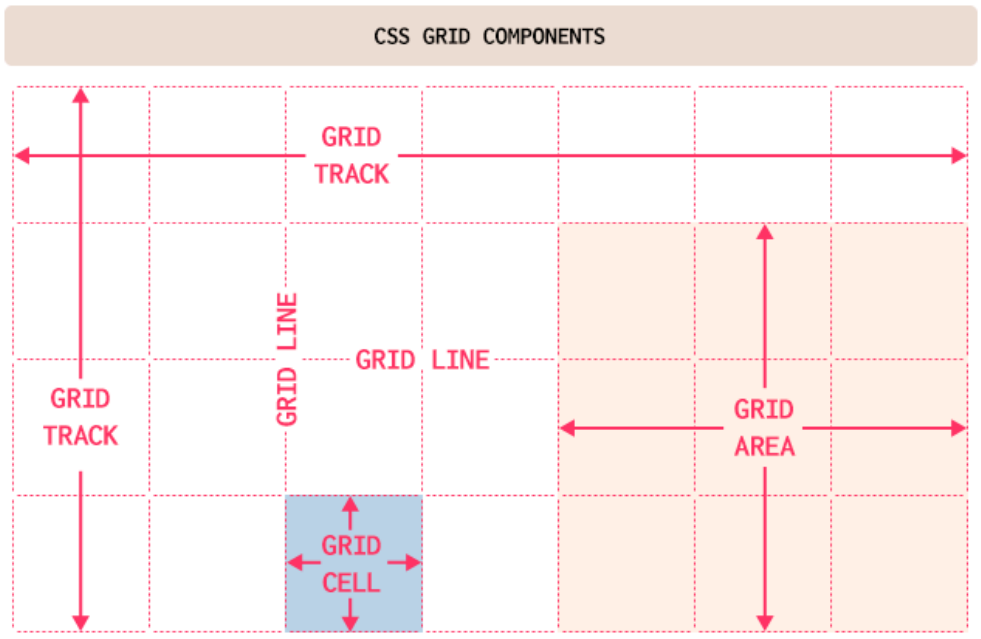
8. **Grid Tracks:**

   Grid tracks are defined by **grid-template-rows** and **grid-template-columns**, which indicate the spaces between adjacent lines on the grid.

   You can think of them as the columns or rows of the grid.

   Grid track is a generic term for a grid column or grid row.

   Note: Each grid track is assigned a sizing function, which controls how wide or tall the column or row may grow, and thus how far apart its bounding grid lines are.



The following diagram highlights the first-row track in the grid layout:

First Row		

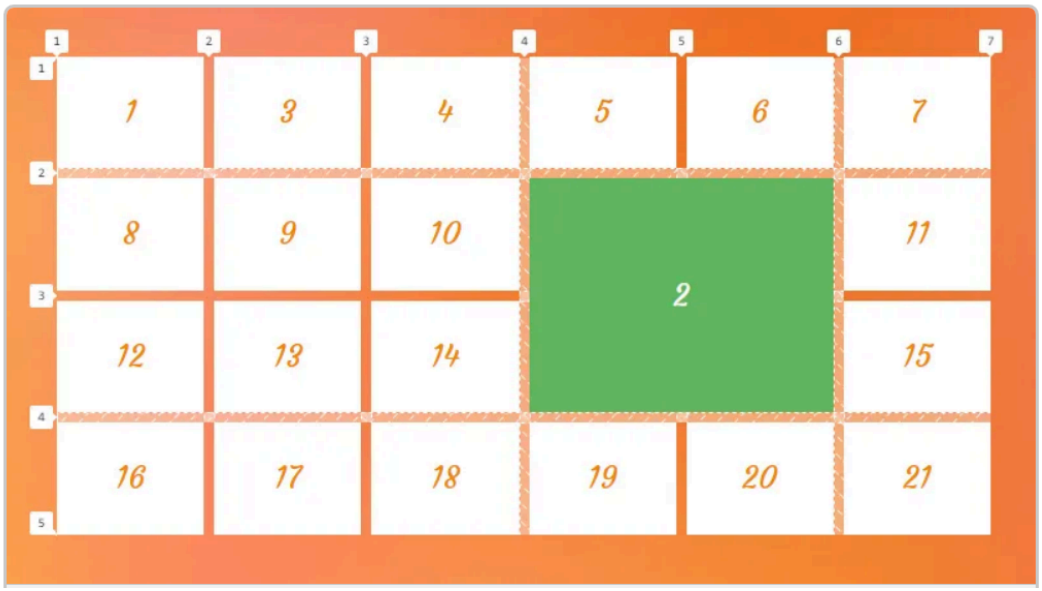
9. **Grid Area:**

The **grid-area** CSS property is a shorthand that specifies the location and the size of a **grid item** in a grid layout by setting the value of **grid-row-start**, **grid-column-start**, **grid-row-end** and **grid-column-end** in one declaration.

Example:

```
.items:nth-child(2) {
    grid-area: 2 / 4 / 4 / 6;
```

```
    /* is equivalent to: */
    grid-row-start: 2;
    grid-column-start: 4;
    grid-row-end: 4;
    grid-column-end: 6;
}
```



CSS Grid Properties:

1. **grid-template-columns** and **grid-template-rows:**

   Defines the columns and rows of the grid with a space-separated list of values. The values represent the **track size**, and the space between them represents the **grid line**.

Grid track using px:

```
.container {
        - -
        grid-template-columns: 300px 300px 300px;
}
```

The above grid property will create a container to have 3 columns with 300px each.

```
        grid-template-columns: 100px 100px auto ;
```

The above grid property will create a container to have 3 columns with 1st column 100px, 2nd column 100px and the 3rd column will takes the remaining spaces.

```
        grid-template-columns: 100px auto 200px;
```

The above grid property will create a container to have 3 columns with 1st column 100px, 3rd column 200px and the 2nd column will takes the remaining spaces.

Grid track using %:

```
  grid-template-columns: 25% 50% 25%;
```

The above grid property will create a container to have 3 columns with 1st column 25%, 2nd column 50% and the 3rd column will takes the 25%..

Grid track using fr:

fr-unit: **Fr is a fractional unit and 1fr is for 1 part of the available space.**

```
grid-template-columns: 1fr 1fr 1fr 1fr;
```

The above line will create grid layout with four columns, each having a width of 1fr.

Grid tracks unequal sizes:

```
grid-template-columns: 1fr 1fr 2fr 1fr
```

The above line will create grid layout with four unequal columns, where 1st, 2nd and 4th column takes up equal space (1fr) while the 3rd column takes up double space (2fr).

Grid tracks with flexible and absolute sizes:

```
grid-template-columns: 1fr 2fr 100px
```

The above line will create a grid layout with three unequal columns the first column takes 1fr of the available space, the second column takes 2fr, and the third column has a fixed width of 100px.

Grid Tracks With repeat():

The **repeat()** notation is useful for large grids with many tracks, to define a pattern of repeated track sizes within a grid.

The below code defines three columns, each with a size of 250px.

```
.container {
    display: grid;
    grid-template-columns: 250px 250px 250px;
}
```

The above code can alternatively be written as:

```
.container {
    display: grid;
```

```
            grid-template-columns: repeat(3, 250px);
        }
```

The repeat notation in Grid Layout allows you to create a repetitive pattern for a part of the track listing.

```
grid-template-columns: 80px repeat(4, 1fr) 80px;
```

The above property create a grid layout with six columns, the first column has a fixed width of 80px, the next four columns with a width of 1fr of the available space, and the last column is 80px wide

```
grid-template-columns: repeat(3, 100px 200px);
```

The above property create a grid layout with 3 pairs of columns(total 6 columns) where each pair consists of 1st column 100px and 2nd column 200px.

grid-template-rows:

Specifies the number (and the heights) of the rows in a grid layout.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .container {
            border: 2px solid green;
            background-color: antiquewhite;
```

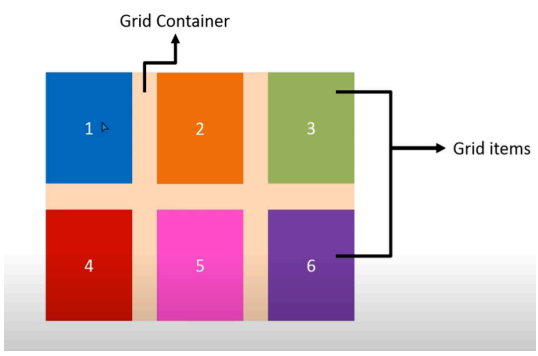
```
            padding: 10px;
            display: grid;
            grid-template-columns: repeat(3, 100px);

/* Here 1st row will takes 100px, 2nd row will takes up 200px and
remaining rows if any will be adjusted accordingly */
            grid-template-rows: 100px 200px;
       /*     grid-template-rows: auto; */
        }
        .items {
            background-color: aqua;
            border: 1px solid blue;
            font-size: 30px;
            text-align: center;
        }
    </style>
</head>
<body>
    <h2 align="center">Grid System</h2>
    <div class="container">
        <div class="items">Item1</div>
        <div class="items">Item2</div>
        <div class="items">Item3</div>
        <div class="items">Item4</div>
        <div class="items">Item5</div>
        <div class="items">Item6</div>
        <div class="items">Item7</div>
```

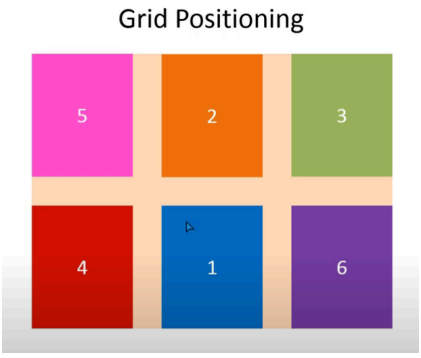
```
        <div class="items">Item8</div>
        <div class="items">Item9</div>
    </div>
</body>
</html>
```

2. **Grid items Positioning properties:**

   To put the grid-item in different location inside the grid-container, is known as grid positioning.

   Example:





   To position each grid item inside the container we need to make use of the following properties:

   - grid-row-start
   - grid-row-end
   - grid-row
   - grid-column-start
   - grid-column-end
   - grid-column

- grid-area

Note: To position any item in the different location we need to know about the grid line numbers.

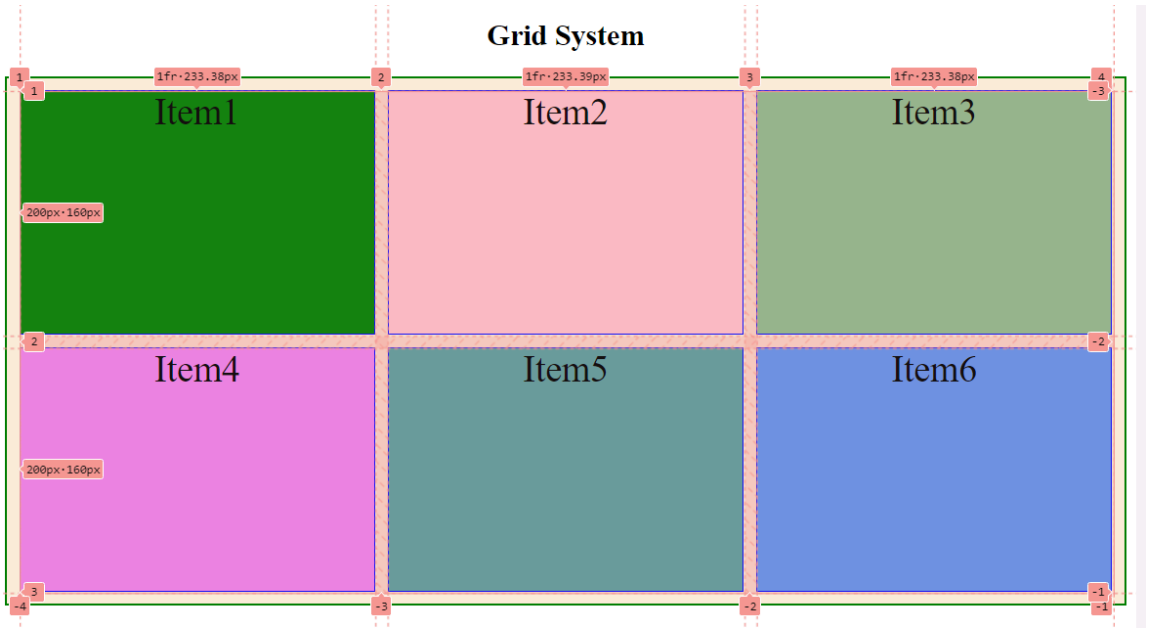
Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .container {
            border: 2px solid green;
            background-color: antiquewhite;
            padding: 10px;
            display: grid;
            grid-template-columns: repeat(3, 1fr);
            grid-template-rows: 200px 200px;
            gap: 10px;
        }
        .items {
            border: 1px solid blue;
            font-size: 30px;
            text-align: center;
```

```
        }
        .item1 {
            background-color: green;
        }
        .item2 {
            background-color: pink;
        }
        .item3 {
            background-color: darkseagreen;
        }
        .item4 {
            background-color: violet;
        }
        .item5 {
            background-color: cadetblue;
        }
        .item6 {
            background-color: cornflowerblue;
        }
    </style>
</head>
<body>
    <h2 align="center">Grid System</h2>

    <div class="container">
        <div class="items item1">Item1</div>
```

```
        <div class="items item2">Item2</div>
        <div class="items item3">Item3</div>
        <div class="items item4">Item4</div>
        <div class="items item5">Item5</div>
        <div class="items item6">Item6</div>
    </div>
</body>
</html>
```

Output: with grid-line numbers:



Putting the item1 to the last-row and last column in the above example:

```
.item1 {
            background-color: green;
            grid-row-start: 2;
```

```
        grid-row-end: 3;

        /* shorthand of the abpve both properties */
        /* grid-row: 2 / 3; */

        grid-column-start: 3;
        grid-column-end: 4;

        /* shorthand of the abpve both properties */
        /* grid-column: 3 / 4; */
    }
```

Putting the item6 to the first-row and first column in the above example:

```
    .item6 {
        background-color: cornflowerblue;
        grid-row-start: 1;
        grid-row-end: 2;
        grid-column-start: 1;
        grid-column-end: 2;
    }
```

Note: Even for the shorthand of **grid-row** and **grid-column** we can also make use of **grid-area** property

syntax:

grid-area: rowLineStart / colLineStart / rowLineEnd / colLineEnd

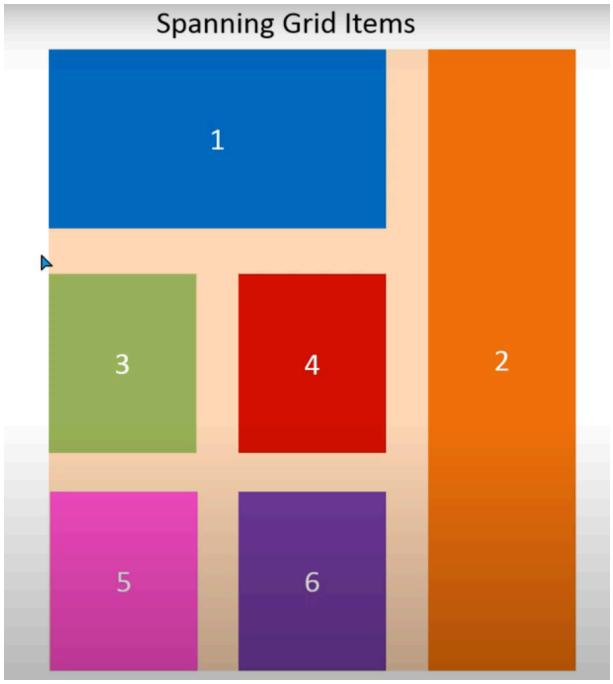
Example

```
.item1 {
    background-color: green;
    /* grid-row-start: 2;
    grid-row-end: 3; */

    /* shorthand of the abpve both properties */
    /* grid-row: 2 / 3; */

    /* grid-column-start: 3;
    grid-column-end: 4; */

    /* shorthand of the above both properties */
    /* grid-column: 3 / 4; */

    grid-area: 2 / 3 / 3 / 4;

}
```

3. **Spanning Grid Items:**

Spanning means we can span row or column to acquire multiple positions.

Example:



Here item1 span 2 column and item2 span 3 rows.

Note: Here also we need to make use of grid-lines, just we need to make use of starting line number and ending line number as above.

Example

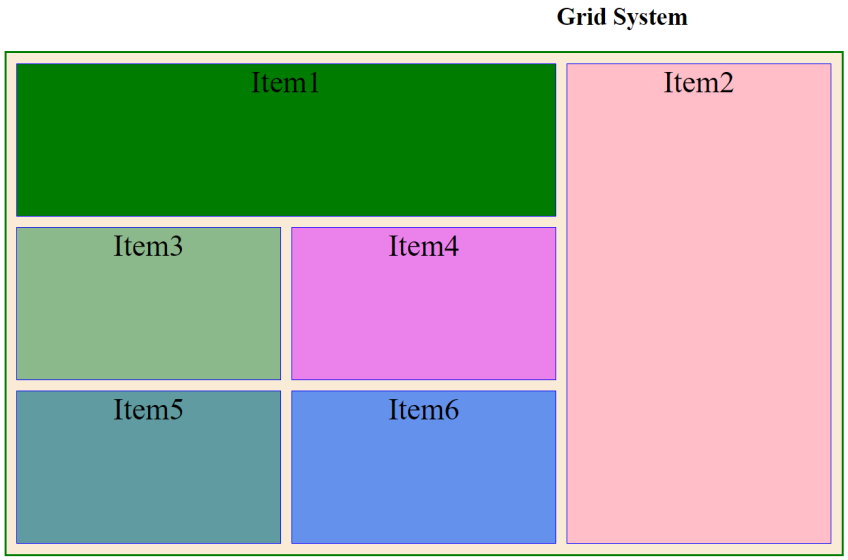
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .container {
            border: 2px solid green;
```

```
    background-color: antiquewhite;
    padding: 10px;
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-template-rows: 150px 150px 150px;
    gap: 10px;
    width: 800px;
}
.items {
    border: 1px solid blue;
    font-size: 30px;
    text-align: center;
}
.item1 {
    background-color: green;
    /* grid-row-start: 1;
    grid-row-end: 2;

    grid-column-start: 1;
    grid-column-end: 3; */

    /* grid-row: 1 / 2;
    grid-column: 1 / 3; */

    grid-area: 1 / 1 / 2 / 3;
```

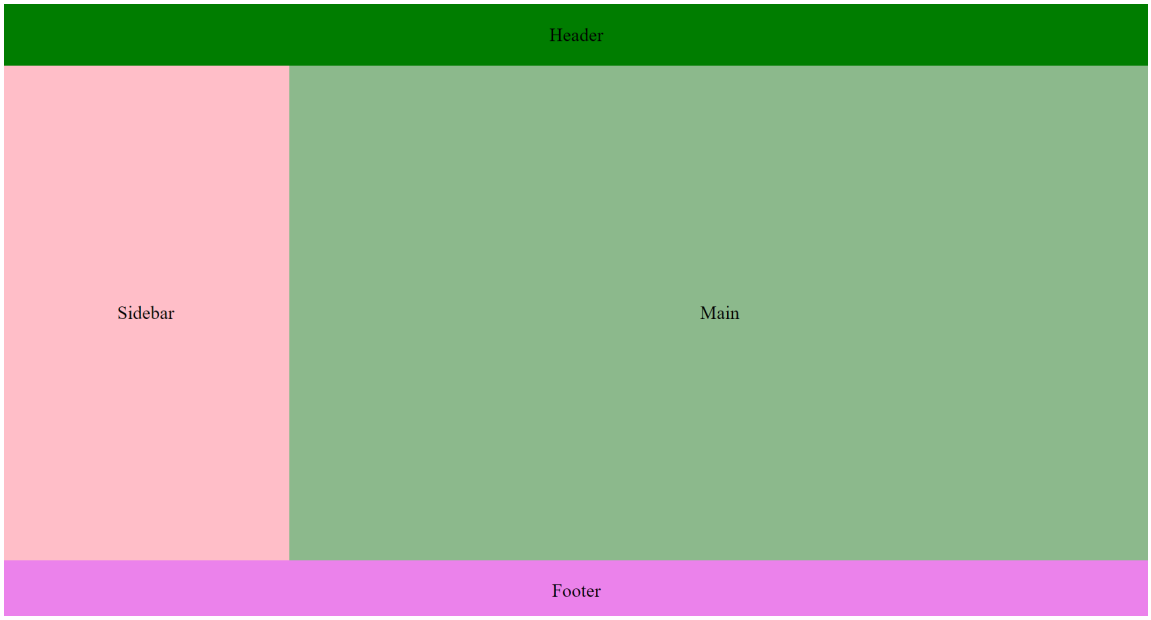
```
        }

        .item2 {
            background-color: pink;

            /* grid-row-start: 1;
            grid-row-end: 4;

            grid-column-start: 3;
            grid-column-end: 4; or we can use -1 means till end  */

            grid-row: 1 / 4;
            grid-column: 3 / 4;

            /* We can also make use of grid-row: 1 / span 3  mean start
from 1 and span the 3 location*/

            /* grid-row: 1 / span 3;
            grid-column: 3/ span 1; */

            /* grid-area: 1 / 3 / 4 / 4;             */
        }

        .item3 {
            background-color: darkseagreen;
        }
```

```
        .item4 {
            background-color: violet;
        }

        .item5 {
            background-color: cadetblue;
        }

        .item6 {
            background-color: cornflowerblue;

        }
    </style>
</head>
<body>
    <h2 align="center">Grid System</h2>
    <div class="container">
        <div class="items item1">Item1</div>
        <div class="items item2">Item2</div>
        <div class="items item3">Item3</div>
        <div class="items item4">Item4</div>
        <div class="items item5">Item5</div>
        <div class="items item6">Item6</div>
    </div>
</body>
</html>
```

Output:



Activity: Create the following layout using CSS grid



Solution:

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        body {
            margin: 0;
            padding: 0;
            height: 100vh;
            display: grid;

            /* Here due to the sidebar we took 4 columns */
            grid-template-columns: repeat(4, 1fr);

            grid-template-rows: 10vh 80vh 10vh;

        }

        .items {
            font-size: 25px;
            display: flex;
```

```
    justify-content: center;
    align-items: center;
}

.header {
    background-color: green;



    grid-row: 1 / 2;
    grid-column: 1 / 5;

    /* grid-row: 1 span 2;
    grid-column: 1 / span 4; */



}

.sidebar {
    grid-row: 2 / 3;
    grid-column: 1 / 2;
    background-color: pink;
}

.main {
    background-color: darkseagreen;
    grid-row: 2/ 3;
```

```
            grid-column: 2/ 5;
        }

        .footer {
            background-color: violet;
            grid-row: 3 / 4;
            grid-column: 1 / 5;
        }
    </style>
</head>

<body>

    <div class="items header">Header</div>
    <div class="items sidebar">Sidebar</div>
    <div class="items main">Main</div>
    <div class="items footer">Footer</div>

</body>

</html>
```

4. **<u>Naming the grid-lines:</u>**

By default grid-lines have the implicit numbers, but we can provide its names also for each grid-lines and then we can refer these names while spanning or positioning the grid-items.

Note: for a complex layout, working with grid numbers will become tedious, to overcome this we can name these lines and refer the line names.

We will give the names for the line numbers in **grid-template-rows** and **grid-template-columns.**

Example:

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        body {
            margin: 0;
            padding: 0;
            height: 100vh;
            display: grid;

            /* Here due to the sidebar we took 4 columns */
            grid-template-columns: repeat(4, 1fr);
```

```
            grid-template-rows: [header-row-start] 10vh [header-row-end
side-main-row-start] 80vh [side-main-row-end footer-row-start]
10vh[footer-row-end];


        }


        .items {
            font-size: 25px;
            display: flex;
            justify-content: center;
            align-items: center;
        }


        .header {
            background-color: green;



            grid-row: header-row-start / header-row-end;
            grid-column: 1 / 5;









        }


        .sidebar {
```

```
            grid-row: side-main-row-start / side-main-row-end;
            grid-column: 1 / 2;

            background-color: pink;
        }

        .main {
            background-color: darkseagreen;
            grid-row: side-main-row-start / side-main-row-end;
            grid-column: 2/ 5;

        }

        .footer {
            background-color: violet;
            grid-row: footer-row-start / footer-row-end;
            grid-column: 1 / 5;

        }
    </style>
</head>

<body>

    <div class="items header">Header</div>
    <div class="items sidebar">Sidebar</div>
```

```
    <div class="items main">Main</div>

    <div class="items footer">Footer</div>


</body>


</html>
```

Similarly we can provide the names of columns also.

5. **Naming Grid-area:**

Here we provide the name of a grid-area and then can position the grid-items.

Here we need not use **grid-column** and **grid-row** properties even no need to make use of grid-lines.

For this, we need to make use of following 2 properties:

- grid-template-areas
- grid-area

First using **grid-area** we need to provide a unique name for each grid item, then using the property **grid-template-area** we make position and span each grid items to different position.

Note: we need to specify the value of **grid-template-area** corresponding to the rows and columns otherwise the entire design will be disturbed.

Example:

```
<!DOCTYPE html>
```

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        body {
            margin: 0;
            padding: 0;
            height: 100vh;
            display: grid;

            /* Here due to the sidebar we took 4 columns */
            grid-template-columns: repeat(4, 1fr);

            grid-template-rows: 10vh 80vh 10vh;

            grid-template-areas:
                "header header header header"
                "side main main main"
                "footer footer footer footer";

        }

        .items {
```

```
    font-size: 25px;
    display: flex;
    justify-content: center;
    align-items: center;
}

.header {
    background-color: green;
    grid-area: header;
}

.sidebar {
    background-color: pink;
    grid-area: side;
}

.main {
    background-color: darkseagreen;
    grid-area: main;
}

.footer {
    background-color: violet;
    grid-area: footer;

}
```

```
    </style>
</head>

<body>

    <div class="items header">Header</div>
    <div class="items sidebar">Sidebar</div>
    <div class="items main">Main</div>
    <div class="items footer">Footer</div>

</body>

</html>
```

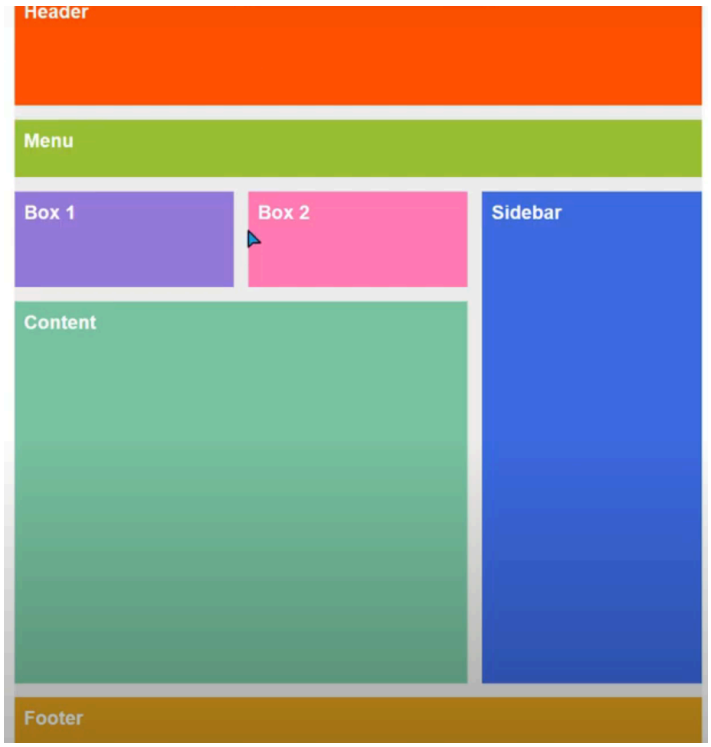
If we want to make a gap of a specific column then we can make use of .(dot)

Example:

```
grid-template-areas:
            " . header header . "
            "side main main main"
            "footer footer footer footer" ;
```

Here the header part will comes to the middle.

Student Activity: create the following layout using **grid-template-areas** property.



1. **Implicit and explicit grid:**

   When we use `grid-template-columns` and `grid-template-rows`, we define an **Explicit Grid**. However, if we attempt to place an item outside of this explicit grid (e.g., by specifying a column or row that doesn't exist in the explicit grid), the browser will create an **Implicit Grid** by adding implicit grid tracks (rows or columns) to accommodate the item.

   Example:

```
<!DOCTYPE html>

<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
           .container {
            border: 2px solid green;
            background-color: antiquewhite;
            padding: 5px;
            display: grid;



            grid-template-columns: 1fr 1fr;
            grid-template-rows: repeat(2, 100px);
            gap: 5px;
        }
        .items {
            border: 1px solid blue;
        }
        .item1 {
            background-color: green;
        }
        .item2 {
            background-color: pink;
        }
        .item3 {
```

```
            background-color: darkseagreen;
        }
        .item4 {
            background-color: violet;
        }
        .item5 {
            background-color: darkred;
        }
        .item6 {
            background-color: teal;
        }
        .item7 {
            background-color: salmon;
        }
        .item8 {
            background-color: navy;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="items item1">Item1</div>
        <div class="items item2">Item2</div>
        <div class="items item3">Item3</div>
        <div class="items item4">Item4</div>
        <div class="items item5">Item5</div>
```

```
        <div class="items item6">Item6</div>
        <div class="items item7">Item7</div>
        <div class="items item8">Item8</div>
    </div>
</body>
</html>
```

Output:

Here Item1 to item4 will comes under the explicit grid where as item5 to item8 will called as implicit grid.

here for the implicit grid the line numbers will be generated automatically.

To style the implicit grid we need to make use of the following properties:

- grid-auto-rows
- grid-auto-columns
- grid-auto-flows

grid-auto-rows: Using this property we can provide the height for the implicit rows.

Example:

```
.container {
    --
    grid-auto-rows: 50px;
}
```

grid-auto-flows: using the **grid-auto-rows,** if we add some extra divs it will comes as a new row with the specified 50px height. but if we want that new divs comes as a column then we need to make use of **grid-auto-flows.**

Example:

```
/* row is the default value */
grid-auto-flow: column;
```

grid-auto-columns: Using this property we can specify the width of the implicit columns

Example:

```
grid-auto-flow: column;
grid-auto-columns: 150px;
/* grid-auto-columns: 1fr; */
```

1. **Grid item alignment:**

Using this we can align each grid item horizontally or vertically inside the grid container.

For this we need to make use of the following properties:

- justify-items
- align-items
- place-items
- justify-self
- align-self
- place-self

justify-items: It is used to align all the grid items **horizontally between their grid line**.

here also we have total 4 values are there:

- start
- end
- center

- stretch (it is the default value)

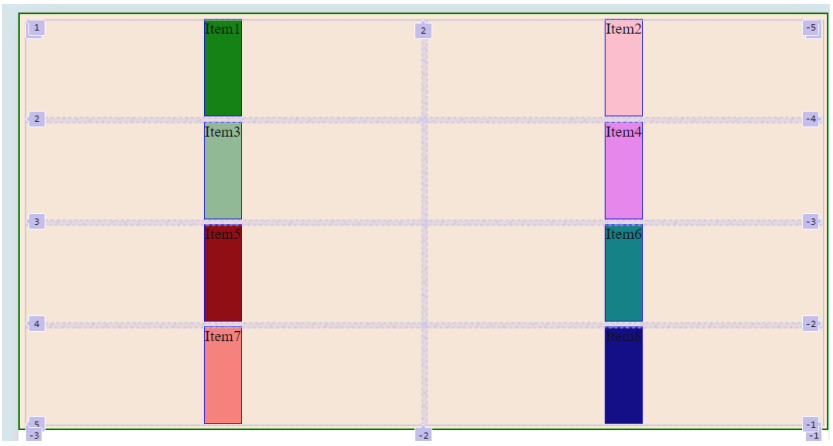
Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .container {
            border: 2px solid green;
            background-color: antiquewhite;
            padding: 5px;
            display: grid;

            grid-template-columns: 1fr 1fr;
            grid-template-rows: repeat(4, 100px);
            gap: 5px;

            justify-items: center;
        }

        .items {
            border: 1px solid blue;
        }
```

```css
.item1 {
    background-color: green;
}
.item2 {
    background-color: pink;
}

.item3 {
    background-color: darkseagreen;
}

.item4 {
    background-color: violet;
}
.item5 {
    background-color: darkred;
}
.item6 {
    background-color: teal;
}
.item7 {
    background-color: salmon;
}
.item8 {
    background-color: navy;
}
```

```
    </style>
</head>

<body>
    <div class="container">
        <div class="items item1">Item1</div>
        <div class="items item2">Item2</div>
        <div class="items item3">Item3</div>
        <div class="items item4">Item4</div>
        <div class="items item5">Item5</div>
        <div class="items item6">Item6</div>
        <div class="items item7">Item7</div>
        <div class="items item8">Item8</div>
    </div>
</body>
</html>
```

Output:



justify-self: This property is used inside the individual grid items to align them horizontally between their 2 grid lines. It will override the **justify-items** property of the container.

Example:

```
.item3 {
            background-color: darkseagreen;
            justify-self: end;
      }
```

align-items: It is used to align all the grid items **vertically between their grid line**.

here we have total 4 values are there:

- start
- end
- center
- stretch (it is the default value)

Example:

```
.container {
    - - -
    align-items: center;
}
```

here each items will be placed in the center vertically between their 2 lines.

Example:

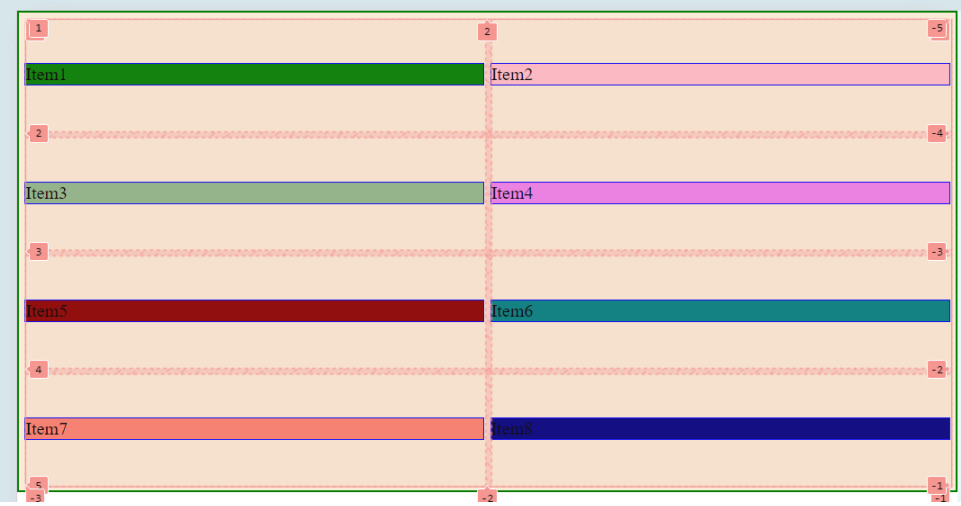
```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
           .container {
            border: 2px solid green;
            background-color: antiquewhite;
            padding: 5px;
            display: grid;

            grid-template-columns: 1fr 1fr;
            grid-template-rows: repeat(4, 100px);
            gap: 5px;

           align-items: center;



 /* here all the items will be places at the end of their grid lines vertically
*/
            /* align-items: end; */

/* here all the items will be places at the start of their grid lines vertically
*/
            /* align-items: start; */
        }
```

```css
.items {
    border: 1px solid blue;
}
.item1 {
    background-color: green;
}
.item2 {
    background-color: pink;
}
.item3 {
    background-color: darkseagreen;
}
.item4 {
    background-color: violet;
}
.item5 {
    background-color: darkred;
}
.item6 {
    background-color: teal;
}
.item7 {
    background-color: salmon;
}
.item8 {
    background-color: navy;
```

```
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="items item1">Item1</div>
        <div class="items item2">Item2</div>
        <div class="items item3">Item3</div>
        <div class="items item4">Item4</div>
        <div class="items item5">Item5</div>
        <div class="items item6">Item6</div>
        <div class="items item7">Item7</div>
        <div class="items item8">Item8</div>
    </div>
</body>
</html>
```

Output:



align-self: This property is used inside the individual grid items to align them vertically between their 2 grid lines. It will override the **align-items** property of the container.

Example:

```
.item3 {
            background-color: darkseagreen;
            align-self: end;
        }
```

place-items / place-self:

These are the shorthand of <**align-items> <justify-items>** and the **<align-self> <justify-self>**

place-items: <**align-items> <justify-items>**

place-self: **<align-self> <justify-self>**

Example:

```
.container {

    place-items: center start;

}
```

Similarly we can use the **place-self** inside each individual item.

Grid Track alignment:

Using this we can provide the alignment of all the items together inside the container vertically and horizontally inside the grid container.

For this we need to make use of the following properties inside the container.

- align-content (for vertical alignment)
- justify-content (for horizontal alignment)
- place-content (shorthand of above two)

For the **align-content** and the **justify-content** we can make use of the following properties:

- start
- end
- center
- stretch
- space-around
- space-between
- space-evenly

Example:

```
<!DOCTYPE html>

<html lang="en">


<head>

    <meta charset="UTF-8">
```

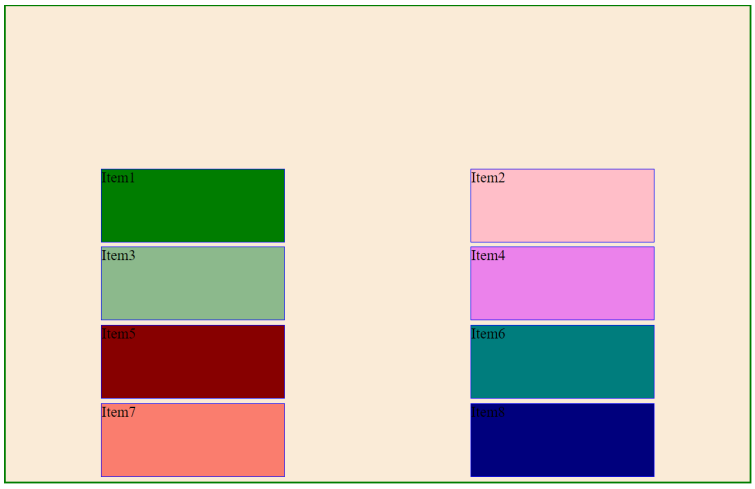
```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<style>
    .container {
        border: 2px solid green;
        background-color: antiquewhite;
        padding: 5px;
        display: grid;

        /* To see the effect of align-content */
        height: 80vh;

        grid-template-columns: 200px 200px;
        grid-template-rows: repeat(4, 80px);
        gap: 5px;

        align-content: end;
        justify-content: space-between;

        /* shorthand for the above properties */
        /* place-content: end space-between; */

    }


    .items {
```

```
    border: 1px solid blue;
}



.item1 {
    background-color: green;
}



.item2 {
    background-color: pink;
}



.item3 {
    background-color: darkseagreen;
}



.item4 {
    background-color: violet;
}



.item5 {
    background-color: darkred;
```

```
        }



        .item6 {
            background-color: teal;
        }



        .item7 {
            background-color: salmon;
        }



        .item8 {
            background-color: navy;
        }
    </style>
</head>

<body>
    <div class="container">
        <div class="items item1">Item1</div>
        <div class="items item2">Item2</div>
        <div class="items item3">Item3</div>
        <div class="items item4">Item4</div>
        <div class="items item5">Item5</div>
```

```
        <div class="items item6">Item6</div>
        <div class="items item7">Item7</div>
        <div class="items item8">Item8</div>
    </div>
</body>


</html>
```

Output:



12. Grid item positioning using order property:

To position each grid-item in different location we have already used the following properties:

- grid-row-start
- grid-row-end
- grid-column-start
- grid-column-end

There is a another way also, using which we can position each grid-items in a different location. by using the **order** property, similar to the **order** property in flex-box.

Each grid-item has the default order value 0.

maximum order value grid-item will be placed at the last.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .container {

            border: 2px solid green;
            background-color: antiquewhite;
            padding: 5px;
            display: grid;

            grid-template-columns: 2fr 2fr 2fr 2fr;
            grid-template-rows: repeat(2, 100px);
            gap: 5px;
        }

        .items {
```

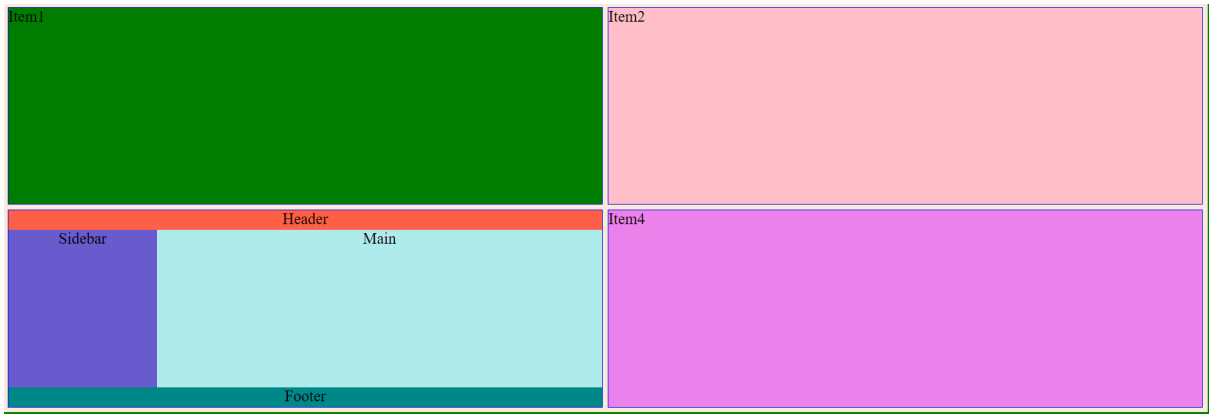
```
    border: 1px solid blue;
}

.item1 {
    background-color: green;
    order: 2;
}

.item2 {
    background-color: pink;
}

.item3 {
    background-color: darkseagreen;
}

.item4 {
    background-color: violet;
    order: -1;
}

.item5 {
    background-color: darkred;
}

.item6 {
```

```
            background-color: teal;
            order: 1;
        }

        .item7 {
            background-color: salmon;
        }

        .item8 {
            background-color: navy;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="items item1">Item1</div>
        <div class="items item2">Item2</div>
        <div class="items item3">Item3</div>
        <div class="items item4">Item4</div>
        <div class="items item5">Item5</div>
        <div class="items item6">Item6</div>
        <div class="items item7">Item7</div>
        <div class="items item8">Item8</div>
    </div>
</body>
</html>
```

14. Nested Grids:

As we can nest flex inside another flex, we can nest a grid inside another grid, i.e we can make a grid item as a grid container also. and style their inner items.

Student Activity: Create the following layout using nested grid:



CSS Grid v/s flexbox

Use CSS flexbox when-

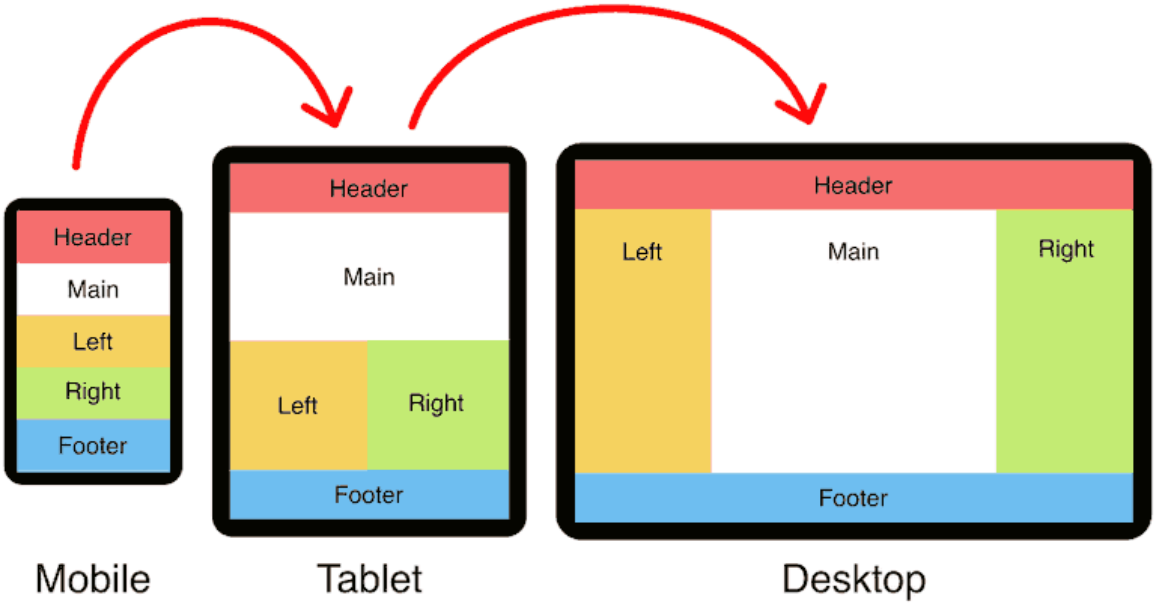
1. <u>You have a small design to implement:</u> Flexbox is ideal when you have a small layout design to implement, with a few rows or a few columns
2. <u>You need to align elements:</u> Flexbox is perfect for that, the only thing we should do is create a flex container using display: flex and then define the flex-direction that we want.
3. <u>You need a content-first design:</u> Flexbox is the ideal layout system to create web pages if you don't know exactly how your content is going to look, so if you want everything just to fit in, Flexbox is perfect for that.

Use CSS grid when

1. <u>You have a complex design to implement:</u> In some use cases, we have complex designs to implement, and that's when the magic of CSS Grid shows itself. The two-dimensional layout system here is perfect to create a complex design. We can use it in our favor to create more complex and maintainable web pages
2. <u>You need to have a gap between block elements:</u> Another thing that's very helpful in CSS Grid, that we don't have in Flexbox, is the gap property. We can define the gap between our rows or columns very easily, without having to use the margin property, which can cause some side effects especially if we're working with many breakpoints

3. You need a layout-first design: When you already have your layout design structure, it's easier to build with CSS Grid, and the two-dimensional layout system helps us a lot when we're able to use rows and columns together, and position the elements the way we want

Student Activity: Design the following layouts across different screen using Grid layout.



Solution:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
```

```
<style>
    body {
        margin: 0px;
        padding: 0px;
    }
    .container {
        display: grid;
        grid-template-columns: repeat(4, 1fr);
        grid-template-rows: 10vh 80vh 10vh;
        grid-template-areas:
            "header header header header"
            "left main main right"
            "footer footer footer footer";
    }
    .box {
        text-align: center;
        font-size: 1.5em;
        padding: 1%;
    }
    .header {
        background-color: indianred;
        grid-area: header;
    }
```

```css
.main {
    background-color: white;
    grid-area: main;
}

.left {
    background-color: yellow;
    grid-area: left;
}
.right {
    background-color: greenyellow;
    grid-area: right;
}

.footer {
    background-color: skyblue;
    grid-area: footer;
}

/* For Tablet */
@media screen and (min-width: 481px) and (max-width: 1024px) {


    .container {
        grid-template-columns: repeat(2, 1fr);
        grid-template-rows: 10vh 45vh 35vh 10vh;
```

```
        grid-template-areas:
            "header header"
            "main main"
            "left right"
            "footer footer";
    }
}
/* For Mobile */
@media screen and (max-width: 480px) {




    .container {
        grid-template-columns: 1fr;
        grid-template-rows: 20vh 20vh 20vh 20vh 20vh;

        grid-template-areas:
            "header"
            "main"
            "left"
            "right"
            "footer";
    }
}
```

```
    </style>
</head>
<body>
    <div class="container">
        <div class="box header">Header</div>
        <div class="box left">Left</div>
        <div class="box main">Main</div>
        <div class="box right">Right</div>
        <div class="box footer">Footer</div>
    </div>
</body>
</html>
```