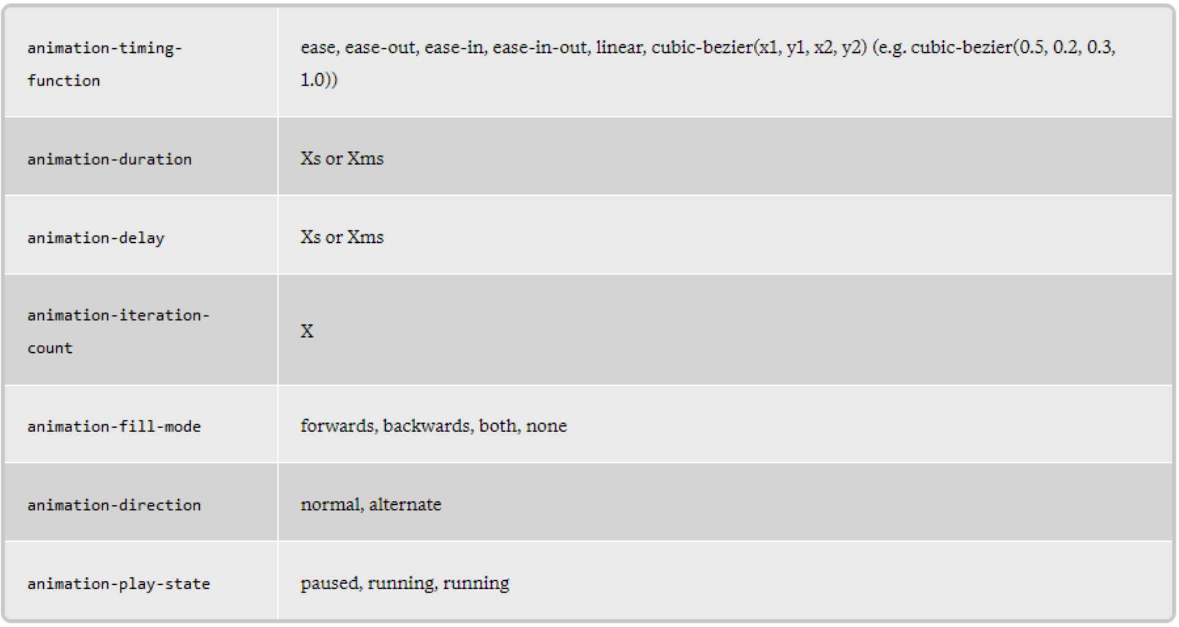
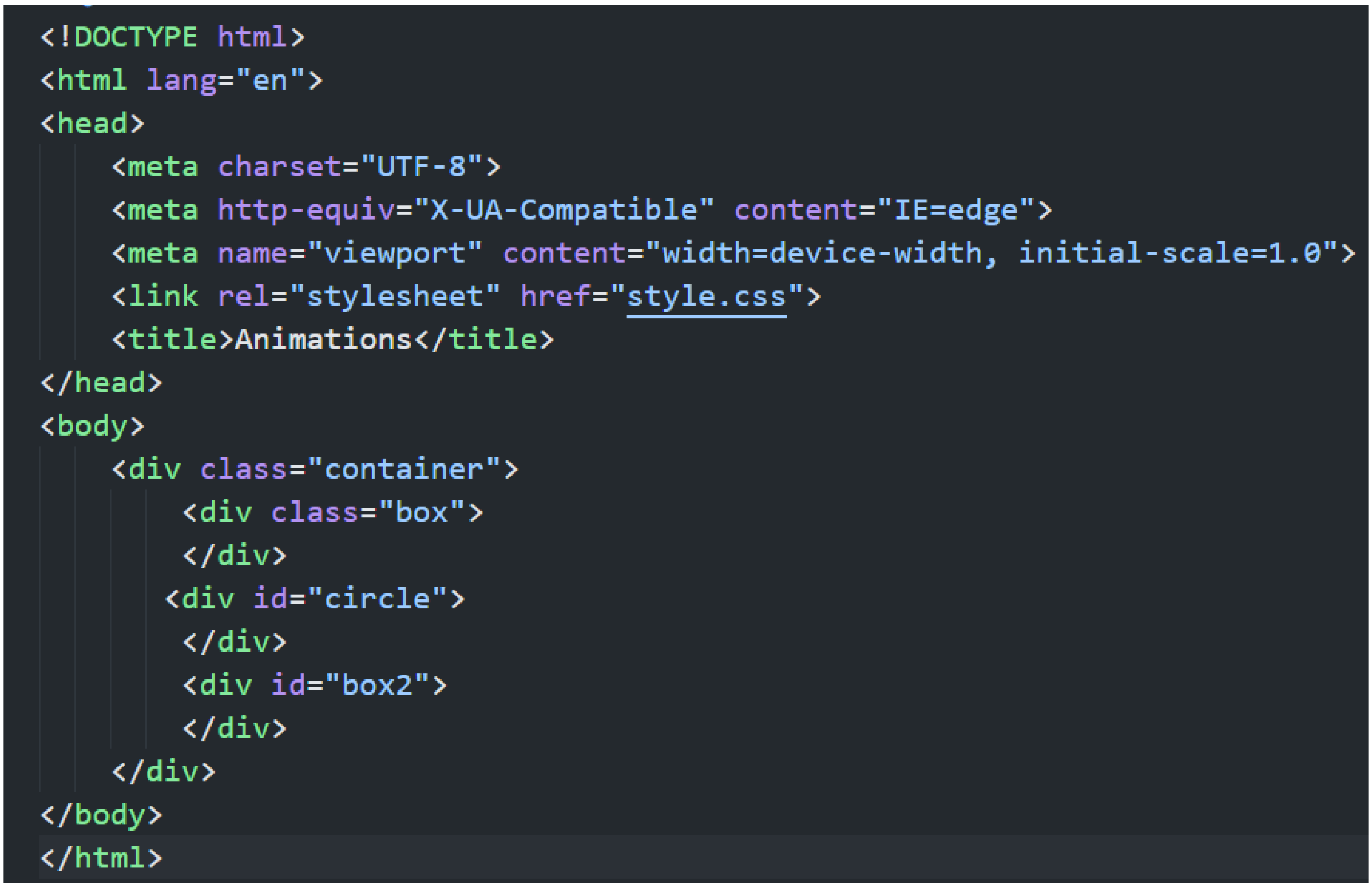
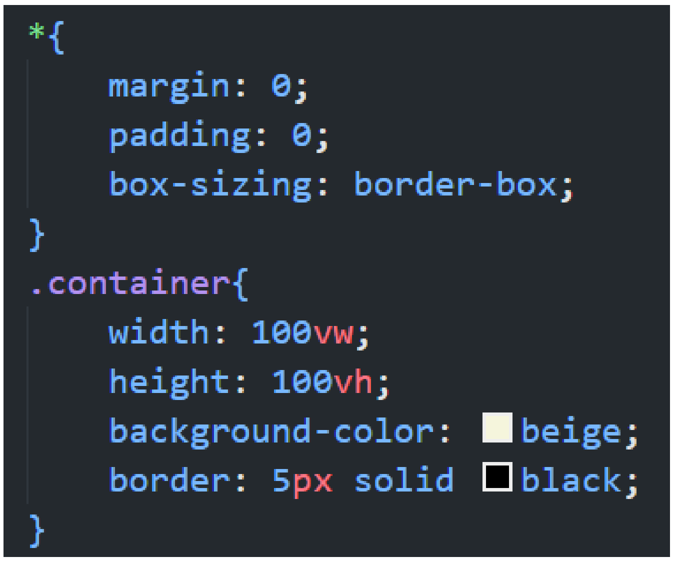


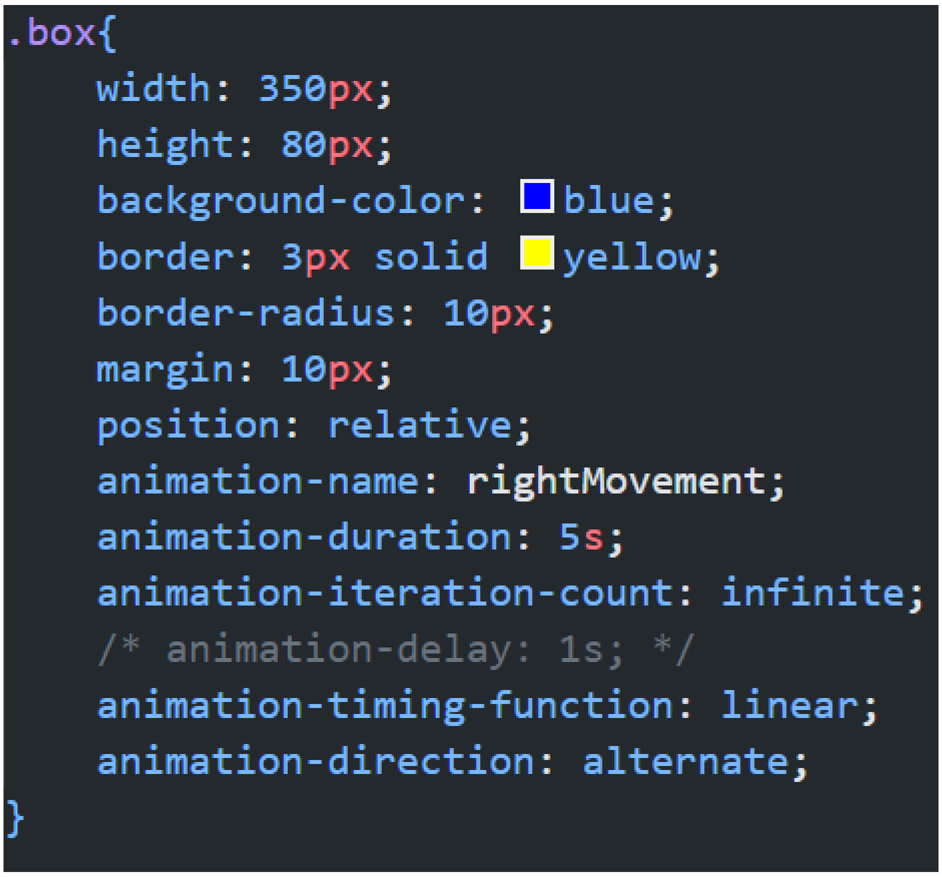
animation-timing-function	ease, ease-out, ease-in, ease-in-out, linear, cubic-bezier(x1, y1, x2, y2) (e.g. cubic-bezier(0.5, 0.2, 0.3, 1.0))
animation-duration	Xs or Xms
animation-delay	Xs or Xms
animation-iteration-count	X
animation-fill-mode	forwards, backwards, both, none
animation-direction	normal, alternate
animation-play-state	paused, running, running



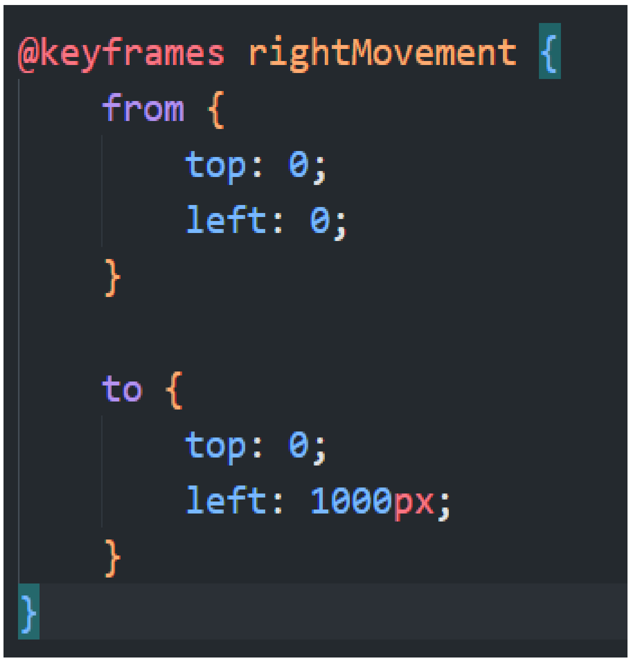
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>Animations</title>
</head>
<body>
    <div class="container">
        <div class="box">
        </div>
       <div id="circle">
        </div>
        <div id="box2">
        </div>
    </div>
</body>
</html>
```



```css
*{
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}
.container{
    width: 100vw;
    height: 100vh;
    background-color: beige;
    border: 5px solid black;
}
```



```css
.box{
    width: 350px;
    height: 80px;
    background-color: blue;
    border: 3px solid yellow;
    border-radius: 10px;
    margin: 10px;
    position: relative;
    animation-name: rightMovement;
    animation-duration: 5s;
    animation-iteration-count: infinite;
    /* animation-delay: 1s; */
    animation-timing-function: linear;
    animation-direction: alternate;
}
```



```css
@keyframes rightMovement {
    from {
        top: 0;
        left: 0;
    }

    to {
        top: 0;
        left: 1000px;
    }
}
```

Media Queries =>

Its is use for making our site responsive. Means on different different display size our website look similar and UI will not break. Media query is a tool and by the help of media query we can define on which display size what would be the arrangement and style of our element. By the help of media query we can apply particular css on the particular size of display.

Media queries allow you to apply CSS styles depending on a device's general type (such as print vs. screen) or other characteristics such as screen resolution or browser viewport width. Media queries are used for the following:

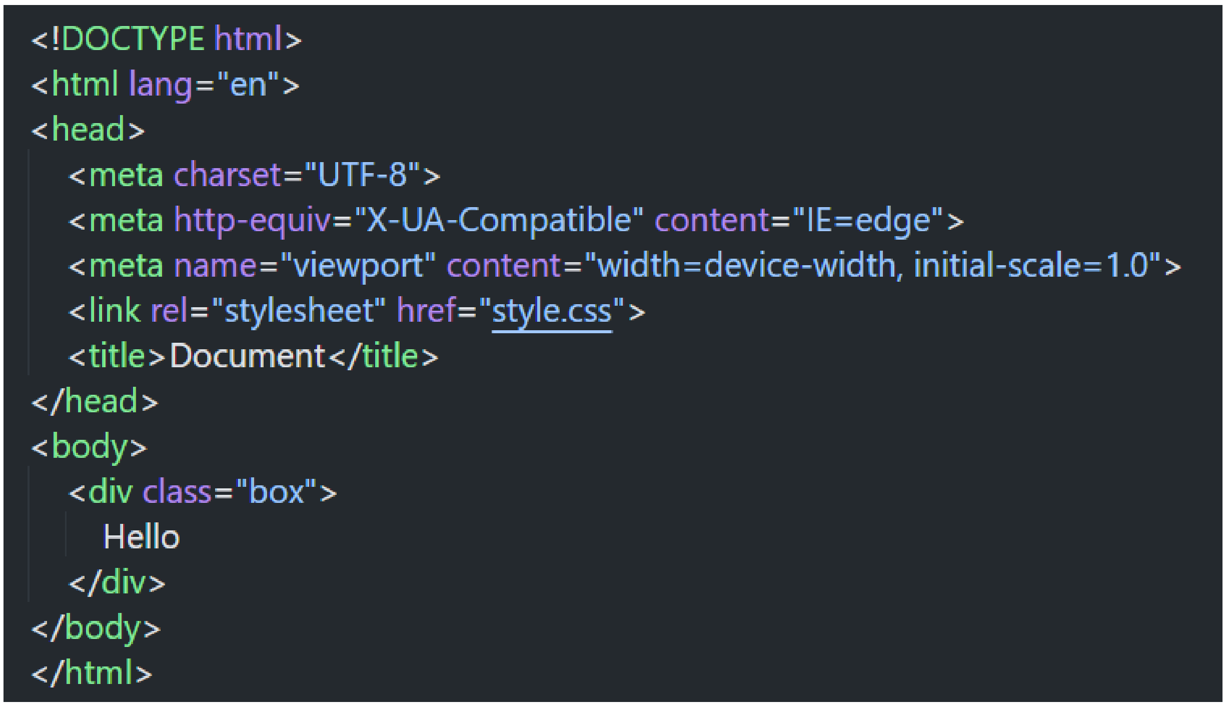
To conditionally apply styles with the CSS @media and @import at-rules.

To target specific media for the <style>, <link>, <source>, and other HTML elements with the media= attribute.

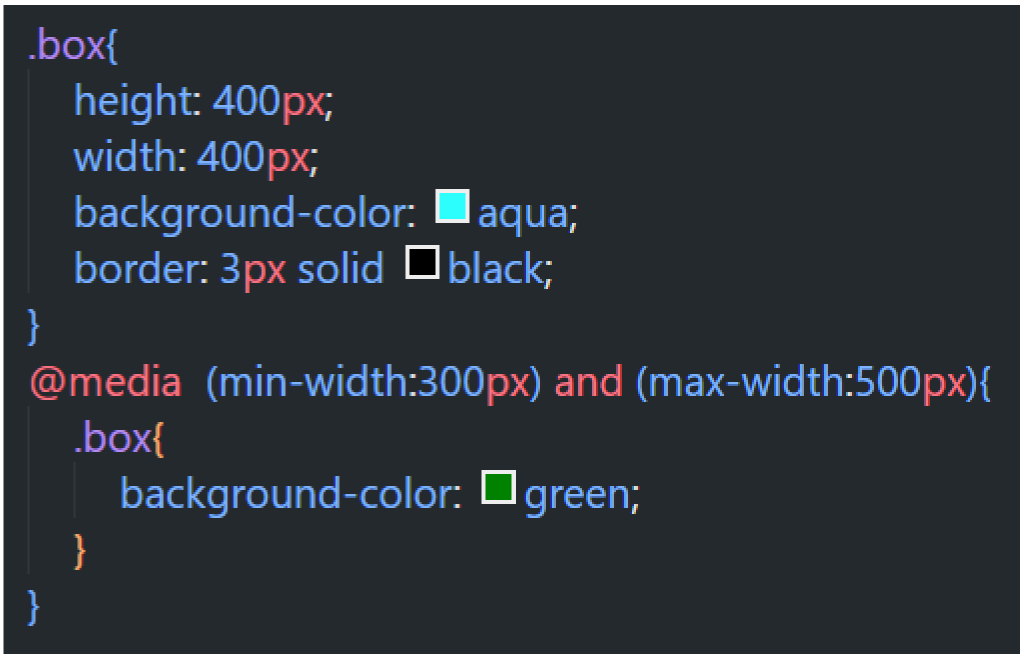
To test and monitor media states using the Window.matchMedia() and EventTarget.addEventListener() methods.

```
@media (max-width: 1250px) {
  /* … */
}
```

```
@media (min-width: 30em) and (orientation: landscape) {
  /* … */
}
```



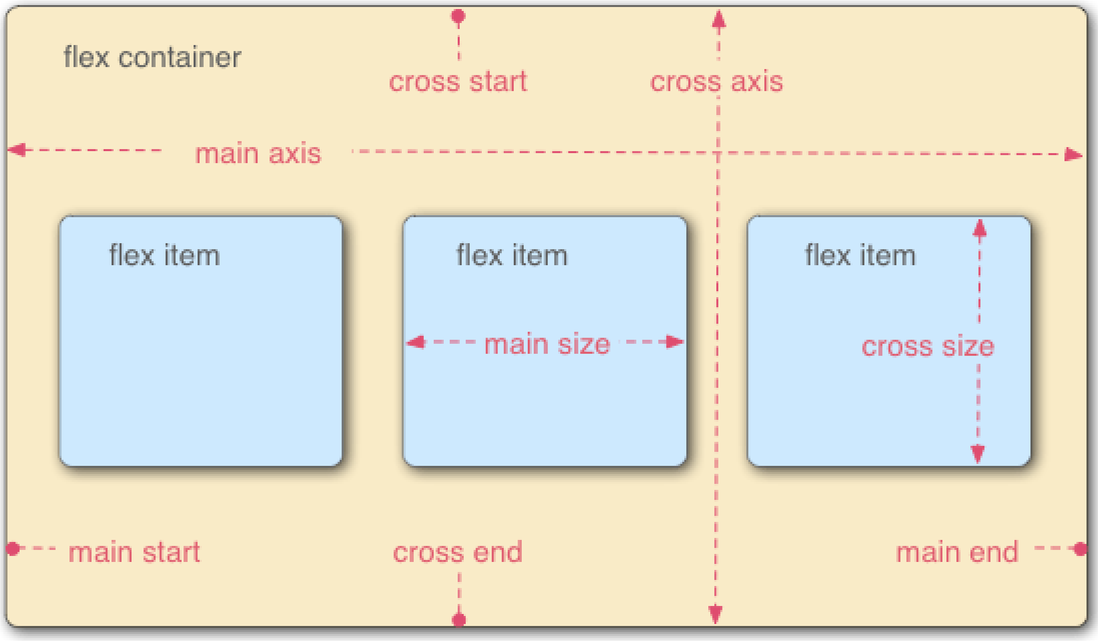
```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>Document</title>
</head>
<body>
    <div class="box">
        Hello
    </div>
</body>
</html>
```



```css
.box{
    height: 400px;
    width: 400px;
    background-color: aqua;
    border: 3px solid black;
}
@media  (min-width:300px) and (max-width:500px){
    .box{
        background-color: green;
    }
}
```

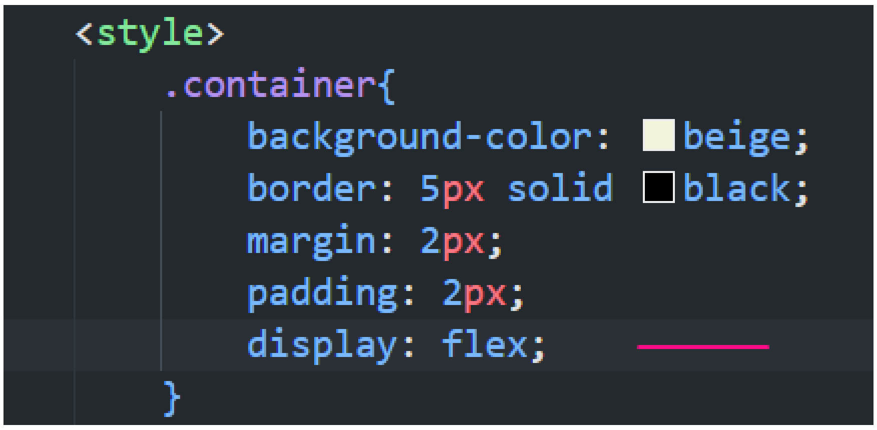
Flexbox =>

It is use for layout in row and column. Flex box and grid. Flex box means flexible box

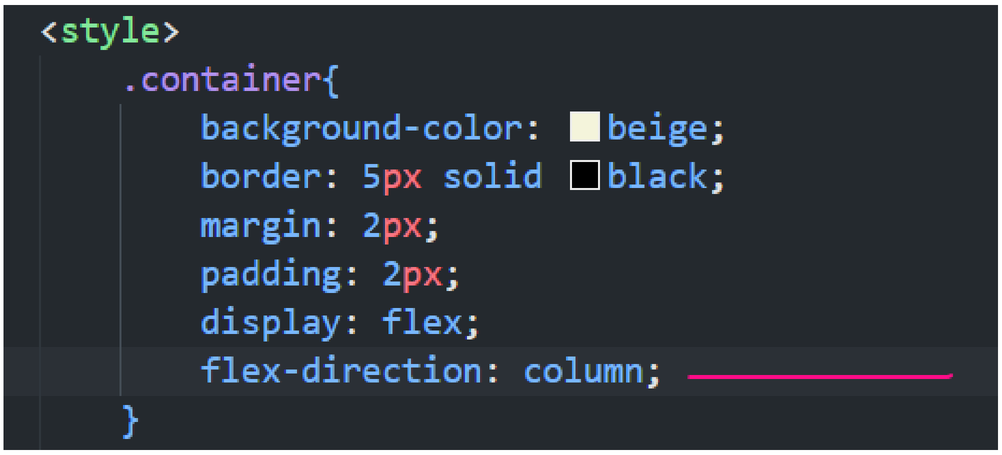


For applying flex box we have to make its display property as flex by doing this that container become flex container and the items which are present inside this container become flex item.

By default the direction of flex s row.



```
<style>
    .container{
        background-color: beige;
        border: 5px solid black;
        margin: 2px;
        padding: 2px;
        display: flex;
    }
```

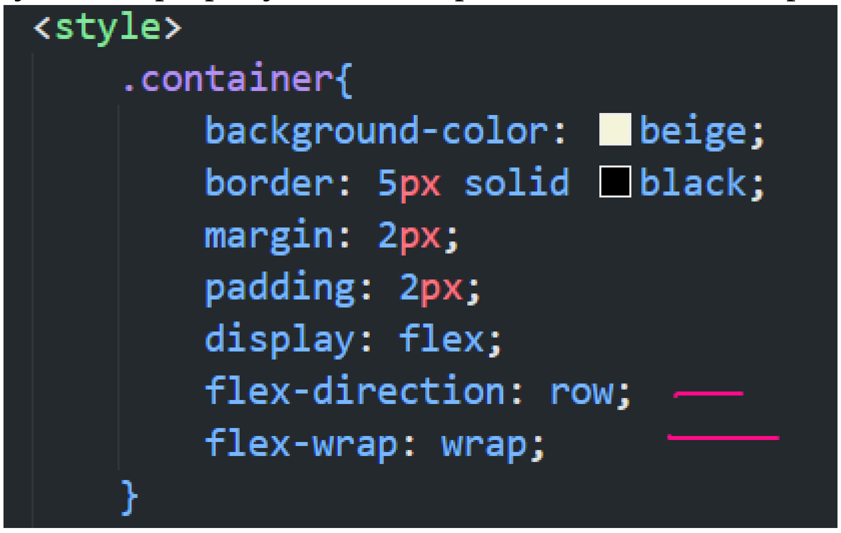


```
<style>
    .container{
        background-color: beige;
        border: 5px solid black;
        margin: 2px;
        padding: 2px;
        display: flex;
        flex-direction: column;
    }
```

flex-direction: row; flex-direction: row-reverse; flex-direction: column-reverse;

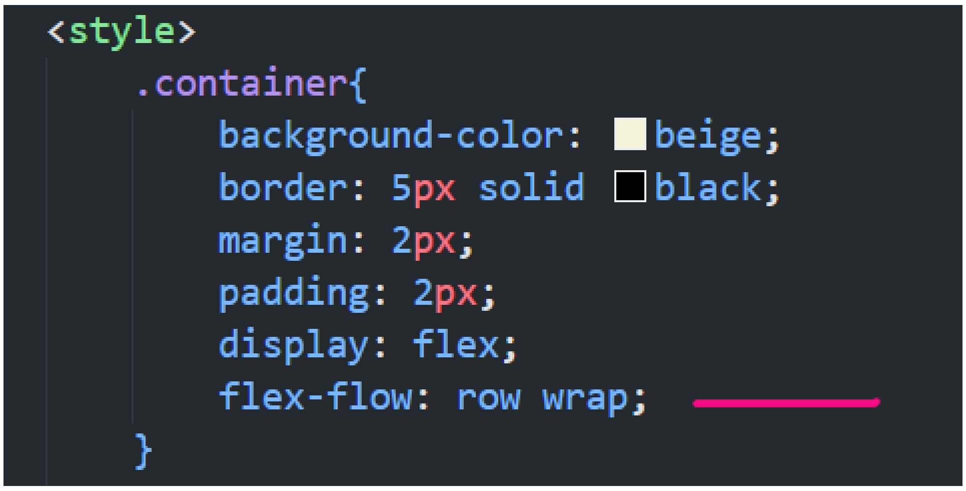
Flex wrap=>

This will wrap our contant till we have space but when we don't have space then the contant will squeeze. Its by default property is now wrap. We can also do wrap-reverse.



```
<style>
    .container{
        background-color: beige;
        border: 5px solid black;
        margin: 2px;
        padding: 2px;
        display: flex;
        flex-direction: row;
        flex-wrap: wrap;
    }
```

Flex-flow is shorthand of flex warp and flex flow



```
<style>
    .container{
        background-color: beige;
        border: 5px solid black;
        margin: 2px;
        padding: 2px;
        display: flex;
        flex-flow: row wrap;
    }
```

Justify content =>

By using this property we can place the content according to the main axis.

If we have flex direction column in such case main axis become vertical.

If we have flex direction row in such case main axis become horizontal.

justify-content: start;   justify-content: end; justify-content: space-between; in this our first and last box is touch with flex container and rest of the boxes are having space in between

justify-content: space-around; by this we are having equal space between each box from left and right.

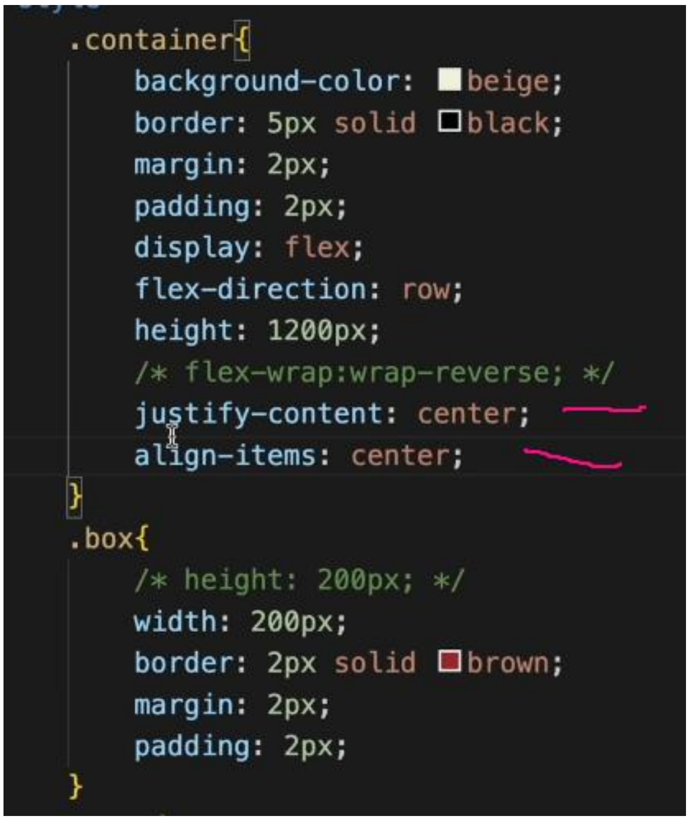
justify-content: space-evenly; by this we have even space between all the boxes.

Align item =>

It will work according to our cross axis and by default we have cross axis vertical.

align-items: start; align-items: end;

By default item always stretch



```
.container{
    background-color: beige;
    border: 5px solid black;
    margin: 2px;
    padding: 2px;
    display: flex;
    flex-direction: row;
    height: 1200px;
    /* flex-wrap:wrap-reverse; */
    justify-content: center;
    align-items: center;
}
.box{
    /* height: 200px; */
    width: 200px;
    border: 2px solid brown;
    margin: 2px;
    padding: 2px;
}
```

align-content:end, align-content:center, align-content:space-between, align-content:space-around, align-content:stretch,

property for flex item =>

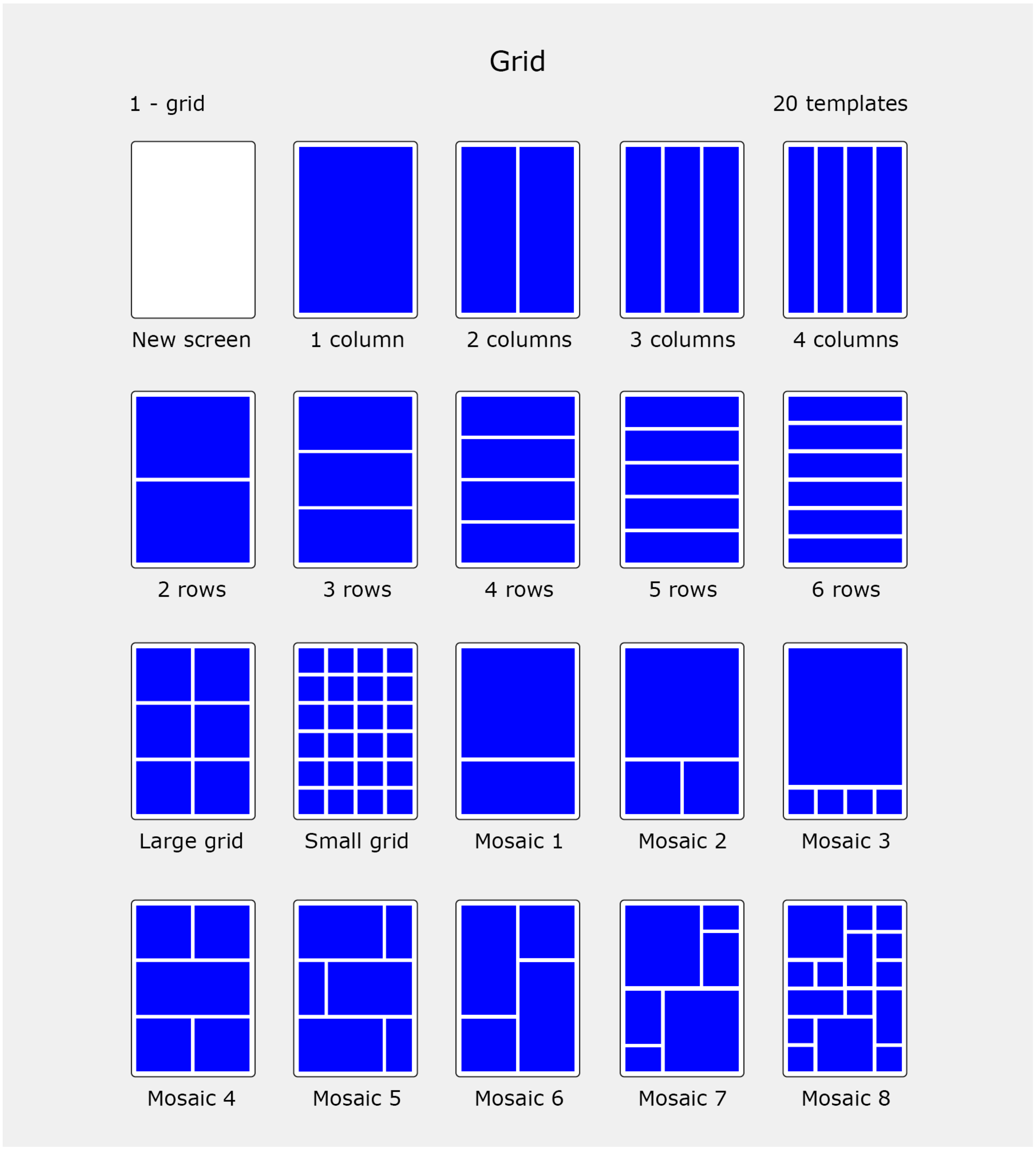
Order

Flex shrink

Flex grow

Flex basis

Grid =>


Grid
1 - grid
20 templates
New screen
1 column
2 columns
3 columns
4 columns
2 rows
3 rows
4 rows
5 rows
6 rows
Large grid
Small grid
Mosaic 1
Mosaic 2
Mosaic 3
Mosaic 4
Mosaic 5
Mosaic 6
Mosaic 7
Mosaic 8

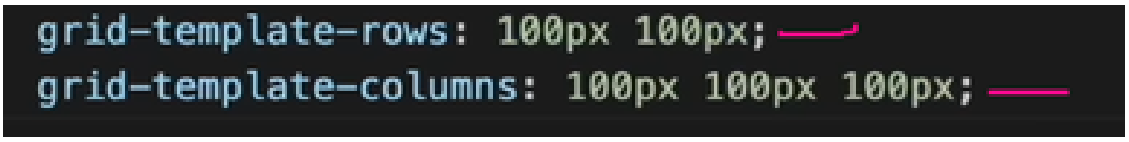
display

Defines the element as a grid container and establishes a new grid formatting context for its contents.

Values:

- **grid** – generates a block-level grid
- **inline-grid** – generates an inline-level grid

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



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

grid-template-columns
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.

But you can choose to explicitly name the lines. Note the bracket syntax for the line names:

```
.container {
  grid-template-columns: [first] 40px [line2] 50px [line3] auto [col4-start] 50px [five] 40px [end];
  grid-template-rows: [row1-start] 25% [row1-end] 100px [third-line] auto [last-line];
}
```

If multiple lines share the same name, they can be referenced by their line name and count.

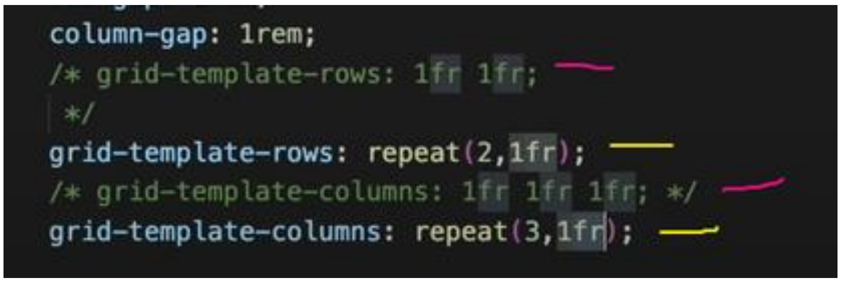
```
.item {
  grid-column-start: col-start 2;
}
```

The fr unit allows you to set the size of a track as a fraction of the free space of the grid container. For example, this will set each item to one third the width of the grid container:

```
.container {
  grid-template-columns: 1fr 1fr 1fr;
}
```

The free space is calculated *after* any non-flexible items. In this example the total amount of free space available to the fr units doesn't include the 50px:

```
.container {
  grid-template-columns: 1fr 50px 1fr 1fr;
}
```


```
column-gap: 1rem;
/* grid-template-rows: 1fr 1fr;
 */
grid-template-rows: repeat(2,1fr);
/* grid-template-columns: 1fr 1fr 1fr; */
grid-template-columns: repeat(3,1fr);
```

column-gap
row-gap
grid-column-gap
grid-row-gap

Specifies the size of the grid lines. You can think of it like setting the width of the gutters between the columns/rows.

Values:

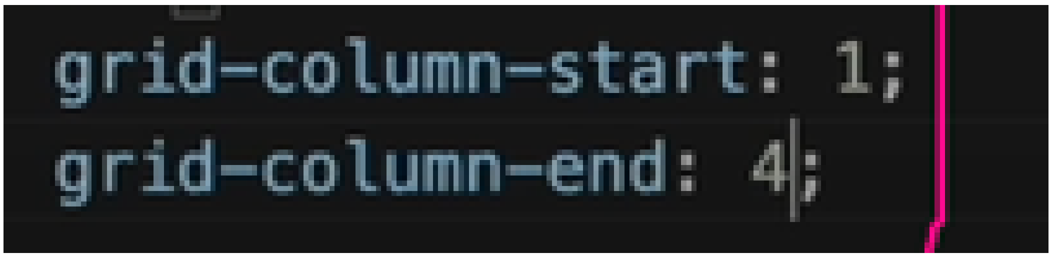
- **<line-size>** – a length value

```
.container {
  /* standard */
  column-gap: <line-size>;
  row-gap: <line-size>;

  /* old */
  grid-column-gap: <line-size>;
  grid-row-gap: <line-size>;
}
```

Example:

```
.container {
  grid-template-columns: 100px 50px 100px;
  grid-template-rows: 80px auto 80px;
  column-gap: 10px;
  row-gap: 15px;
}
```


```
grid-column-start: 1;
grid-column-end: 4;
```

justify-items

Aligns grid items along the *inline (row)* axis (as opposed to align-items which aligns along the *block (column)* axis). This value applies to all grid items inside the container.

Values:

- **start** – aligns items to be flush with the start edge of their cell
- **end** – aligns items to be flush with the end edge of their cell
- **center** – aligns items in the center of their cell
- **stretch** – fills the whole width of the cell (this is the default)

```
.container {
  justify-items: start | end | center | stretch;
}
```

Examples:

```
.container {
  justify-items: start;
}

.container {
  justify-items: end;
}

.container {
  justify-items: center;
}

.container {
  justify-items: stretch;
}
```

This behavior can also be set on individual grid items via the justify-self property.

align-items

Aligns grid items along the *block (column)* axis (as opposed to justify-items which aligns along the *inline (row)* axis). This value applies to all grid items inside the container.

Values:

- **stretch** – fills the whole height of the cell (this is the default)
- **start** – aligns items to be flush with the start edge of their cell
- **end** – aligns items to be flush with the end edge of their cell
- **center** – aligns items in the center of their cell

- **baseline** – align items along text baseline. There are modifiers to baseline — first baseline and last baseline which will use the baseline from the first or last line in the case of multi-line text.

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

Examples:

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

.container {
  align-items: end;
}

.container {
  align-items: center;
}

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

This behavior can also be set on individual grid items via the align-self property.

There are also modifier keywords safe and unsafe (usage is like align-items: safe end). The safe keyword means “try to align like this, but not if it means aligning an item such that it moves into inaccessible overflow area”, while unsafe will allow moving content into inaccessible areas (“data loss”).

place-items

place-items sets both the align-items and justify-items properties in a single declaration.

Values:

- **<align-items> / <justify-items>** – The first value sets align-items, the second value justify-items. If the second value is omitted, the first value is assigned to both properties.

For more details, see align-items and justify-items.

This can be very useful for super quick multi-directional centering:

```
.center {
  display: grid;
  place-items: center;

}
```

justify-content

Sometimes the total size of your grid might be less than the size of its grid container. This could happen if all of your grid items are sized with non-flexible units like px. In this case you can set the alignment of the grid within the grid container. This property aligns the grid along the *inline (row)* axis (as opposed to align-content which aligns the grid along the *block (column)* axis).

Values:

- **start** – aligns the grid to be flush with the start edge of the grid container
- **end** – aligns the grid to be flush with the end edge of the grid container
- **center** – aligns the grid in the center of the grid container
- **stretch** – resizes the grid items to allow the grid to fill the full width of the grid container
- **space-around** – places an even amount of space between each grid item, with half-sized spaces on the far ends
- **space-between** – places an even amount of space between each grid item, with no space at the far ends
- **space-evenly** – places an even amount of space between each grid item, including the far ends

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

align-content

Sometimes the total size of your grid might be less than the size of its grid container. This could happen if all of your grid items are sized with non-flexible units like px. In this case you can set the alignment of the grid within the grid container. This property aligns the grid along the *block (column)* axis (as opposed to justify-content which aligns the grid along the *inline (row)* axis).

Values:

- **start** – aligns the grid to be flush with the start edge of the grid container
- **end** – aligns the grid to be flush with the end edge of the grid container
- **center** – aligns the grid in the center of the grid container
- **stretch** – resizes the grid items to allow the grid to fill the full height of the grid container
- **space-around** – places an even amount of space between each grid item, with half-sized spaces on the far ends

- **space-between** – places an even amount of space between each grid item, with no space at the far ends
- **space-evenly** – places an even amount of space between each grid item, including the far ends

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

justify-self

Aligns a grid item inside a cell along the *inline (row)* axis (as opposed to align-self which aligns along the *block (column)* axis). This value applies to a grid item inside a single cell.

Values:

- **start** – aligns the grid item to be flush with the start edge of the cell
- **end** – aligns the grid item to be flush with the end edge of the cell
- **center** – aligns the grid item in the center of the cell
- **stretch** – fills the whole width of the cell (this is the default)

```
.item {
  justify-self: start | end | center | stretch;
}
```

Examples:

```
.item-a {
  justify-self: start;
}

.item-a {
  justify-self: end;
}

.item-a {
  justify-self: center;
}

.item-a {
  justify-self: stretch;
}
```

To set alignment for *all* the items in a grid, this behavior can also be set on the grid container via the justify-items property.

align-self

Aligns a grid item inside a cell along the *block (column)* axis (as opposed to justify-self which aligns along the *inline (row)* axis). This value applies to the content inside a single grid item.

Values:

- **start** – aligns the grid item to be flush with the start edge of the cell
- **end** – aligns the grid item to be flush with the end edge of the cell
- **center** – aligns the grid item in the center of the cell
- **stretch** – fills the whole height of the cell (this is the default)

place-self

place-self sets both the align-self and justify-self properties in a single declaration.

Values:

- **auto** – The "default" alignment for the layout mode.
- **<align-self> / <justify-self>** – The first value sets align-self, the second value justify-self. If the second value is omitted, the first value is assigned to both properties.

The repeat() Function and Keywords

The repeat() function can save some typing:

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

/* easier: */
grid-template-columns:
  repeat(8, 1fr);

/* especially when: */
grid-template-columns:
  repeat(8, minmax(10px, 1fr));
```

But repeat() can get extra fancy when combined with keywords:

- auto-fill: Fit as many possible columns as possible on a row, even if they are empty.
- auto-fit: Fit whatever columns there are into the space. Prefer expanding columns to fill space rather than empty columns.

This bears the most famous snippet in all of CSS Grid and one of the all-time great CSS tricks:

```
grid-template-columns:
  repeat(auto-fit, minmax(250px, 1fr));
```

justify-items

Aligns grid items along the *inline (row)* axis (as opposed to align-items which aligns along the *block (column)* axis). This value applies to all grid items inside the container.

Values:

- **start** – aligns items to be flush with the start edge of their cell
- **end** – aligns items to be flush with the end edge of their cell
- **center** – aligns items in the center of their cell
- **stretch** – fills the whole width of the cell (this is the default)

```
.container {
  justify-items: start | end | center | stretch;
}
```