overflow-wrap:

The CSS property overflow-wrap is used to control how words are broken within an element when they exceed the bounds of their container. This property ensures that the content doesn't overflow the container by allowing words to break and wrap onto the next line if necessary.

Syntax:

css

```
element {
  overflow-wrap: normal | break-word | anywhere;
}
```

Values:

normal (default):

Words will only break at normal word break points (such as spaces or hyphens). If the word is too long to fit within the container, it will overflow.

break-word:

This value allows for the breaking of long words to prevent overflow. Words will break at arbitrary points if needed to ensure they stay within the container.

anywhere:

This forces a break at any point in the text, even in the middle of words, to prevent overflow.

Example:

css

```
p {
  width: 200px;
  overflow-wrap: break-word;
}
```

Use Case:

The overflow-wrap property is particularly useful when working with responsive designs where text content may need to adapt to varying container sizes, or when displaying long, unbroken strings like URLs or emails.

The difference between anywhere and break-word in the overflow-wrap CSS property can be subtle, but here's a breakdown of how they behave:

break-word:

Behavior: It breaks long words only if they exceed the width of the container, but only at word boundaries (no breaking of words unless necessary).

Word-breaking: Happens when the word is too long to fit inside the container, and it breaks at an arbitrary point in the word to prevent overflow.

Example: If a word like "supercalifragilisticexpialidocious" is too long, it will break to prevent overflow, but the break will only occur if the word exceeds the container's size.

anywhere:

Behavior: This value breaks words at any point (even within words), without waiting for a boundary like a space or hyphen. It allows breaking even if the word could still fit in the container but is too long to handle gracefully.

Word-breaking: More aggressive, breaks text anywhere in the content, whether necessary or not, even mid-word.

Example: "supercalifragilisticexpialidocious" could break at any point, even if part of the word could still fit within the container, breaking it into multiple lines if needed.

Key Difference:

break-word only breaks long words when necessary to prevent overflow, and it tries to avoid breaking words unless absolutely required.

anywhere is more flexible and will break words at any character, even if it's not strictly necessary to avoid overflow.

Visual Example:

css

```
p.break-word {
  width: 100px;
  overflow-wrap: break-word;
```

}

p.anywhere {

  width: 100px;

  overflow-wrap: anywhere;

}

For a long word:

break-word: Only breaks when the word exceeds the container.

anywhere: Can break even before reaching the container's edge, breaking it into multiple smaller pieces.

Example:

HTML:

html

```html
<p class="break-word">supercalifragilisticexpialidocious</p>

<p class="anywhere">supercalifragilisticexpialidocious</p>
```

CSS:

css

```css
p {

  width: 100px;

  border: 1px solid black;

}

.break-word {

  overflow-wrap: break-word;

}
```

```
.anywhere {
  overflow-wrap: anywhere;
}
```

Behavior:

- For `.break-word`:
  - If the word "supercalifragilisticexpialidocious" exceeds the 100px width, it will break only if it's necessary to prevent overflow.
  - The browser will attempt to fit as much of the word as possible before breaking it. Once it can no longer fit, it breaks the word in a reasonable place (mid-word if necessary).
- For `.anywhere`:
  - The word "supercalifragilisticexpialidocious" can break at any character, regardless of whether it is at a boundary or not. This means the browser is more aggressive in wrapping, potentially breaking the word even before it's needed, so parts of the word may appear on different lines.

Results:

`break-word` might break the word like this (only after the container is full):

Copy code

```
supercalifragilisticexpialido-

cious
```

-

`anywhere` could break the word like this (breaking anywhere, even if not strictly necessary):

Copy code

```
supercalifragi-

listicexpialidoc-

ious
```

-

Key takeaway:

- `break-word`: Tries to fit as much of the word as possible before breaking it.
- `anywhere`: Breaks the word more aggressively, even at places where it could have still fit more characters in the container.

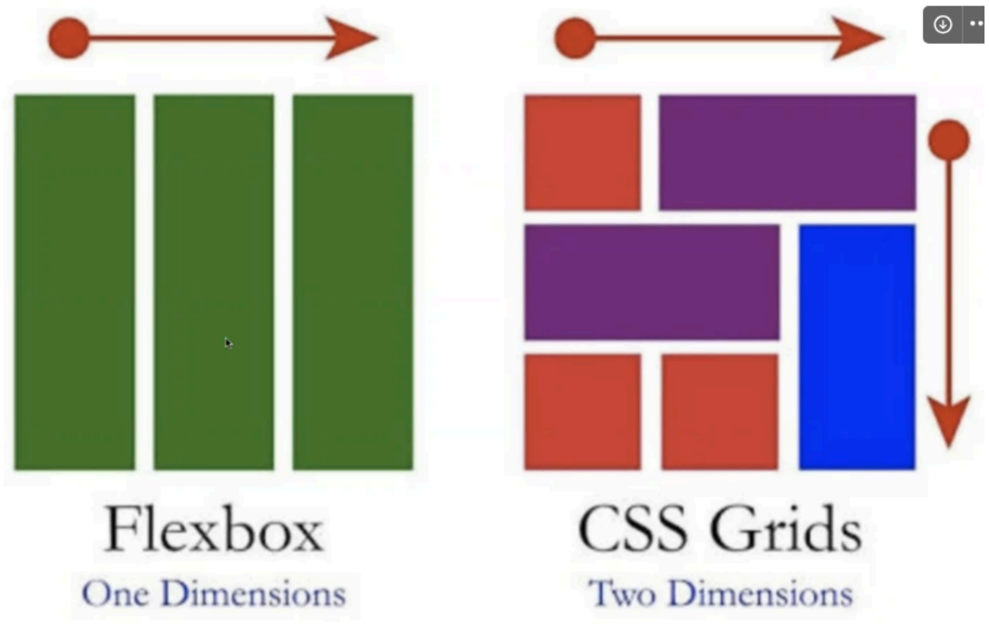
CSS Grid:

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

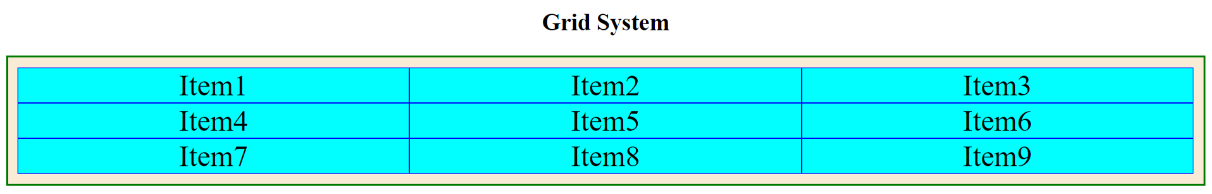
To get started you have to define a container element as a grid with **display: grid** set the column and row sizes with **grid-template-columns** and **grid-template-rows**.

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

Note: if we use **display: inline-grid** the container div will become the inline element.

Important CSS grid terminologies:

1. **Grid Container:**

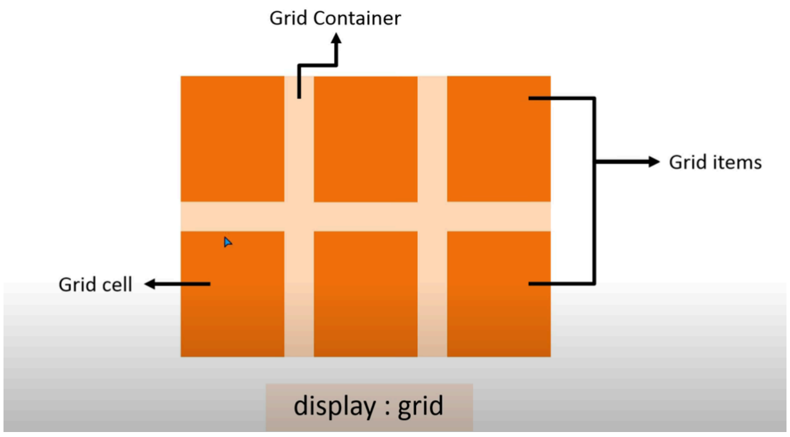
   The element on which display: grid is applied. It’s the direct parent of all the grid items.

2. **Grid item:**

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

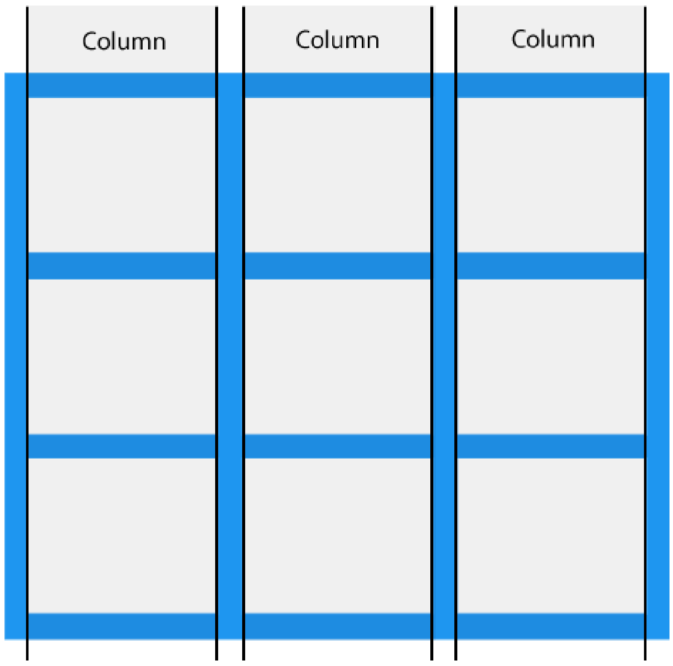
3. **Grid cell:**

   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.



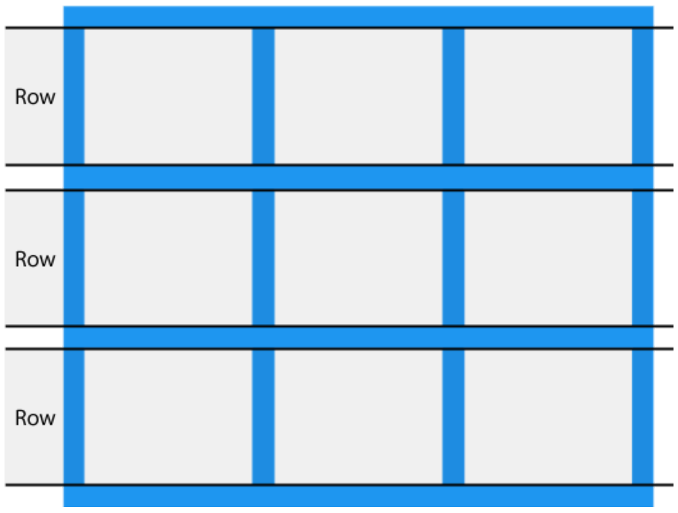
4. Grid Columns:

The vertical lines of grid items are called columns.



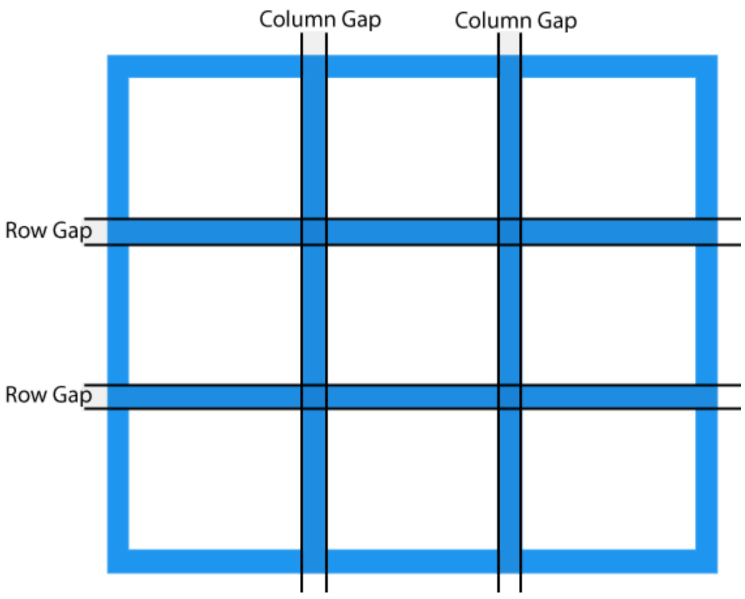
5. Grid Rows:

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:

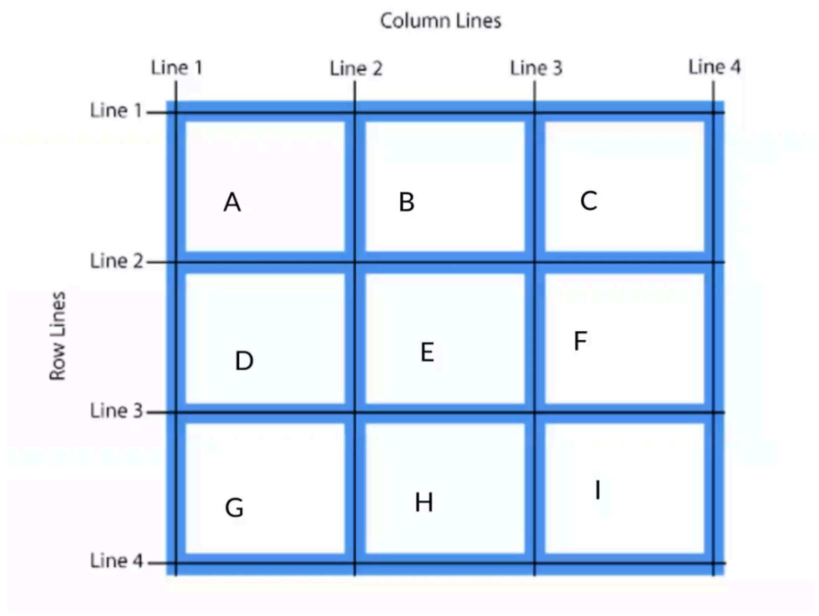
- `column-gap`
- `row-gap`
- `gap`

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;

    }
```

7. Grid Lines:

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

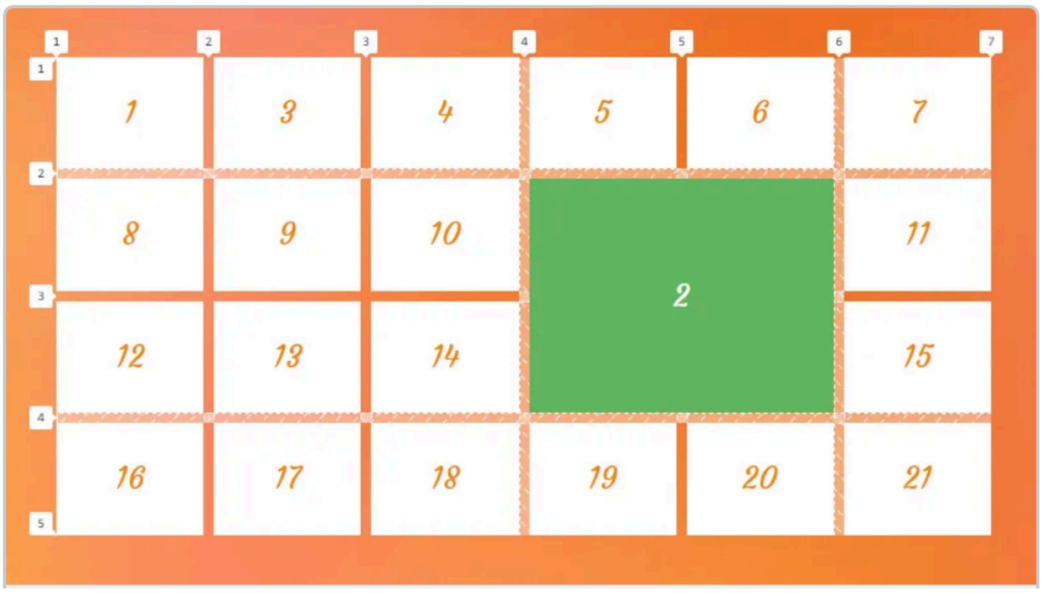


8. Grid Area:

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:

```css
.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. gap:

The **gap** CSS shorthand property sets the gaps (gutters) between rows and columns.

Early versions of the specification called this property **grid-gap**, and to maintain compatibility with legacy websites, browsers will still accept **grid-gap** as an alias for **gap**.

we can use **row-gap** (grid-row-gap) or **column-gap** (grid-column-gap) to specify the gaps individually for row and columns.

Example:

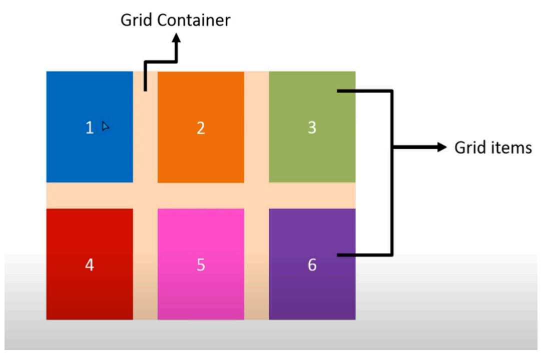
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
    <style>
        .container {

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

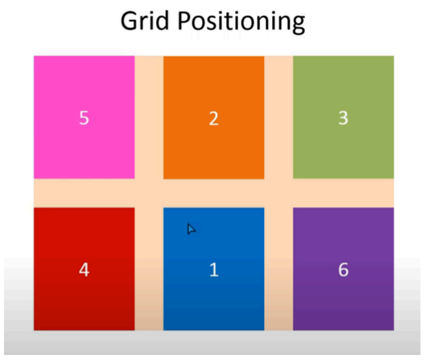
```
                gap: 10px;

             /* It is simillar to the row-gap: 10px and column-gap: 10px */

            }

            .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>
```

3. **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:

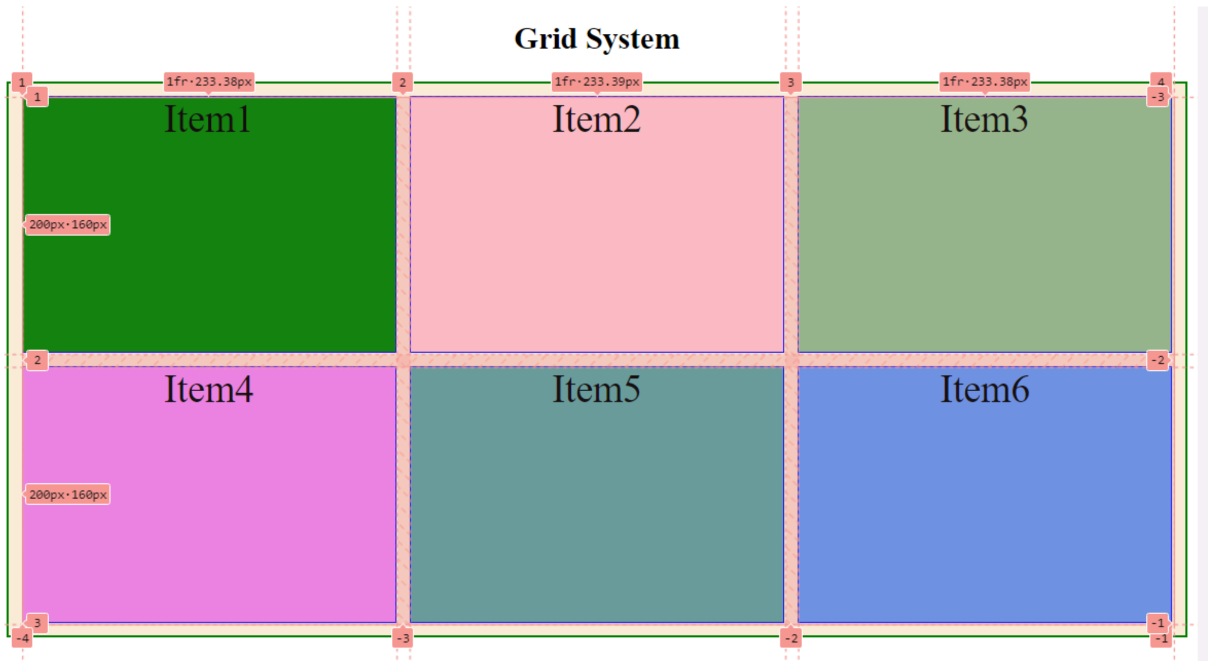
- grid-row-start
- grid-row-end
- grid-row
- grid-column-start
- grid-column-end
- grid-column
- grid-area

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

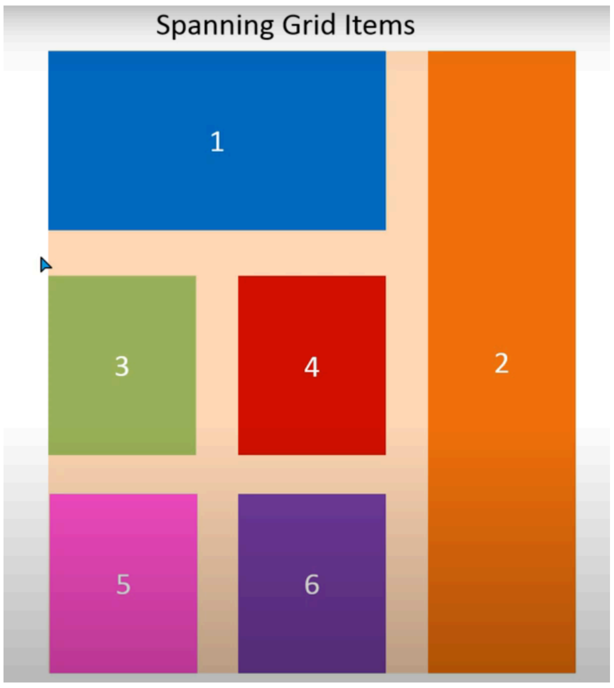
Example

```
              .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;


        }
```

4. **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.

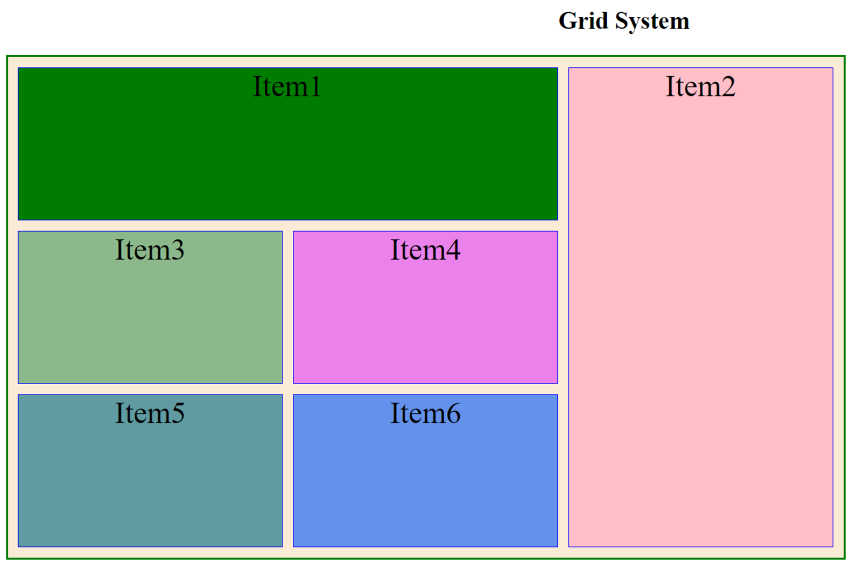
Note: Here also we need to make use of grid-lines, just we need to make use of starting line number and ending line number as above.

Example

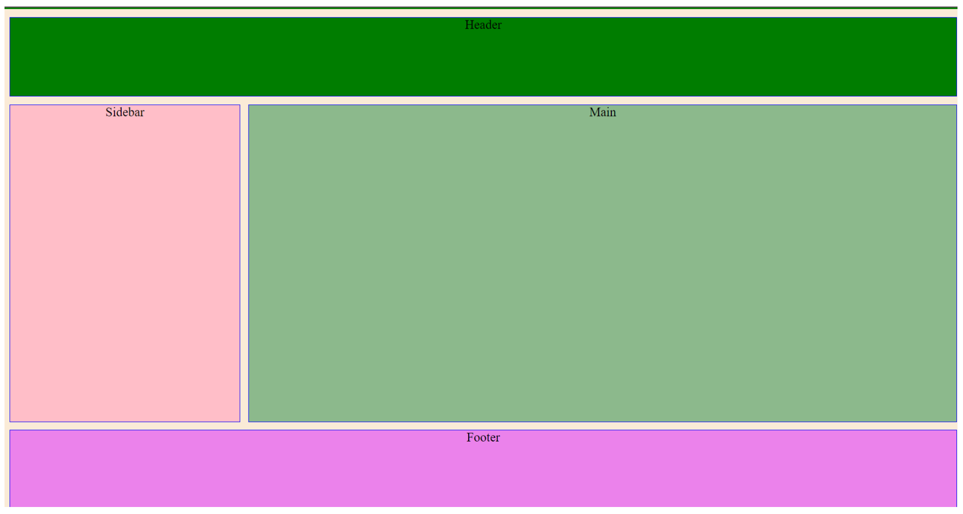
```
<!DOCTYPE html>

<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;
```

```css
            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



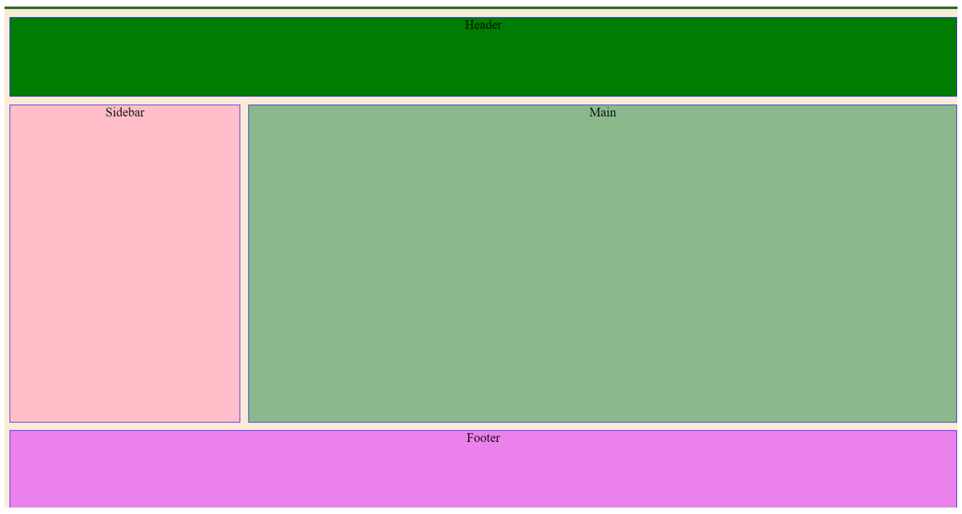
Solution:

```
<!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;

        }

        .container {

            border: 2px solid green;

            background-color: antiquewhite;

            padding: 10px;

            display: grid;


            /* Here due to the sidebar we took 4 columns */

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


            grid-template-rows: 100px 400px 100px;

            gap: 10px;

        }

        .items {

            border: 1px solid blue;

            text-align: 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>
```

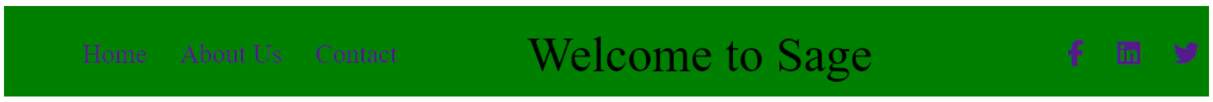
```
    <div class="container">
        <div class="items header">Header</div>
        <div class="items sidebar">Sidebar</div>
        <div class="items main">Main</div>
        <div class="items footer">Footer</div>
    </div>
</body>
</html>
```

Experiment:

Task 1: create the following layout using grid:



Task2: make the header item as a flex container and create the header layout as follows:



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. <u>You need a layout-first design:</u> 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

Responsive Web Design

To make a design responsive; we have to include following in out design

- Give dimensions in % or vw/vh

- Use em and rem unit to mention the font-size instead of px

- Ensure that the images take a percentage of the container width, adapting to different screen size. To so the same, apply following CSS to image
```
max-width: 100%;
height: auto;
```
-
- To make the background images responsive i.e. background image should adjust according to container size; apply following CSS
```
background-image: url('path to image')
background-size: cover;
background-position: center;
```

Also; use different images for different screen size.

- Implement a grid based layout that adjusts based on the screen size.
  `display: grid;`
  `grid-template-columns: repeat(auto-fill, minmax(200px, 1fr))`
  The grid-template-columns: repeat(auto-fill, minmax(200px, 1 1fr)) means every element will occupy the 200px minimum, if additional width; say we have 850px total width of container then 4 boxes will be in a single row and the additional 50px will also be distributed equally. The maximum width is 1 fr.

- Define responsive grid layout using named template areas
  `display: grid;`
  `grid-template-areas: ‘header header’ ‘main sidebar’ ‘footer footer’.`

- Use CSS media query to adjust the layout according to various screen size.

Example 1: adjust the screen to see the effect.

```
<!DOCTYPE html>
<html>

<head>
    <title>% v/s vw</title>
    <style>
        * {
            box-sizing: border-box;
        }

        .parent {
            width: 800px;
            border: 3px solid #CCFF00;
        }

        .child_per {
            width: 50%;
            border: 3px dashed #00CCFF;
        }

        .child_vw {
            width: 20vw;
            border: 3px double #CC00FF;
        }
    </style>
</head>

<body>
    <div class='parent'>

        <h3>The green bordered div is of 800px width; resize the screen
for best result</h3>



        <div class='child_per'>
            <strong>The sky bordered div is 50% i.e. 400px of width no
matter what is the screen size</strong><br />
            Lorem Ipsum is simply dummy text of the printing and
typesetting industry. Lorem Ipsum has been the
            industry's standard dummy text ever since the 1500s, when
an unknown printer took a galley of type and
            scrambled it to make a type specimen book.
        </div>

        <br><br>

        <div class='child_vw'>
            <strong>The sky bordered div is 20vw i.e. 20% of total
width of screen</strong>
            Lorem Ipsum is simply dummy text of the printing and
typesetting industry. Lorem Ipsum has been the
            industry's standard dummy text ever since the 1500s, when
an unknown printer took a galley of type and
            scrambled it to make a type specimen book.
        </div>
    </div>
</body>

</html>
```

Example2: creating a 200px*200px div to the center of a webpage

```
<!DOCTYPE html>

<html lang="en">


<head>

    <meta charset="UTF-8">

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

    <title>Centered Div</title>

    <style>

        body {

            margin: 0;

            padding: 0;

            /* Set body to full height */

            height: 100vh;


            /* Positioning context for absolute positioning */

            position: relative;

        }


        .centered {

            width: 200px;

            height: 200px;
            background-color: blue;
            /* Position the div in the center */
            position: absolute;
            /* Center horizontally */
            left: 50%;
            margin-left: -100px;
            /* Half of the width */
            /* Center vertically */
            top: 50%;
            margin-top: -100px;
            /* Half of the height */
        }
    </style>
</head>

<body>
    <div class="centered"></div>
</body>
</html>
```

Example3: Using flex:

```
<!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;

        }
        .container {
            background-color: aquamarine;
            width: 100vw;
            height: 100vh;
            display: flex;
        }
        .box {
            background-color: blueviolet;
            width: 200px;
            height: 200px;
            margin: auto;
        }
    </style>
</head>

<body>
    <div class="container">
        <div class="box"></div>
    </div>
</body>
</html>
```

Example4: Using Grid:

```
<!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;

        }

        .container {

            background-color: aquamarine;

            width: 100vw;

            height: 100vh;

            display: grid;


            grid-template-columns: auto;

            grid-template-rows: auto;


            justify-items: center;

            align-items: center;

            /* place-items: center center; */

        }

        .box {
            background-color: blueviolet;

            width: 200px;

            height: 200px;

        }

    </style>

</head>


<body>


    <div class="container">

        <div class="box"></div>

    </div>

</body>

</html>
```

Note: Here also we can make use of **margin: auto** inside the **box div**, then no need to take: **grid-template-row, grid-template-column, justify-items, align-items** properties.

CSS Media Queries:

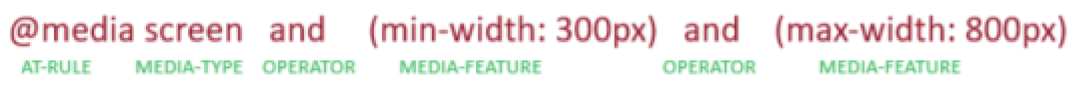
CSS Media Queries are a way for web developers to apply different styles to a webpage based on various factors like the size of the screen, the device being used, or even the orientation of the device.

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



```
@media screen and (min-width: 300px) and (max-width: 800px)
```

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. You can also comma-separate features as a way of using an or operator to match different ones. 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.

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 */

}
```

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 style="text-align: center;">Welcome to Sage</h1>
</body>
```

```
</html>
```

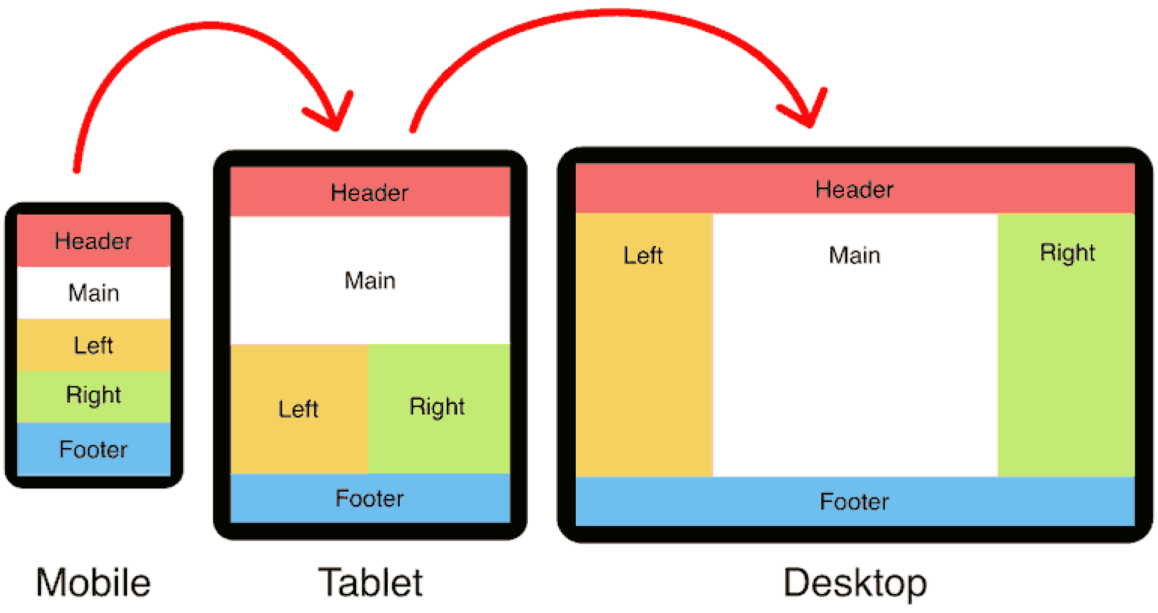
Example2:

```
<!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 style="text-align: center;">Welcome to Sage1</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: 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;
    }
```

```
.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 {
        display: grid;

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

        grid-template-rows: 10vh 45vh 35vh 10vh;



        grid-template-areas:

            "header header header header"

            "main main main main"

            "left left right right"

            "footer footer footer footer";

    }

}

/* For Mobile */

@media screen and (max-width: 480px) {



    .container {

        display: grid;

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

        grid-template-rows: 20vh 20vh 20vh 20vh 20vh;


        grid-template-areas:

            "header header header header"

            "main main main main"

            "left left left left"

            "right right right right"

            "footer footer footer 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>
```

Deployment:

```
1. npm i -g vercel
2. vercel
3. vercel –prod
```