Lesson:

Grid Container properties

PW SKILLS

Topics Covered

- Introduction to grid container
- display
- grid-template-columns
- grid-template-rows
- grid-auto-rows
- grid-auto-columns
- grid-template-areas
- grid-column-gap
- grid-row-gap
- justify-content
- align-content
- justify-items
- align-items

Introduction

A grid container is an HTML element that defines a grid layout. To control the layout of the grid items within the container, various CSS properties can be applied to the container itself. Let's discuss some of the most commonly used properties for styling a grid container.

display: grid

display property is set to "grid" to define the container as a grid container. Its initial value is block.

Let's suppose, we have a container div with six child divs inside it,

index.html

```
Unset
<div class="container">
    <div>1</div>
    <div>2</div>
    <div>3</div>
    <div>4</div>
    <div>5</div>
    <div>6</div>
</div>
```

To make the container as a grid, we need to specify the display as a grid.

style.css

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

grid-template-columns

(initial value - none)
grid-template-columns property specifies the number and size of columns in the grid. You can use values like pixels, percentages, or fractions to define the column widths. Let's understand it by some examples,

Example 1: Equal-sized columns
Now, we need to use the **grid-template-column: repeat(3, 1fr).** The **repeat()** function takes two arguments: the first argument specifies the number of times to repeat the pattern, and the second argument specifies the size of each grid column.

style.css

```
Unset
.container {
   display: grid;
   grid-template-columns: repeat(3, 1fr);
}

.container > div {
   padding: 5px;
   background-color: aquamarine;
   border: 1px solid green;
}
```

Browser Output -

1	2	3
4	5	6

In this example, the **repeat()** function repeats the pattern three times, and each column has a width of 1fr. The 1fr unit is a flexible unit that distributes the available space equally among the grid columns. For example, if the container has a width of 900px and three columns, each column will have a width of 300px.

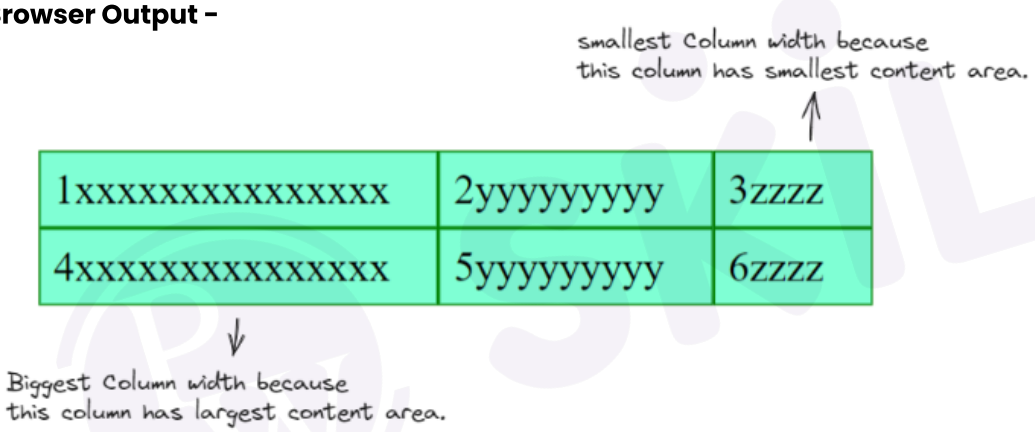
Example 2: Size columns automatically to fit content
Considering the below , to make columns automatically size according to the content width, use **grid-template-column : auto auto auto.**

style.css

```
Unset
.container{
  display: grid;
  grid-template-columns: auto auto auto;
}

.container > div{
  border:1px solid black;
}
```

Browser Output -



Example 3: Fixed columns widths
To set column width, use **grid-template-column: 100px 200px 300px.**

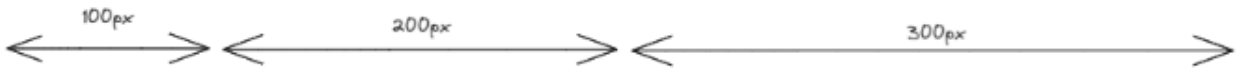
style.css

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

```
  grid-template-columns: 100px 200px 300px;
}

.container > div{
  border:1px solid black;
}
```

Browser Output -



1	2	3
4	5	6

In this example, we are creating columns with width **100px, 200px, and 300px** for the first, second, and third columns respectively.

grid-template-rows
(initial value - none)
Similarly, **grid-template-rows** property specifies the number and size of rows in the grid. You can use values like pixels, percentages, or fractions to define the row heights.

Example 1: Auto equal-sized rows

```
JavaScript
.container {
    display: grid;
    grid-template-rows: repeat(3, 1fr);
    grid-template-columns: repeat(2, 1fr);
    font-size: 32px;
}
```

```
.container > div {
    padding: 5px;
    background-color: aquamarine;
    border: 1px solid green;
}
```

Browser Output

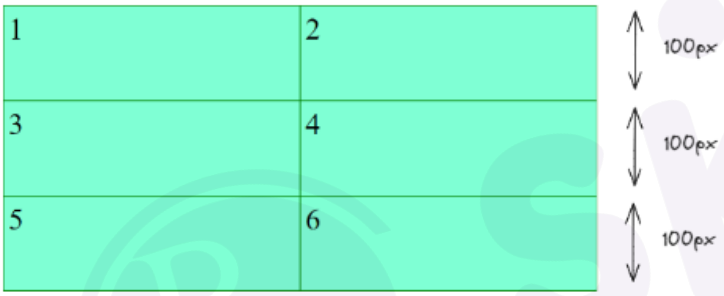
1	2
3	4
5	6

Example 2: Fixed and equal-sized rows.

```
JavaScript
.container {
      display: grid;
      grid-template-rows: 100px 100px 100px;
      grid-template-columns: repeat(2, 1fr);
      font-size: 32px;
    }

    .container > div {
      padding: 5px;
      background-color: aquamarine;
      border: 1px solid green;
    }
```

Browser Output



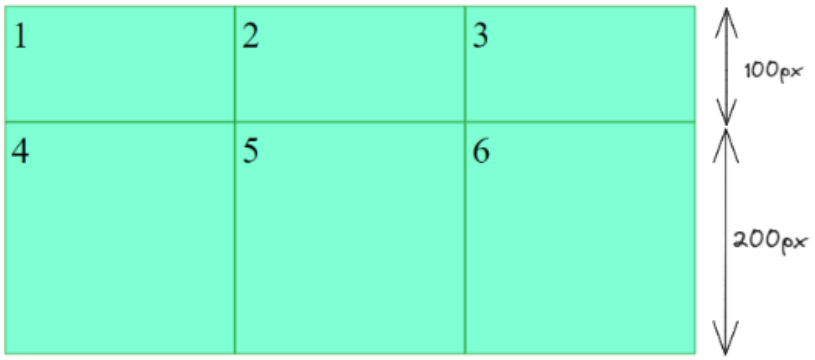
In this case, it creates three rows of the same height as 100px.

Example 3: Fixed and unequal rows.
style.css

```
Unset
.container {
    display: grid;
    grid-template-rows: 100px 200px;
    grid-template-columns: repeat(3, 1fr);
    font-size: 32px;
}

.container > div {
    padding: 5px;
    background-color: aquamarine;
    border: 1px solid green;
}
```

Browser Output



In this case, it creates two rows of different heights. The first row has a height of 100 pixels, and the second row has a height of 200 pixels.
If there are more rows in the grid than the number of values specified in grid-template-rows, the additional rows will have a height of auto.

If there are fewer rows in the grid than the number of values specified in grid-template-rows, the extra values will be ignored.

grid-template
(initial value - none)
grid-template is a shorthand property that combines grid-template-rows and grid-template-columns into a single property. It allows you to specify both the rows and columns of the grid in a single line.

Example: Grid of two rows and three columns.
1st way : Use grid-template-row and grid-template-column.
style.css

```
Unset
.container {
  display: grid;
  grid-template-rows: repeat(2, 1fr);
  grid-template-columns: repeat(3, 1fr);
}
```

2nd way : Use grid-template.
style.css

```
Unset
.container {
  display: grid;
  grid-template:
    repeat(2, 1fr)
```

```
    / repeat(3, 1fr);
}
```

Browser Output

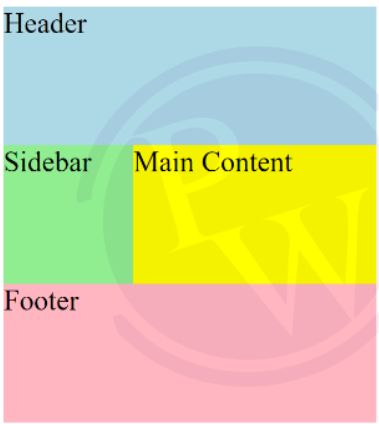
1	2	3
4	5	6

In the 2nd way, the repeat(3, 1fr) specifies that the grid should have three rows, with each row having a height of 1fr. The repeat(3, 1fr) specifies that the grid should have three columns, with each column having a width of 1fr.

The forward slash (/) separates the grid-template-rows from the grid-template-columns.

grid-template-areas
(initial value - none)
grid-template-areas property defines named grid areas within the grid container. Each area can be assigned a name and then referenced in the CSS rules for the grid items.

This approach can make it easier to create and visualize complex layouts, as you can see the overall structure of the grid just by looking at the CSS. It can also make it easier to modify the layout later, as you can simply adjust the grid-template-areas
Example: Page Layout with the header, footer, sidebar, main aligned as shown in below picture,



style.css

```
Unset
.grid-container {
  display: grid;
  grid-template-areas:
    "header header header "
    "sidebar main main"
    "footer footer footer";
  height: 90vh;
}

.top {
  grid-area: header;
  background-color: lightblue;
}

.left {
  grid-area: sidebar;
  background-color: lightgreen;
}
```

```
.right {
  grid-area: main;
  background-color: yellow;
}

.bottom {
  grid-area: footer;
  background-color: lightpink;
}
```

In this example, we're using grid-template-areas to define a grid with three rows and three columns. We've given names to each of the grid areas and used grid-area to assign each item to its corresponding grid area.

grid-auto-rows
(initial value - auto)
grid-auto-rows property specifies the height of rows that are automatically created when there is no explicit row definition.

index.html

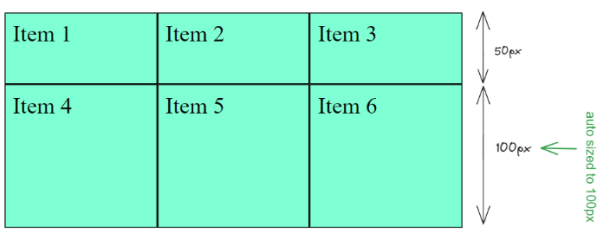
```
Unset
<div class="container">
    <div>Item 1</div>
    <div>Item 2</div>
    <div>Item 3</div>
    <div>Item 4</div>
    <div>Item 5</div>
    <div>Item 6</div>
</div>
```

style.css

```
Unset
.container {
    display: grid;
    grid-template-areas: "X X X";
    grid-template-rows: 50px;
    grid-auto-rows: 100px;
}

.container > div {
    border: 1px solid black;
    background-color: aquamarine;
    padding: 5px;
}
```

Browser Output



This example creates a grid container with three columns defined by grid-template-areas. The first row is displayed with height 50px because we have mentioned grid-template-rows for the first row as 50px. And the remaining rows will be displayed with a height of 100px, because of the grid-auto-rows:100px property. That's why the second row is displayed with a height of 100px.
In short, If there are any explicitly defined rows in the grid using the "grid-template-rows" property, then those rows will have their own heights, and the "grid-auto-rows" property will only apply to any additional rows that are created automatically.

grid-auto-columns
(initial value - auto)
Similarly, the grid-auto-columns property specifies the width of columns that are automatically created when there is no explicit column definition.

index.html

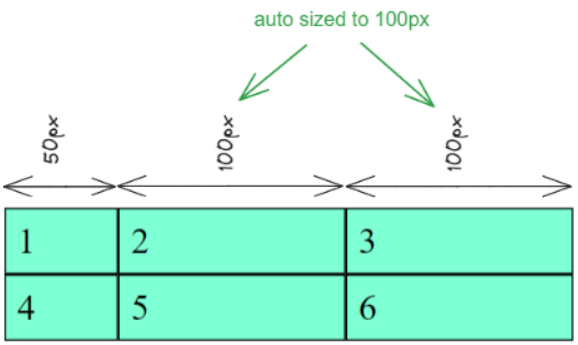
```
Unset
<div class="container">
    <div>Item 1</div>
    <div>Item 2</div>
    <div>Item 3</div>
    <div>Item 4</div>
    <div>Item 5</div>
    <div>Item 6</div>
</div>
```

```
Unset
 .container {
    display: grid;
    grid-template-areas: "X X X";
    grid-template-columns: 50px;
    grid-auto-columns: 100px;
}

.container > div {

    border: 1px solid black;
    background-color: aquamarine;
    padding: 5px;
}
```

Browser Output



This example creates a grid container with three columns defined by grid-template-areas. The first column is displayed with a width of 50px because we have mentioned grid-template-columns for the first row as 50px. And the remaining columns will be displayed with a width of 100px, because of the grid-auto-columns:100px property. That's why the second and third columns are displayed with a width of 100px.

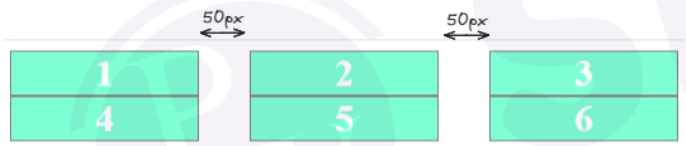
grid-column-gap
(initial value - 0)
grid-column-gap is a CSS property that defines the size of the gap or space between the columns of a grid. It is used to create space between the columns and make the grid more visually appealing. Another name for grid-column-gap is column-gap.

style.css

```
Unset
.container {
        display: grid;
        grid-template-columns: auto auto auto;
        grid-column-gap: 50px;
  }

 .container > div {
        background-color: aquamarine;
        border: 1px solid gray;
        color: white;
        text-align: center;
        font-size: 30px;
        font-weight: bold;
  }
```

Browser Output



In the above code, you can see that there is a gap of 50px between the columns which we were able to add using the property grid-column-gap

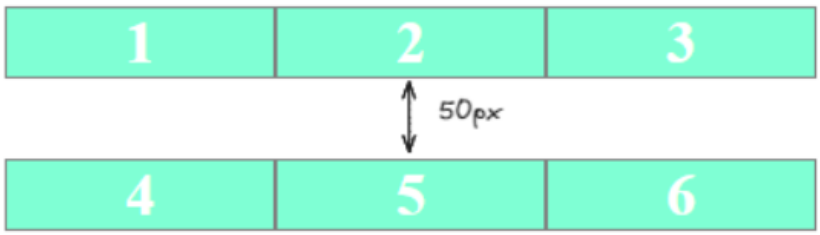
grid-row-gap
(initial value - 0)
grid-row-gap is a CSS property that defines the size of the gap or space between the rows of a grid. It is used to create space between the rows and make the grid more visually appealing. Another name for grid-row-gap is row-gap.

style.css

```
Unset
.container {
      display: grid;
      grid-template-columns: auto auto auto;
      grid-row-gap: 50px;
}

 .container > div {
      background-color: aquamarine;
      border: 1px solid gray;
      color: white;
      text-align: center;
      font-size: 30px;
      font-weight: bold;
 }
```

Browser Output



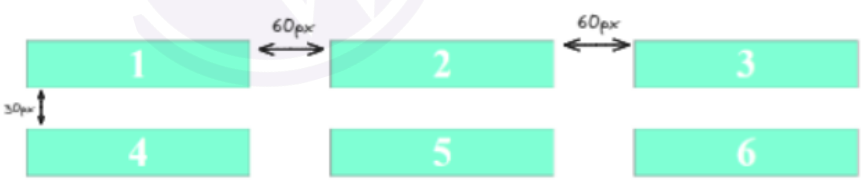
In the above code, you can see that there is a gap of 50px between the rows which we were able to add using the property grid-row-gap.

grid-gap
(initial value - 0)
grid-gap property sets the size of the gap between grid items, both horizontally and vertically. It's a shorthand property for grid-column-gap and grid-row-gap properties. Another name for grid-column-gap is gap.

Example 1:

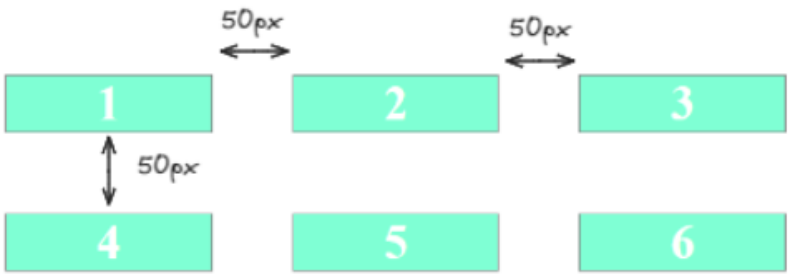
```
JavaScript
.container {
    display: grid;
    grid-template-columns: auto auto auto;
    grid-gap: 30px 60px;
}

.container > div {
     background-color: aquamarine;
     border: 1px solid gray;
     color: white;
     text-align: center;
     font-size: 30px;
     font-weight: bold;
}
```



It sets the gap between each row to 30px and the column to 60 pixels.
The grid-gap property is a shorthand property for setting both grid-row-gap and grid-column-gap. If you only want to set the gap between rows or columns, you can use the grid-row-gap or grid-column-gap properties separately.

Browser Output



It sets the gap between each row and column to 50 pixels.

justify-content
(default value - stretch)
It controls how the grid columns are arranged along the inline axis (left to right) i.e. horizontally.

Let's consider below HTML page,

index.html

```
Unset
<div class="container">
   <div>Item 1</div>
   <div>Item 2</div>
   <div>Item 3</div>
   <div>Item 3</div>
</div>
```

style.css

```
Unset
.container {
   display: grid;
   grid-template-columns: auto auto;
   border: 1px solid black;
}

.container > div {
   padding: 5px;
   background-color: aquamarine;
   border: 1px solid green;
}
```

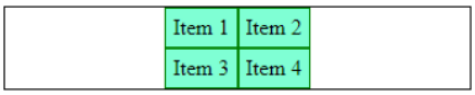
Browser Output

Item 1	Item 2
Item 3	Item 3

Now, we will see how items get arranged using each possible value of justify-content.

1. **justify-content: center -** Place grid items at the center, with no gap between items, unless provided explicitly.

```
Unset
.container{
   // other styles
   justify-content: center;
}
```



2. **justify-content: start -** Place items from the start, with no gap between items, unless provided explicitly.

```
Unset
.container{
   // other styles
   justify-content: start;
}
```

3. **justify-content: end -** Place items from the end, with no gap between items, unless provided explicitly.

```
Unset
.container{
   // other styles
   justify-content: end;
}
```

4. **justify-content: space-between -** Distributes items evenly, with no starting and ending gaps.

```
Unset
.container{
   // other styles
   justify-content: space-between;
}
```

5. **justify-content: space-around -** Distributes items evenly, with starting and ending gaps being half the size of the space between each item.

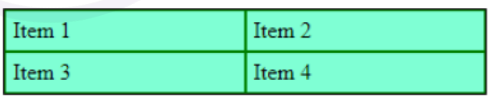
```
Unset
.container{
   // other styles
   justify-content: space-around;
}
```

6. **justify-content: space-evenly -** Distributes items evenly, with starting and ending gaps equal to space between each item.

```
Unset
.container{
   // other styles
   justify-content: space-evenly;
}
```

7. **justify-content: stretch -** Distributes items evenly, with no starting and ending gaps, and items (only auto-sized) are stretched to fill the available space.

```
Unset
.container{
   // other styles
   justify-content: stretch;
}
```



justify-items
(default value - stretch)
This property allows us to set the default way of justifying each item in their respective grid cells.

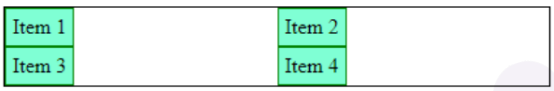
Now, we will see how items get arranged using each possible value of justify-items.

1. **justify-items: center -** Place grid items at the center of their respective grid cells.

```
Unset
.container{
   // other styles
   justify-items: center;
}
```

2. **justify-items: start -** Place grid items from the start of their respective grid cells.

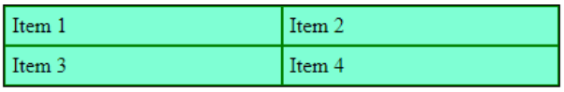
```
Unset
.container{
   // other styles
   justify-items: start;
}
```



3. **justify-items: end -** Place grid items from the end of their respective grid cells.

4. **justify-items: stretch -** Stretch grid items to fill the available space, in their respective grid cells. It is the default value of justify-items.

```
Unset
.container{
   // other styles
   justify-items: stretch;
}
```

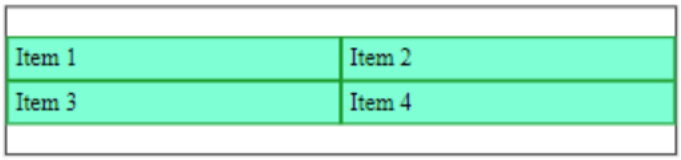


align-content
(default value - stretch)
It controls how the grid rows are arranged along the block axis (top to bottom) i.e. vertically.

Now, we will see how items get arranged using each possible value of align-content.

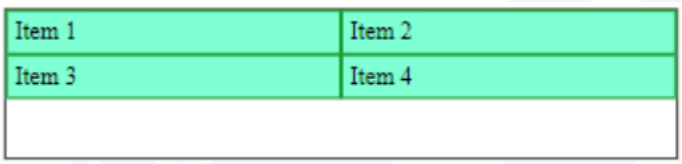
1. align-content: center - Place grid items at the center, with no gap between items, unless provided explicitly.

```
.container{
   // other styles
   align-content: center;
}
```



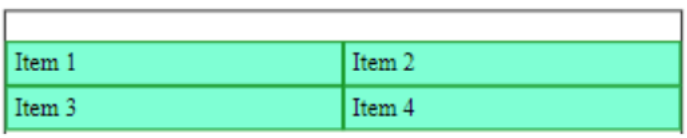
2. align-content: start - Place grid items from the start, with no gap between items, unless provided explicitly.

```
.container{
   // other styles
   align-content: start;
}
```



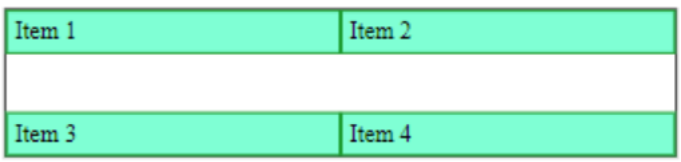
3. align-content: end - Place grid items from the end, with no gap between items, unless provided explicitly.

```
.container{
   // other styles
   align-content: end;
}
```



4. align-content: space-between - Distributes items evenly, with no starting and ending gaps.

```
.container{
   // other styles
   align-content: space-between;
}
```



5. align-content: space-around - Distributes items evenly, with starting and ending gaps being half the size of the space between each item.

```
.container{
   // other styles
   align-content: space-around;
}
```

Item 1	Item 2
Item 3	Item 4

6. align-content: space-evenly - Distributes items evenly, with no starting and ending gaps.

```
.container{
   // other styles
   align-content: space-evenly;
}
```

Item 1	Item 2
Item 3	Item 4

7. align-content: stretch - Distributes items evenly, with no starting and ending gaps, and items (only auto-sized) are stretched to fill the available space.

```
.container{
   // other styles
   align-content: stretch;
}
```

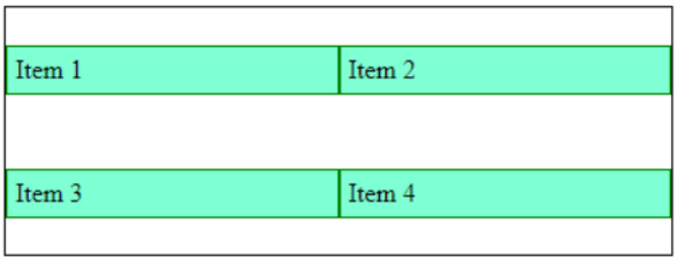
Item 1	Item 2
Item 3	Item 4

align-items
(default value - stretch)
This property allows us to set the default way of aligning each item in their respective grid cells along the block direction (top to bottom).

Now, we will see how items get arranged using each possible value of align-items.

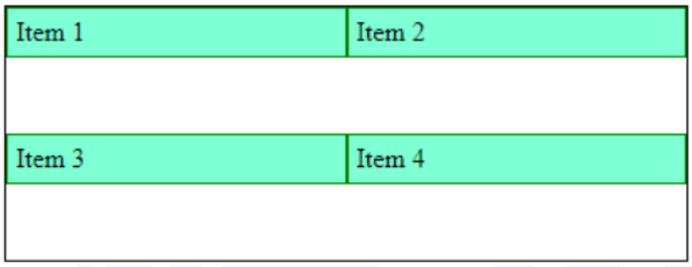
1. align-items: center - Place grid items at the center of their respective grid cells.

```
.container{
   // other styles
   align-items: center;
}
```



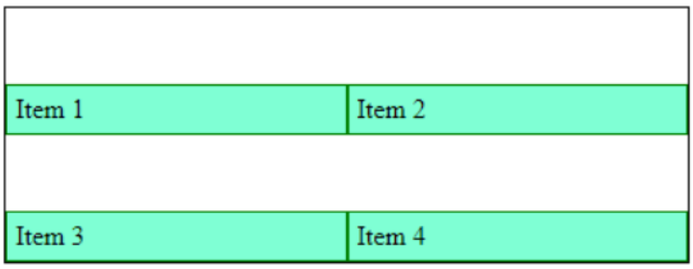
2. align-items: start - Place grid items from the start of their respective grid cells.

```
.container{
   // other styles
   align-items: start;
}
```



3. align-items: end **-** Place grid items from the end of their respective grid cells.

```
.container{
   // other styles
   align-items: end;
}
```



4. align-items: stretch - Stretch grid items to fill the available space, in their respective grid cells. It is the default value of justify-items.

```
.container{
   // other styles
   align-items: stretch;
}
```