MỤC LỤC

1. Sử dụng thuộc tính Float trong CSS để thiết kế Layout

1.1. Khái niệm về thuộc tính float

Thuộc tính float có tác dụng đẩy phần tử sang bên trái hoặc bên phải. Thuộc tính này thường được áp dụng vào việc thiết kế bố cục của trang web (Layout)

Phần menu bên trái	Phần nội dung	Phần quảng cáo

Với layout như trên, ta sử dụng CSS như sau:

```
<div style="width:30%; float:left;">
    Phần menu bên trái
</div>
<div style="width:50%; float:left">
    Phần nội dung
</div>
<div style="width:20%; float:left">
    Phần quảng cáo
</div>
```

1.2. Cách sử dụng thuộc tính float

```
float: left | right | none | inherit;
```

Trong đó:

- left: đẩy phần tử sang bên trái
- right: đẩy phần tử sang bên phải
- none: không thiết lập thuộc tính float (mặc định)
- inherit: kế thừa giá trị thuộc tính float của phần tử chứa nó

1.3. Nguyên tắc hoạt động của thuộc tính float

Khi một phần tử được thiết lập thuộc tính float:

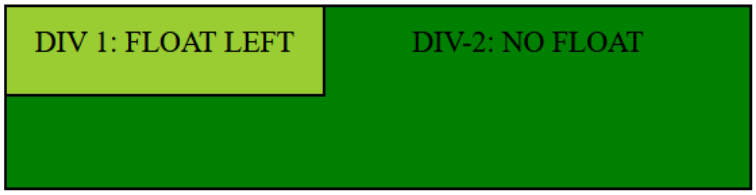
- Nó sẽ được bắt đầu ở hàng phía trên , nếu hàng phía trên còn đủ chỗ trống để chứa nó.
- Nó sẽ được bắt đầu ở hàng mới, nếu hàng phía trên không đủ chỗ trống để chứa nó.

Nếu một phần tử được thiết lập thuộc tính float mà trong khi phần tử đứng trước nó không được thiết lập thuộc tính float, thì mặc định nó được bắt đầu ở hàng mới.

Khi trên một hàng có nhiều phần tử được thiết lập thuộc tính float và mỗi phần tử có chiều cao khác nhau, nếu hàng không đủ chỗ chứa phần tử thì phần tử sẽ bắt đầu bên cạnh phần tử có chiều cao thấp nhất và còn đủ khoảng trống để chứa nó.

1.4. Tắt sự ảnh hưởng của thuộc tính float

Nếu một phần tử được thiết lập thuộc tính float, trong khi phần tử nằm phía sau nó không được thiết lập thuộc tính float, thì khi đó phần tử nằm phía sau sẽ bị tác động xấu bởi thuộc tính float.

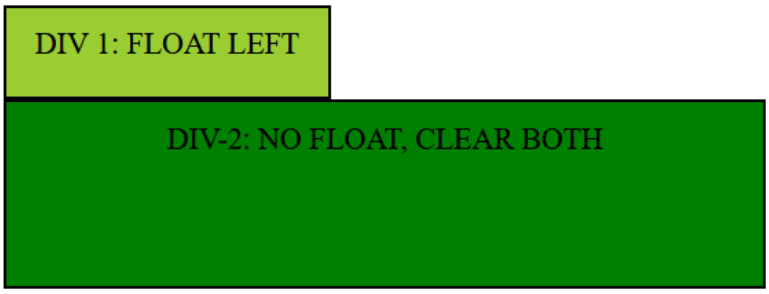


Để khắc phục vấn đề này, ta thêm thuộc tính clear vào phần tử phía sau theo cú pháp:

```
clear: none | left | right | both | inherit;
```

Trong đó:

- none: cho phép phần tử chịu sự ảnh hưởng của thuộc tính float
- left: không bị ảnh hưởng bởi thuộc tính float left
- right: không bị ảnh hưởng bởi thuộc tính float right
- both: không bị ảnh hưởng bởi thuộc tính float
- inherit: kế thừa giá trị thuộc tính clear của phần tử chứa nó



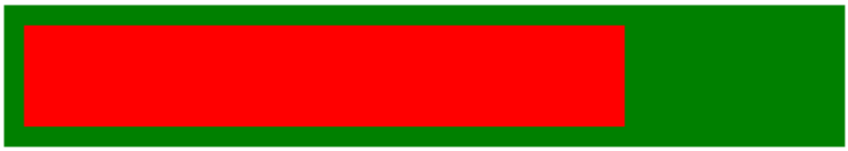
1.5. Chống tràn phần tử

Nếu một phần tử được thiết lập thuộc tính float và có kích thước lớn hơn phần tử chứa nó thì khi đó phần tử sẽ bị tràn ra khỏi phần tử chứa nó.

```
<!DOCTYPE html>
<html>
<head>
    <style>
```

```
        .box-1{
            width:400px;
            background-color: green;
            padding:10px;
        }
        .box-2{
            width:300px;
            height:50px;
            background-color:red;
            float:left;
        }
    </style>
</head>
<body>
    <div class="box-1">
        <div class="box-2"></div>
    </div>
</body>
</html>
```

Để khắc phục vấn đề này, ta thêm thuộc tính overflow:auto vào phần tử chứa nó.

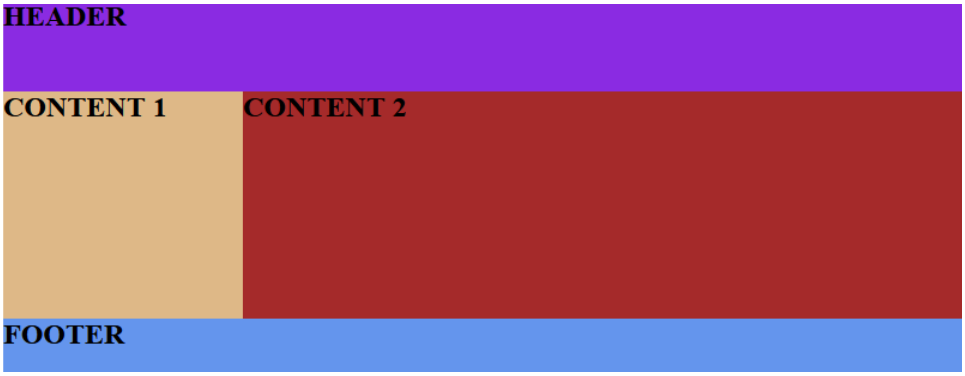


Ví dụ:

```
<!DOCTYPE html>
<html>
<head>
    <style>
        .box-1{
            width:400px;
            background-color: green;
            padding:10px;
            overflow: auto;
        }
        .box-2{
            width:300px;
            height:50px;
```

```
            background-color:red;
            float:left;
        }
    </style>
</head>
<body>
    <div class="box-1">
        <div class="box-2"></div>
    </div>
</body>
</html>
```

1.6. Ví dụ minh họa sử dụng Float để tạo Layout cho trang web




```
<!DOCTYPE html>
<html>
<head>
    <style>
        * {
            box-sizing: border-box;
            margin: 0;
            padding: 0;
        }
        .container {
            width:800px;
            margin:0 auto;
            background-color: #000;
```

```
        }
        #header {
            height: 80px;
            background-color: blueviolet;
        }
        #content-1 {
            min-height:200px;
            background-color: burlywood;
            width:200px;
            float:left;
        }
        #content-2 {
            min-height: 200px;
            background-color: brown;
            width:600px;
            float: left;
        }
        #footer {
            height:50px;
            background-color: cornflowerblue;
            clear: both;
        }
    </style>
</head>

<body>
    <div class="container">
        <div id="header">
            <h2>HEADER</h2>
        </div>
        <div id="content-1">
            <h2>CONTENT 1</h2>
        </div>
```

```
        <div id="content-2">
            <h2>CONTENT 2</h2>
        </div>
        <div id="footer">
            <h2>FOOTER</h2>
        </div>
    </div>
</body>
</html>
```

2. Sử dụng Flexbox trong thiết kế layout

2.1. Flexbox là gì?

Flexbox Layout (hay còn gọi là Flexible Box) là một kiểu bố cục trang có khả năng tự cân đối kích thước, thay đổi chiều rộng/chiều cao và thứ tự phần tử bên trong để phù hợp với tất cả các loại thiết bị hiển thị và kích thước màn hình.

Với bố cục thông thường, bạn cần phải thiết lập kích thước của phần tử, thiết lập hiển thị dạng block hay inline, cho nó float, còn với Flexbox bạn chỉ cần thiết lập phần hiển thị theo chiều ngang hay chiều dọc, lúc đó các phần tử bên trong có thể hiển thị theo ý muốn..

Lưu ý: Flexbox Layout phù hợp nhất để thiết lập bố cục ở quy mô nhỏ, còn thiết lập bố cục với phạm vi lớn hơn thì vẫn nên sử dụng kiểu thông thường là dàn trang theo dạng lưới (grid layout)

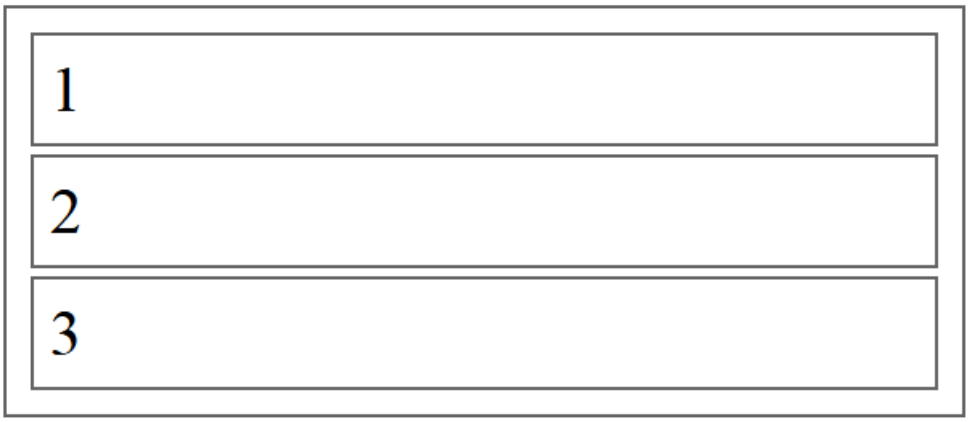
2.2. Sử dụng flex

Để sử dụng flex cho một thành phần nào đó trong CSS, chúng ta chỉ cần khai báo thuộc tính display của thành phần đó là flex (hoặc inline-flex). Trong trường hợp đó, thành phần này sẽ trở thành một flex container và các thành phần bên trong nó sẽ là các flex item

```
<!DOCTYPE html>
<html>
<head>
    <style>
        * {
            box-sizing: border-box;
            margin:5px;
            padding: 5px;
            font-size: 50px;
        }
        div {
            border:2px solid #646464;
            padding: 10px;
        }
        .container {
            width:600px;
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="item">1</div>
        <div class="item">2</div>
```

```
        <div class="item">3</div>
    </div>
</body>
</html>
```

Trong ví dụ trên, thành phần container chưa được thiết lập thuộc tính flex và giao diện khi hiển thị trên trình duyệt sẽ như sau:

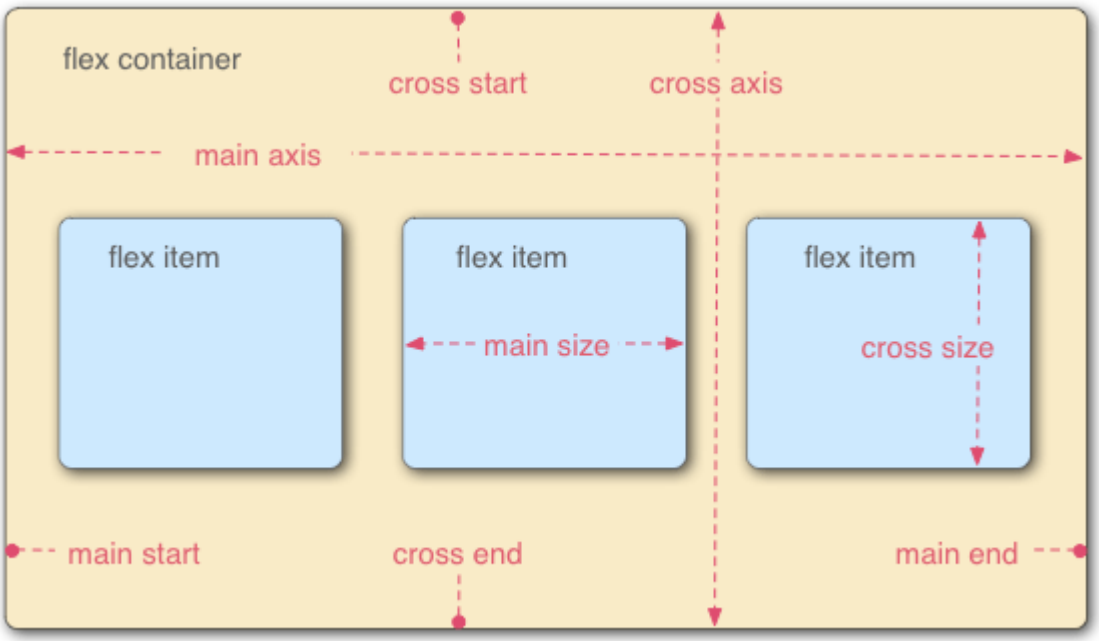


Nếu thiết lập thuộc tính display cho container:

```
.container {
     display: flex;
     width:600px;
}
```

Chúng ta sẽ nhận được kết quả như sau trên trình duyệt:

2.3. Các khái niệm trong Flexbox



- **Flex container**: là thành phần lớn bao quanh các phần tử bên trong, các item bên trong sẽ hiển thị dựa trên thiết lập của container này.
- **Flex item**: là phần tử con của container, bạn có thể thiết lập nó sẽ sử dụng bao nhiêu cột trong một container, hoặc thiết lập thứ tự hiển thị của nó.

  Các item sẽ được bố trí theo trục main axis (bắt đầu từ main-start, kết thúc ở main-end) hoặc theo trục cross axis (bắt đầu từ cross-start, kết thúc ở cross-end).

- **Main axis**: đây là trục chính để điều khiển hướng mà các item sẽ hiển thị. Cần lưu ý rằng, main axis không phải lúc nào cũng nằm ngang như sơ đồ trên, bạn có thể sử dụng thuộc tính flex-direction để thay đổi hướng của trục và lúc đó các item sẽ hiển thị theo nó.
- **Cross axis:** cross axis luôn là trục vuông góc của main axis. Hướng của nó phụ thuộc vào hướng của main axis.
- **Main start** và **main end**: khi thiết lập flexbox, các item nằm trong container hiển thị từ điểm bắt đầu gọi là main-start tới điểm kết thúc gọi là main-end.
- **Cross start** và **cross end**: có ý nghĩa tương tự như main start và main end nhưng luôn vuông góc với main start, main end.
- **Main size**: kích thước (chiều rộng hoặc chiều cao) của các item, tùy thuộc vào hướng của main axis.
- **Cross size**: kích thước (chiều rộng hoặc chiều cao) của các item dựa trên trục cross axis, tùy thuộc vào hướng của main axis.

2.4. Các thuộc tính của flex container

2.4.1. flex-direction

Thuộc tính này xác định hướng của main-axis để container sắp xếp các item.

Cú pháp:

```
.container {
  flex-direction: row | row-reverse | column | column-reverse;
}
```

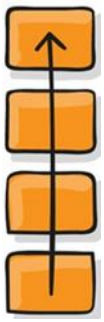
Trong đó:

- **row**: mặc định, flex item được sắp xếp theo chiều ngang, **từ trái qua phải** (main axis nằm ngang).

- **row-reverse:** flex item được sắp xếp theo chiều ngang, **từ phải qua trái** (main axis nằm ngang).

- **column:** flex item được sắp xếp theo chiều dọc, **từ trên xuống dưới** (main axis đứng dọc).

- **column-reverse:** flex item được sắp xếp theo chiều dọc, **từ dưới lên trên** (main axis đứng dọc).



2.4.2. flex-wrap

Theo mặc định, item sẽ tự động thay đổi kích thước phần tử để nó luôn hiển thị trên cùng một dòng dù bạn có resize trình duyệt theo kích thước nào, điều này dễ làm cho nội dung bên trong (nếu có) bị giãn hay ép nhỏ lại, có thể gây xấu giao diện.

Vì vậy, ta có **thuộc tính flex-wrap** cho phép item tự động xuống dòng khi kích thước container thay đổi.

Cú pháp:

```
.container{
  flex-wrap: nowrap | wrap | wrap-reverse;
}
```

Trong đó:

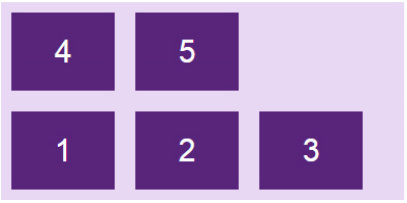
- **nowrap:** mặc định, tất cả các item sẽ nằm trên một dòng.
- **wrap:** khi kích thước container thay đổi và tổng chiều rộng các item cộng lại lớn hơn chiều rộng của container thì item sẽ tự động xuống dòng.
- **wrap-reverse:** tương tự như **wrap,** nhưng thay vì xuống dòng thì item sẽ tự động nhảy lên trên.

Ví dụ:

```
.container {
  display: flex;
  flex-wrap: nowrap;
}
```


```
.flex-container {
  display: flex;
  flex-wrap: wrap;
}
```


```
.flex-container {
  display: flex;
  flex-wrap: wrap-reverse;
}
```



2.4.3. flex-flow

Thuộc tính flex-flow sử dụng để gộp chung hai thuộc tính flex-direction và flex-wrap.

Cú pháp:

```
flex-flow: <'flex-direction'> || <'flex-wrap'>
```

Ví dụ:

```
.flex-container {
   display: flex;
   flex-flow: row wrap;
}
```

2.4.4. justify-content

Theo mặc định, các item bên trong sẽ bắt đầu từ main start đến main end, tuy nhiên, đôi khi container vẫn còn khoảng trống. Vì vậy, bạn có thể sử dụng **thuộc tính justify-content** để

điều chỉnh vị trí bắt đầu và căn chỉnh các item bên trong container theo **dọc theo trục main axis**, chiều ngang hoặc chiều dọc tùy thuộc vào flex-direction.

Cú pháp:

```
.container {
   justify-content: flex-start | flex-end | center |
                    space-between | space-around | space-evenly;
}
```

Trong đó:

- **flex-start:** giá trị mặc định, item sẽ bắt đầu từ lề chính **main-start** của container.
- **flex-end:** item sẽ bắt đầu từ lề chính **main-end** của container (khác với row-reverse là đổi hướng hiển thị).
- **center:** item sẽ nằm giữa container.
- **space-between:** các item sẽ có khoảng cách giữa các phần tử bằng nhau do container sẽ tự động căn khoảng cách, item đầu tiên sát lề chứa điểm main-start, item cuối cùng sát lề chứa điểm main-end.
- **space-around:** tương tự space-between, nhưng khác ở chỗ là mỗi item có khoảng cách 2 bên cạnh và những khoảng cách này bằng nhau.
- **space-evenly:** các item được phân phối sao cho khoảng cách giữa hai item bất kỳ, giữa item và các lề là bằng nhau.

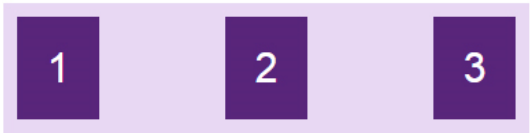
Ví dụ: Xét trường hợp flex với trục main axis nằm ngang

```
.container {
  display: flex;
  justify-content: flex-start;
}
```

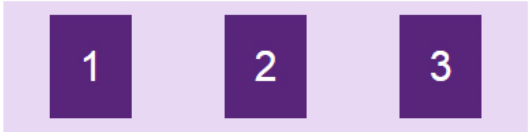

```
.container {
  display: flex;
  justify-content: flex-end;
}
```


```
.container {
  display: flex;
  justify-content: center;
}
```

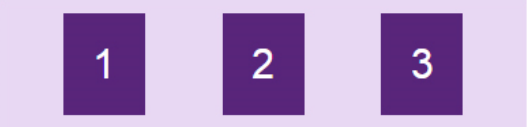

```
.container {
  display: flex;
  justify-content: space-between;
}
```




```
.container {
  display: flex;
  justify-content: space-evenly;
}
```

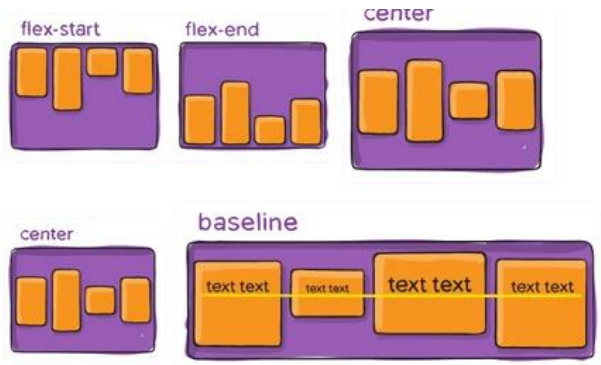


```
.container {
  display: flex;
  justify-content: space-around;
}
```



2.4.5. align-items

Thuộc tính align-items sử dụng để điều chỉnh vị trí bắt đầu và căn chỉnh các item bên trong container theo dọc theo trục cross axis, chiều ngang hoặc chiều dọc tùy thuộc vào flex-direction.

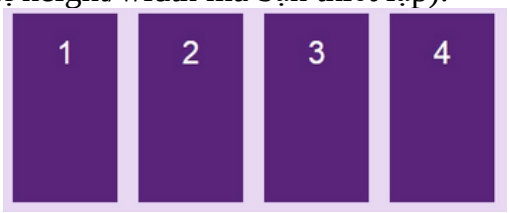


Cú pháp:

```
.container {
  align-items: stretch | flex-start | flex-end |
               center | baseline;
}
```

Trong đó:

- **stretch:** giá trị mặc định, các phần tử sẽ được kéo dài để lấp đầy container chứa nó, nhưng sẽ ưu tiên giá trị height/width nếu có (khi đó item sẽ không cao lấp đầy vùng container mà chỉ lấy giá trị height/width mà bạn thiết lập).

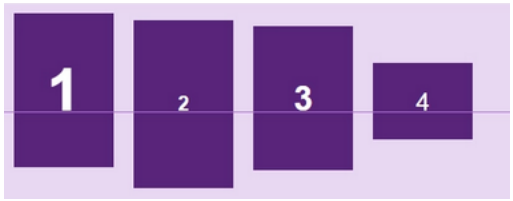


- **flex-start:** item sẽ bắt đầu từ lề **cross-start** của container.

- **flex-end:** item sẽ bắt đầu từ lề cross-end của container. align-content

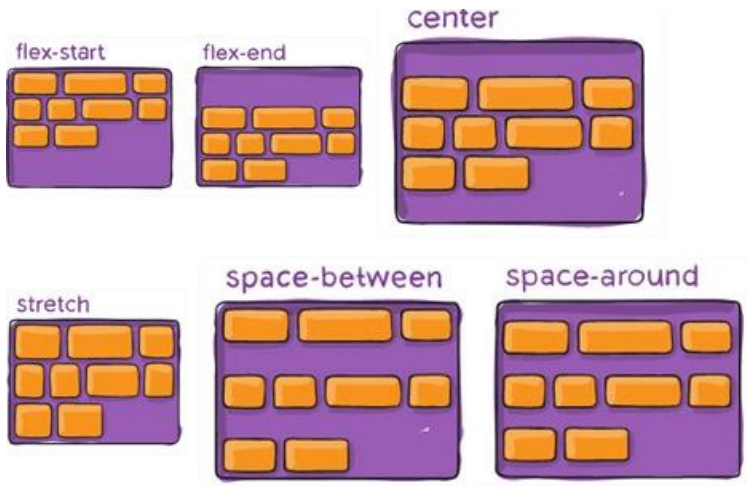
- **center:** item sẽ căn giữa theo chiều của cross axis.

- **baseline:** item được căn chỉnh theo đường cơ sở của chúng. Đường cơ sở là đường mà tất cả các chữ cái sẽ nằm trên đó.



2.4.6. align-content

Thuộc tính align-content sử dụng để **căn chỉnh khoảng cách** các item bên trong container theo dọc theo trục cross axis, chiều ngang hoặc chiều dọc tùy thuộc vào flex-direction.



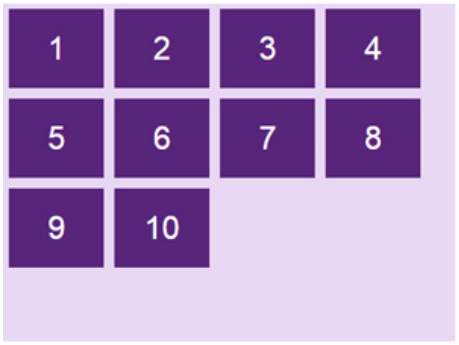
Cú pháp:

```
.container {
  align-content: flex-start | flex-end | center |
                 space-between | space-around | stretch;
}
```

Trong đó:

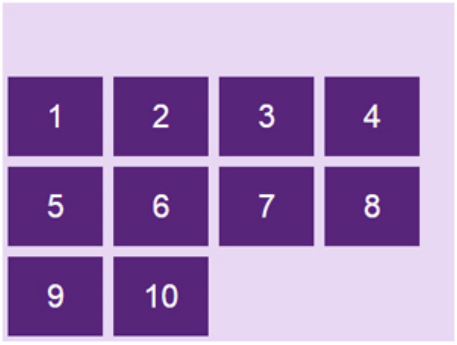
- **flex-start:** item sẽ bắt đầu từ lề chứa cross-start của container.

```
.container {
  display: flex;
  flex-wrap: wrap;
  align-content: flex-start;
}
```



- **flex-end:** item sẽ bắt đầu từ lề chứa cross-end của container.

```
.container {
  display: flex;
  flex-wrap: wrap;
  align-content: flex-end;
}
```

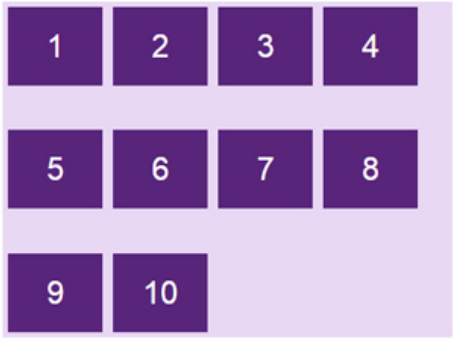


- **center:** item sẽ nằm giữa container căn theo cross-axis.

```
.container {
  display: flex;
  flex-wrap: wrap;
  align-content: center;
}
```

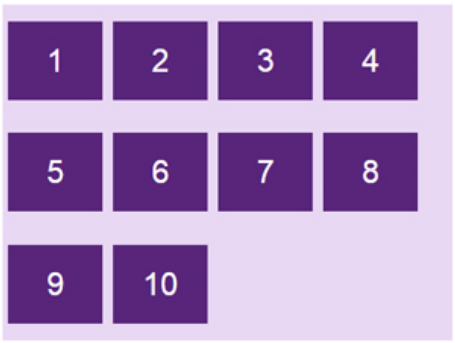
- **space-between:** các item sẽ có khoảng cách giữa các phần tử bằng nhau do container tự động căn khoảng cách, item đầu tiên sát lề chứa cross-start, item cuối cùng sát lề chứa cross-end.

```
.container {
  display: flex;
  flex-wrap: wrap;
  align-content: space-between;
}
```



- **space-around:** tương tự space-between, nhưng khác ở chỗ là mỗi item có khoảng cách 2 bên cạnh và những khoảng cách này bằng nhau.

```
.container {
  display: flex;
  flex-wrap: wrap;
  align-content: space-around;
}
```



- **stretch**: giá trị mặc định, các phần tử sẽ được kéo dài, căn chỉnh sao cho lấp đầy container chứa nó (vẫn ưu tiên giá trị height/width nếu có).

```
.container {
  display: flex;
  flex-wrap: wrap;
  align-content: stretch;
}
```

2.5. Các thuộc tính của flex item

2.5.1. order

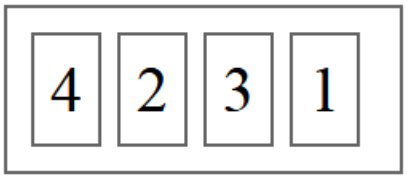
Theo mặc định, các item sẽ hiển thị theo thứ tự xuất hiện trong mã HTML, nhưng với **thuộc tính order**, bạn có thể sắp xếp lại vị trí sắp xếp của các item.

Cú pháp:

```
.item {
  order: <integer>; /* mặc định là 0 */
}
```

Ví dụ:

```
<div class="container">
   <div style="order: 4">1</div>
   <div style="order: 2">2</div>
   <div style="order: 3">3</div>
   <div style="order: 1">4</div>
</div>
```



2.5.2. flex-grow

Thuộc tính flex-grow cho phép các phần tử giãn theo độ rộng của container.

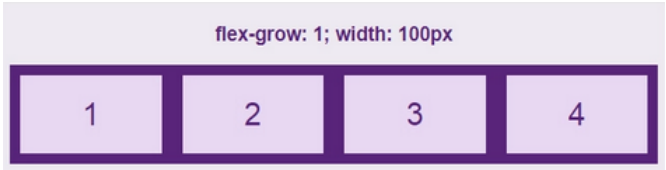
Cú pháp:

```
.item {
  flex-grow: <number>; /* mặc định là 0 */
}
```

Chúng ta có thể hình dung tác động của thuộc tính này thông qua các tình huống minh họa sau đây, với một flex container bao gồm 4 item:

- Các item được thiết lập với độ rộng là 100px. Trong trường hợp này, giá trị mặc định của thuộc tính **flex-grow là 0**, các item sẽ không tự động co giãn kích thước, để lại nhiều khoảng trống trong container.

- Khi ta tăng **flex-grow = 1,** item sẽ tự động giãn ra đều nhau sao cho vừa với khung container.

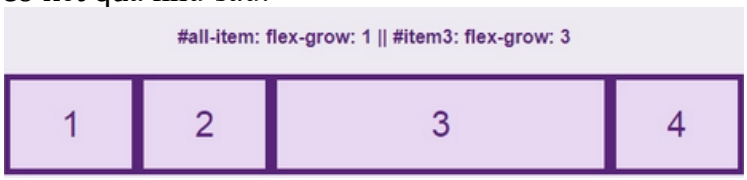


Giá trị của flex-grow rất linh động, khi thiết lập thuộc tính này cho **tất cả các item với cùng một giá trị** thì các item sẽ **có tỉ lệ như nhau** và được dàn đều lấp đầy container. Ví dụ thiết lập tất cả các phần tử flex-grow là 1 thì tất cả đều như nhau tỉ lệ 1:1, và nếu thiết lập flex-grow là 100 thì kết qua vẫn sẽ ra tương tự với tỉ lệ 1:1.

- Ta có giá trị mặc định ở tất cả phần tử là flex-grow = 0, thay đổi riêng giá trị của item2 thành 1, kết quả như sau:

Ta có thể thấy item2 sẽ lấy phần trống còn lại của container đắp vào chính nó.

- Nếu chúng ta thiết lập giá trị thuộc tính flex-grow cho phần tử thứ 3 là 3 và các phần tử còn lại là 1, ta sẽ có kết quả như sau:



Lúc này thì tất cả các item đều được giãn ra lấp đầy phần trống của container, nhưng item3 có flex-grow: 3 sẽ được thừa hưởng nhiều phần trống hơn các item còn lại chỉ có flex-grow: 1, cụ thể là hơn khoảng 3 lần

2.5.3. flex-shrink

Thuộc tính flex-shrink ngược lại với thuộc tính flex-grow ở trên, nó không giãn ra mà lại co lại khi chúng ta thay đổi độ rộng của container.

Cú pháp:

```
.item {
  flex-shrink: <number>; /* mặc định là 1 */
}
```

Giá trị mặc định trong flex-shrink là 1, cho phép các phần tử **co lại bằng nhau** khi độ rộng container giảm xuống. Nếu flex-shrink: 0 thì item sẽ **không co giãn** mà lấy nguyên giá trị của thuộc tính width/height.

Nếu muốn item3 nó co lại nhiều hơn so với các item khác thì tăng giá trị flex-shrink của nó lên.

2.5.4. flex-basis

Thuộc tính flex-basis sử dụng để xác định độ dài ban đầu của một item.

Cú pháp:

```
.item {
    flex-basis: <length> | auto; /* mặc định là auto */
}
```

2.5.5. flex

Thuộc tính flex sử dụng để gộp chung ba thuộc tính **flex-grow, flex-shrink và flex-basis.**

Cú pháp:

```
.item {
  flex: none | [ <'flex-grow'> <'flex-shrink'> || <'flex-basis'> ]
}
```

Ví dụ: Thay vì phải sử dụng cả 3 thuộc tính:

```
.item {
  flex-grow: 1;
  flex-shrink: 3;
  flex-basis: 250px;
}
```

Thì bạn có thể sử dụng thuộc tính flex một cách ngắn gọn:

```
flex: 1 3 250px;
```

Giá trị mặc định của flex là:

```
flex: 0 1 auto;
```

Lưu ý:

- Nếu thuộc tính chỉ có một tham số thế này: *flex: 1*; thì ta hiểu đó là **flex-grow.**
- Nếu thuộc tính chỉ có một tham số có đơn vị độ dài như thế này: *flex: 250px;* thì ta hiểu đó là **flex-basis**.

- Nếu thuộc tính có hai tham số thế này: *flex: 1 250px;* thì ta hiểu đó là **flex-grow và flex-basis.**
- Nếu thuộc tính có hai tham số thế này: *flex: 1 2;* thì ta hiểu đó là **flex-grow và flex-shrink.**

2.5.6. align-self

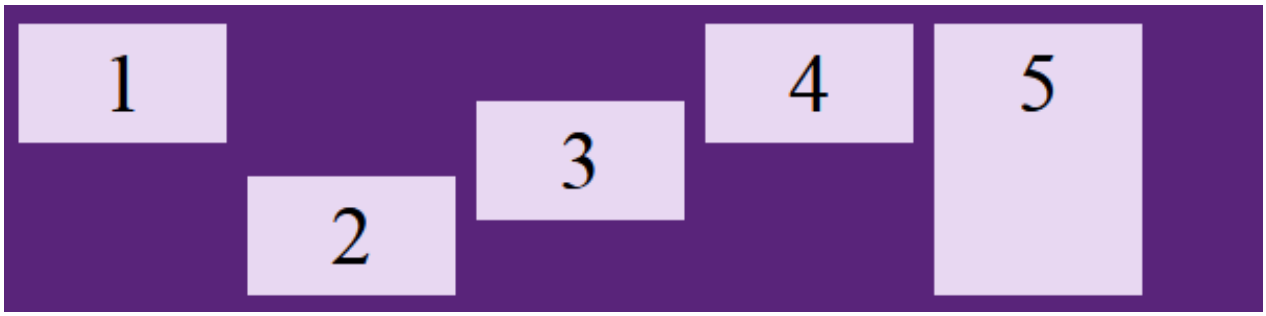
Thuộc tính align-self có tác dụng tương tự như align-items của phần container nhưng sử dụng riêng cho từng item, bạn có thể dùng nó để đặt lại vị trí cho một số item mà align-items đã quy định.

Cú pháp:

```
.item {
    align-self: auto | flex-start | flex-end | center
                | baseline | stretch;
}
```

Ví dụ:

```
<div class="container" style="display:flex; height:150px">
        <div class="item" style="align-self: flex-start">1</div>
        <div class="item" style="align-self: flex-end">2</div>
        <div class="item" style="align-self: center">3</div>
        <div class="item" style="align-self: baseline">4</div>
        <div class="item" style="align-self: stretch">5</div>
</div>
```



3. Thiết kế Layout với CSS Grid

3.1. Grid là gì?

CSS grid là chế độ hiển thị (display) được áp dụng cho một phần tử khi chúng ta cần chia bề mặt của chúng thành một lưới (grid) bao gồm các dòng (row) và các cột (column). Khi một phần tử được thiết lập chế độ hiển thị dưới dạng grid, các phần tử bên trong nó sẽ được bố trí trong các ô lưới.

CSS grid cho phép bạn có thể dễ dàng thiết kế layout và bố trí các phần tử trên trang web thay vì phải sử dụng các thuộc tính float và position.

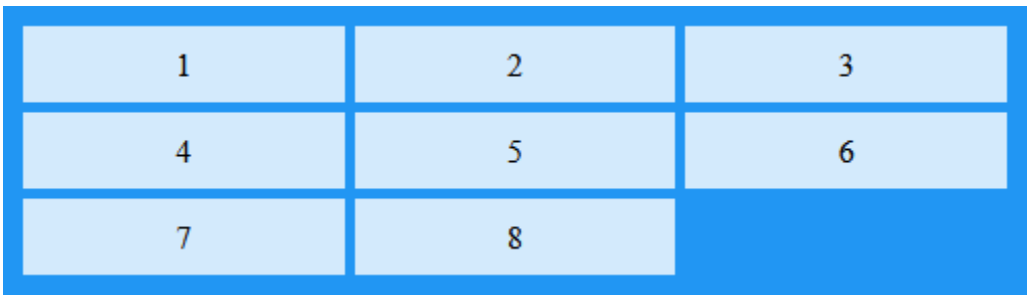
Một bố cục dạng grid bao gồm:

- Một phần tử cha (grid container)
- Một hoặc nhiều phần tử con nằm bên trong phần tử cha (grid items)

Ví dụ: Đoạn mã dưới đây thiết lập chế độ hiển thị dạng grid được chia thành 3 cột có độ rộng bằng nhau.

```
<!DOCTYPE html>
<html>
<head>
    <title>Grid Example</title>
    <style>
        .grid-container {
```

```
            display: grid;
            grid-template-columns: auto auto auto;
            gap:5px;
            background-color: #2196F3;
            padding: 10px;
        }
        .grid-item {
            background-color: #D3EAFC;
            text-align: center;
            padding:10px;
        }
    </style>
</head>
<body>
    <div class="grid-container">
        <div class="grid-item">1</div>
        <div class="grid-item">2</div>
        <div class="grid-item">3</div>
        <div class="grid-item">4</div>
        <div class="grid-item">5</div>
        <div class="grid-item">6</div>
        <div class="grid-item">7</div>
        <div class="grid-item">8</div>
    </div>
</body>
</html>
```

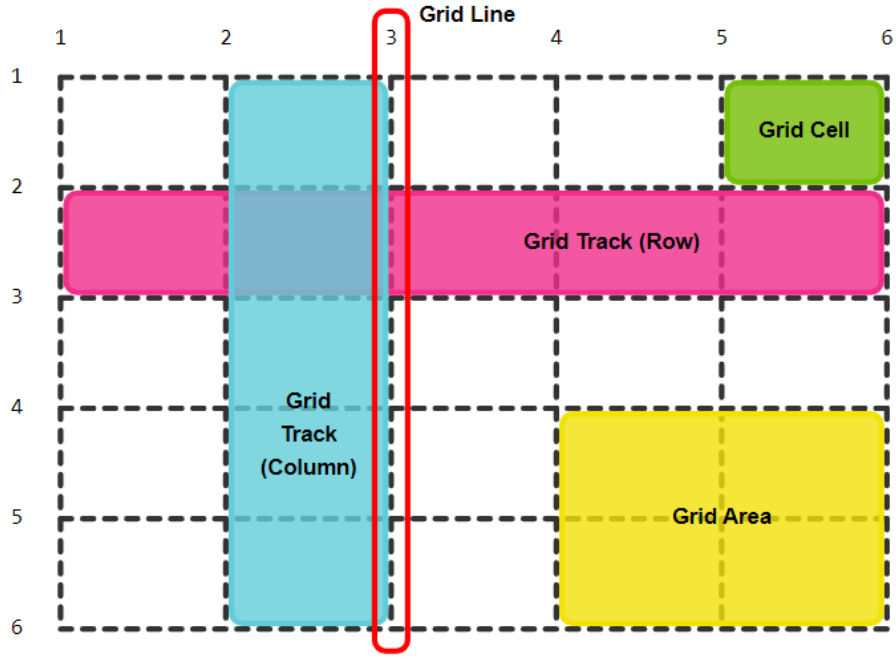


Để một phần tử được bố cục dưới dạng grid (tức là trở thành một grid container), bạn thiết lập giá trị thuộc tính display của phần tử là grid hoặc inline-grid. Khi đó, tất cả các phần tử là con của grid container sẽ là các grid item

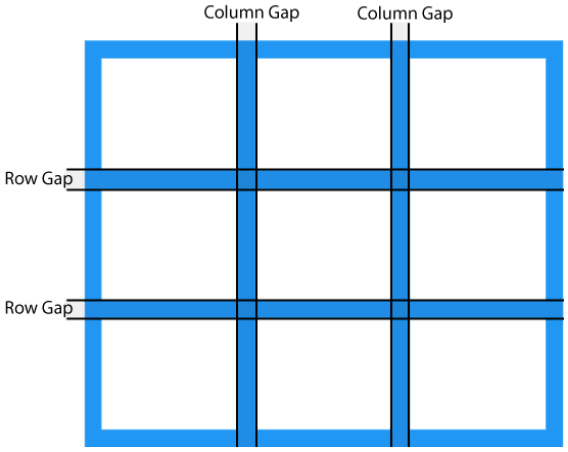
Cú pháp:

```
display: grid | inline-grid
```

3.2. Các khái niệm trong CSS Grid



- **Grid tracks**: Là hàng (row) hoặc cột (column) trong grid.
- **Grid cell**: Là một ô được tạo bởi giao của hàng và cột.
- **Grid area**: Là một vùng hình chữ nhật được tạo ra bởi tập các ô gần nhau.
- **Grid lines**: Là đường kẻ nằm ngang hoặc thẳng đứng phân chia bề mặt của grid thành các ô.
- **Grid gaps**: Là vùng đệm tạo ra khoảng cách giữa các dòng hoặc các cột



3.3. Tạo bố cục grid sử dụng grid-template-columns và grid-template-rows

Thuộc tính **grid-template-columns** được sử dụng để đặc tả số lượng và độ rộng của các cột trong một grid. Các giá trị được phân cách nhau bởi khoảng trắng và mỗi giá trị tương ứng với độ rộng của một cột trong lưới.

Cú pháp:

```
grid-template-columns: none|auto|length|max-content|min-content
                       |initial|inherit;
```

Trong đó:

none	Giá trị mặc định. Cột sẽ được tạo ra nếu cần
auto	Độ rộng của cột được xác định dựa vào kích thước của vùng chứa (container) và kích thước nội dung của các phần tử trong cột.
length	Là giá trị cụ thể nhằm thiết lập độ rộng của cột
max-content	Kích thước của cột dựa vào phần tử lớn nhất trong cột
min-content	Kích thước của cột dựa vào phần tử nhỏ nhất trong cột

Tương tự, thuộc tính **grid-template-rows** được sử dụng để đặc tả số lượng và chiều cao của các dòng trong một grid. Các giá trị cũng được phân cách nhau bởi khoảng trắng và mỗi giá trị tương ứng với chiều cao của một dòng trong lưới.

Cú pháp:

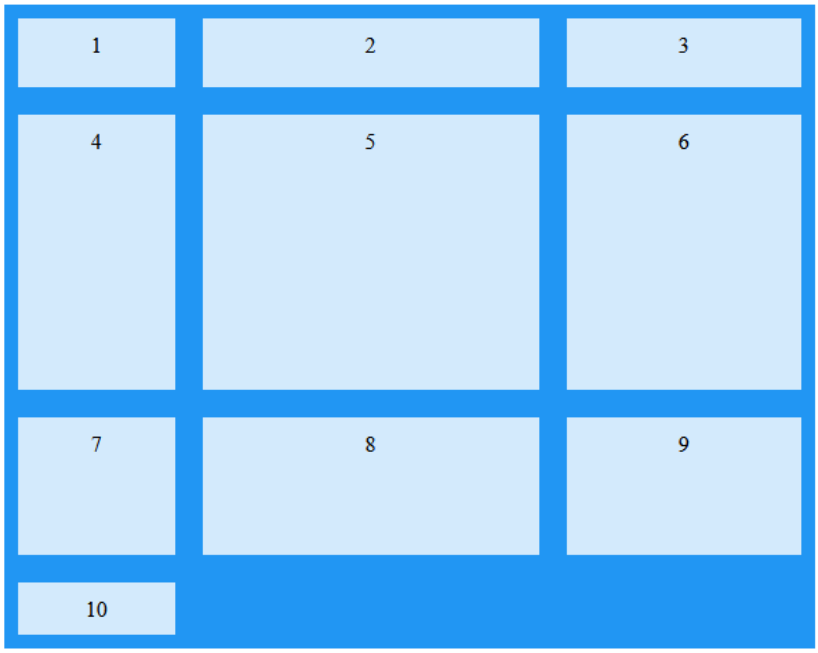
```
grid-template-rows:    none|auto|length|max-content|min-content
                       |initial|inherit;
```

Ví dụ dưới đây tạo ra một grid có bố cục bao gồm:

- 3 cột có độ rộng lần lượt là 20%, auto và 30%
- 3 dòng trên cùng có chiều cao lần lượt là 50px, 200px và 100px

```
<!DOCTYPE html>
<html>
<head>
    <style>
        .grid-container {
            display: grid;
            grid-template-columns: 20% auto 30%;
            grid-template-rows: 50px 200px 100px;
            gap:20px;
            background-color: #2196F3;
            padding: 10px;
        }
        .grid-item {
            background-color: #D3EAFC;
            text-align: center;
            padding:10px;
        }
```

```
    </style>
</head>
<body>
    <div class="grid-container">
        <div class="grid-item">1</div>
        <div class="grid-item">2</div>
        <div class="grid-item">3</div>
        <div class="grid-item">4</div>
        <div class="grid-item">5</div>
        <div class="grid-item">6</div>
        <div class="grid-item">7</div>
        <div class="grid-item">8</div>
        <div class="grid-item">9</div>
        <div class="grid-item">10</div>
    </div>
</body>
</html>
```



Trong ví dụ trên, bạn có thể thấy phần tử thứ 10 được đưa xuống dòng bên dưới và tạo nên đường gọi là Implicit Track. Các dòng Implicit Track này sẽ có chiều cao tự động trừ khi bạn thiết lập thuộc tính **grid-auto-rows** cho container.

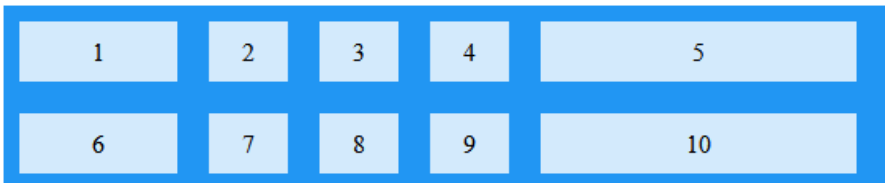
3.3.1. Sử dụng hàm repeat()

Khi khai báo **grid-template-columns/rows**, trong trường hợp cần lặp lại nhiều lần một khai báo dòng/cột, bạn có thể sử dụng hàm **repeat()**

Ví dụ:

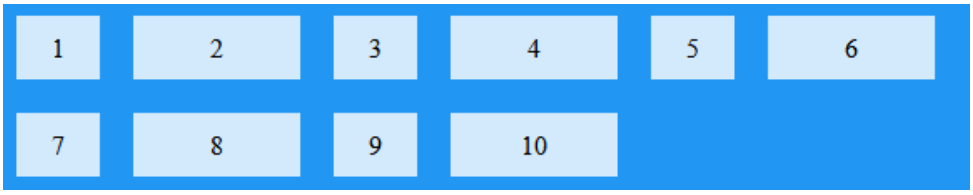
Tạo grid gồm 5 cột với các cột từ thứ 2 đến thứ 4 có độ rộng là 50px

```
grid-container {
     display: grid;
     grid-template-columns: 100px repeat(3, 50px) 200px;
     gap:20px;
     background-color: #2196F3;
     padding: 10px;
}
```



Tạo grid gồm 6 cột:

```
.grid-container {
    display: grid;
    grid-template-columns: repeat(3, 50px 100px);
    gap:20px;
    background-color: #2196F3;
    padding: 10px;
}
```



3.3.2. Đơn vị tính fr (Fraction) trong grid

Đơn vị **fr** (fraction/phân số) là đơn vị mới được sử dụng trong grid. Mỗi một fr (1fr) tương ứng với một phần trong không gian trống của grid container.

Ví dụ: Trong ví dụ dưới đây, 4 cột sau cùng của grid sẽ chia nhau mỗi cột chiếm 1/4 phần trống còn lại trong grid-container

```
.grid-container {
     display: grid;
     width:500px;
     grid-template-columns: 100px repeat(4, 1fr);
     gap:20px;
     background-color: #2196F3;
     padding: 10px;
}
```

3.3.3. Thiết lập vùng hiển thị của Grid item

Để thiết lập vùng hiển thị của một Grid item, bạn sử dụng các thuộc tính sau:

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

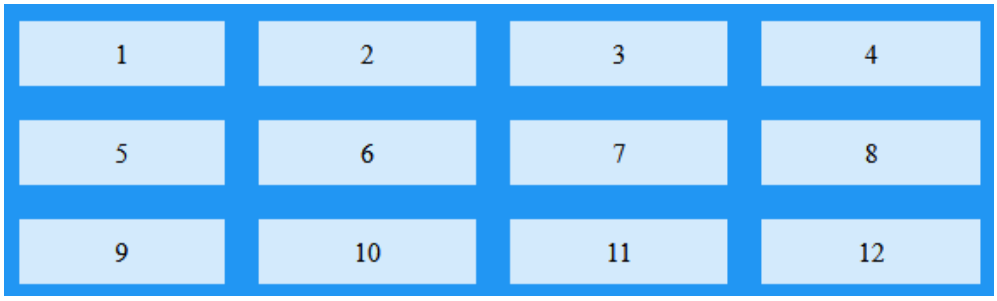
Cú pháp:

```
grid-column-start: auto|span n|column-line;
grid-column-end: auto|span n|column-line;
grid-row-start: auto|span n|row-line;
grid-row-start: auto|span n|row-line;
```

Ví dụ: Xét grid được tạo bởi đoạn mã sau:

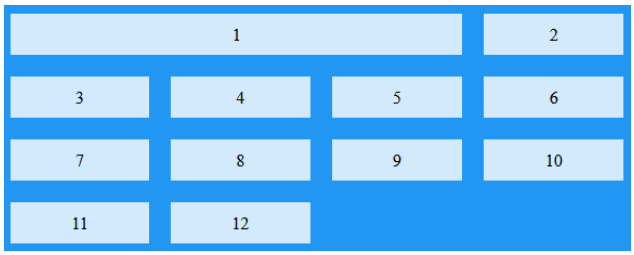
```
<!DOCTYPE html>
<html>
<head>
    <style>
        .grid-container {
            display: grid;
            grid-template-columns: auto auto auto auto;
            gap:20px;
            background-color: #2196F3;
            padding: 10px;
        }
        .grid-container > div {
            background-color: #D3EAFC;
            text-align: center;
            padding:10px;
        }
    </style>
</head>
<body>
    <div class="grid-container">
        <div class="item1">1</div>
        <div class="item2">2</div>
        <div class="item3">3</div>
        <div class="item4">4</div>
        <div class="item5">5</div>
        <div class="item6">6</div>
```

```
        <div class="item7">7</div>
        <div class="item8">8</div>
        <div class="item9">9</div>
        <div class="item10">10</div>
        <div class="item11">11</div>
        <div class="item12">12</div>
    </div>
</body>
</html>
```

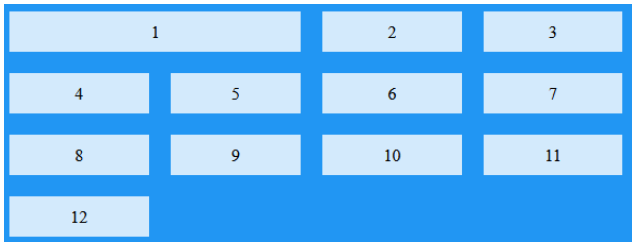


Chúng ta sẽ cùng tìm hiểu cách sử dụng 4 thuộc tính trên bằng cách thiết lập giá trị các thuộc tính cho phần tử đầu tiên trong lưới (item1) với các tình huống sau:

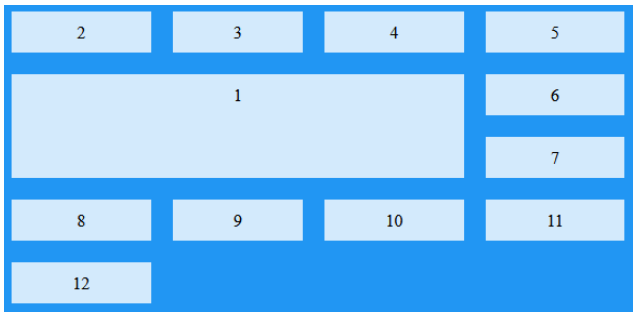
```
.item1 {
    grid-column-start: 1;
    grid-column-end: 4;
}
```



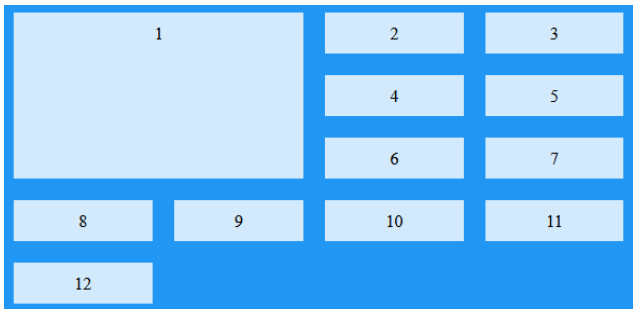

```
.item1 {
     grid-column-start: span 2;
}
```




```
.item1 {
    grid-column-start: 1;
    grid-column-end: 4;
    grid-row-start:2;
    grid-row-end:4;
}
```



```
.item1 {
    grid-column-start: span 2;
    grid-row-start:span 3;
}
```



3.4. Tạo bố cục grid với grid-template-areas và grid-erea

Bạn có thể sử dụng thuộc tính **grid-template-areas** để tạo nên bố cục cho grid bằng cách sử dụng tên của các phần tử để định nghĩa các **Grid Area**. Định nghĩa của mỗi **Grid Area** được đặt trong cặp nháy đơn hoặc nháy kép.

Tên của các phần tử trong grid được định nghĩa thông qua thuộc tính **grid-area** của mỗi phần tử.

Ví dụ: Giả sử chúng ta cần tạo grid với bố cục bao gồm các phần tử có tên là header, left, main, right và footer như minh họa dưới đây:

header	header	header	header	header
left	main	main	main	right
left	footer	footer	footer	footer

```
<!DOCTYPE html>
<html>
<head>
    <style>
        /* Sử dụng thuộc tính grid-area để đặt tên cho các phần tử */
        .header {
            grid-area: header;
        }
        .left-content {
            grid-area: left;
        }
        .main-content {
            grid-area: main;
        }
```

```
        .right-content {
            grid-area: right;
        }
        .footer {
            grid-area: footer;
        }
        /* Sử dụng thuộc tính grid-template-areas để bố trí các phần tử */
        .grid-container {
            display: grid;
            grid-template-areas:
                    'header header header header header'
                    'left main main main right'
                    'left footer footer footer footer';
            gap:5px;
            background-color: #2196F3;
            padding: 10px;
        }
        .grid-container > div {
            background-color: #D3EAFC;
            text-align: center;
            padding:10px;
        }
    </style>
</head>
<body>
    <div class="grid-container">
        <div class="header">HEADER</div>
        <div class="left-content">LEFT</div>
        <div class="main-content">MAIN</div>
        <div class="right-content">RIGHT</div>
        <div class="footer">FOOTER</div>
    </div>
</body>
</html>
```