This chapter includes **multiple columns**, **styled images**, and **gradients** for better visual presentation. Various **transformations, transitions, and animations** are applied to create interactive and dynamic effects. Additionally, **CSS variables** and **media queries** are used to make designs flexible and responsive across different devices.

CHAPTER 5

Chapter-5 CSS Effects, & Responsive Design

CSS Multiple Column

- ✓ The CSS multi-column layout allows easy definition of multiple columns of text - just like in newspapers:
- ✓ Properties are:
  - **column-count:** Specifies the number of columns an element should be divided into
  - **column-gap:** Specifies the gap between the columns
  - **column-rule-style:** Specifies the style of the rule between columns. (none, dotted, dashed, solid, double)
  - **column-rule-color:** Specifies the color of the rule between columns
  - **column-rule-width:** Specifies the width of the rule between columns

Shorthand:
column-rule: width style color
example : column-rule: 1px dotted red

column-rule-style defaults to none, so if you don't include a style, the rule won't render even if width and color are given.

Example:

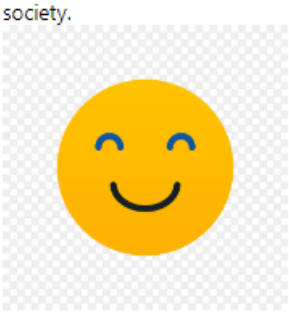
```
<html>
<head>
<style>
.news
{
column-count:4;
column-gap:20px;
column-rule-style:dashed;
column-rule-color:rgb(219, 12, 47);
column-rule-width:7px;
text-align: justify;
}
</style>
</head>
<body>
<h1 align="center">CSS MULTIPLE COLUMS</h1>
<div class="news">
LJ Group of Institutes is managed by Lok Jagruti Kendra (LJK), a Charitable Trust and a Registered Society established in1980 by eminent academicians and visionaries like Prof. B.M. Peerzada, former Dean of Commerce Faculty,Gujarat University, Padma Bhushan Lord Meghnad Desai (L ondon School of Economics ), Prof. Gautam Appa (LSE), Late Prof. M.S. Trivedi, former Vice Chancellor, South Gujarat University, renowned jurist Late Shri Girishbhai Patel and Shri Subodhbhai Shah. It was envisioned as "a key player in education and social development by promoting and nurturing creativity, scholarship, innovation and excellence through a chain of quality institutes." LJK's mission has been "to establish and manage institutions with an environment in which new ideas, delivery strategies and scholarship flourish and from where leaders and innovators of tomorrow shall emerge."
LJK runs 32 institutions on two well-developed environment-friendly campuses with dedicated buildings and infrastructure facilities, and offering various diploma, undergraduate and postgraduate programmes
duly approved by respective apex bodies. LJ Institutes have over 21000 students and 1000+ faculties engaged in teaching-learning, research and extension activities and striving to develop students into complete citizens not only having the necessary subject knowledge and skills, but also the empathy towards various environmental, social, cultural and other issues affecting the society.
<img src="smiley.png">
```

```
</div>
</body>
</html>
```

Output

CSS MULTIPLE COLUMS

LJ Group of Institutes is managed by Lok Jagruti Kendra (LJK), a Charitable Trust and a Registered Society established in1980 by eminent academicians and visionaries like Prof. B.M. Peerzada, former Dean of Commerce Faculty,Gujarat University, Padma Bhushan Lord Meghnad Desai (L ondon School of Economics), Prof. Gautam Appa (LSE), Late Prof. M.S. Trivedi, former Vice Chancellor, South Gujarat University, renowned jurist Late Shri Girishbhai Patel and Shri Subodhbhai Shah. It was envisioned as "a key player in education and social development by promoting and nurturing creativity, scholarship, innovation and excellence through a chain of quality institutes." LJK's mission has been "to establish and manage institutions with an environment in which new ideas, delivery strategies and scholarship flourish and from where leaders and innovators of tomorrow shall emerge." LJK runs 32 institutions on two well-developed environment-friendly campuses with dedicated buildings and infrastructure facilities, and offering various diploma, undergraduate and postgraduate programmes duly approved by respective apex bodies. LJ Institutes have over 21000 students and 1000+ faculties engaged in teaching-learning, research and extension activities and striving to develop students into complete citizens not only having the necessary subject knowledge and skills, but also the empathy towards various environmental, social, cultural and other issues affecting the society.



CSS style images

- ✓ The styling of an image in CSS is similar to the styling of an element by using the borders, margins, paddings etc.
- ✓ There are multiple CSS properties such as border property, height property, width property, etc. that helps us to style an image.

Rounded image

- ✓ The border-radius property sets the radius of the bordered image.
- ✓ It is used to create the rounded images.

Property	Syntax	Example	What it does
border-radius	border-radius:radius;	border-radius: 20px;	Sets the corner rounding for **all four corners**.
border-top-left-radius	border-top-left-radius: radius;	border-top-left-radius: 15px;	Rounds only the **top-left** corner.
border-top-right-radius	border-top-right-radius;	border-top-right-radius: 15px;	Rounds only the **top-right** corner.
border-bottom-right-radius	border-bottom-right-radius: radius;	border-bottom-right-radius: 15px;	Rounds only the **bottom-right** corner.
border-bottom-left-radius	border-bottom-left-radius: radius;	border-bottom-left-radius: 15px;	Rounds only the **bottom-left** corner.

Example

```
<html>
  <head><style>
    #img1{
       border: 5px solid red;
       border-radius:10px;
       padding:20px;
       width: 200px;
       background-color: aqua;
    }
    #img2{
       border: 2px solid blue;
       border-radius:50%;
       width:200px;
       padding:5px;
    }
  </style> </head>
  <body>
    <h1>Rounded Image</h1>
    <img src="a.jpg" id="img1"></img>
    <h1>Circle Image</h1>
    <img src="smiley.png" id="img2"></img>
  </body>
</html>
```

Rounded Image

Circle Image

Thumbnail Image

The border property and padding property are used to make a thumbnail image.

Example

```
<html> <head>
   <style>
   img{ border: 2px solid blue;
      border-radius:8px;
      padding:20px;
      width: 300px;}
   h2{ color:red; }
   </style> </head>
<body>
   <h1>Thumbnail Image</h1>
   <img src="smiley.png"></img>
</body>
</html>
```

Thumbnail Image

Transparent image

- ✓ To make an image transparent, we have to use the opacity property. The value of this property lies between 0.0 to 1.0.

Example

```
<html> <head> <style>
img{
   border: 2px solid red;
   border-radius:5px;
   opacity:0.3; }
h2{ color:red; }
</style> </head>
<body>
   <h1>Transparent Image</h1>
   <img src="a.jpg"></img>
</body>
</html>
```

Transparent Image

Responsive Image

- ✓ It automatically adjusts to fit on the screen size.
- ✓ It is used to adjust the image to the specified box automatically.

Example

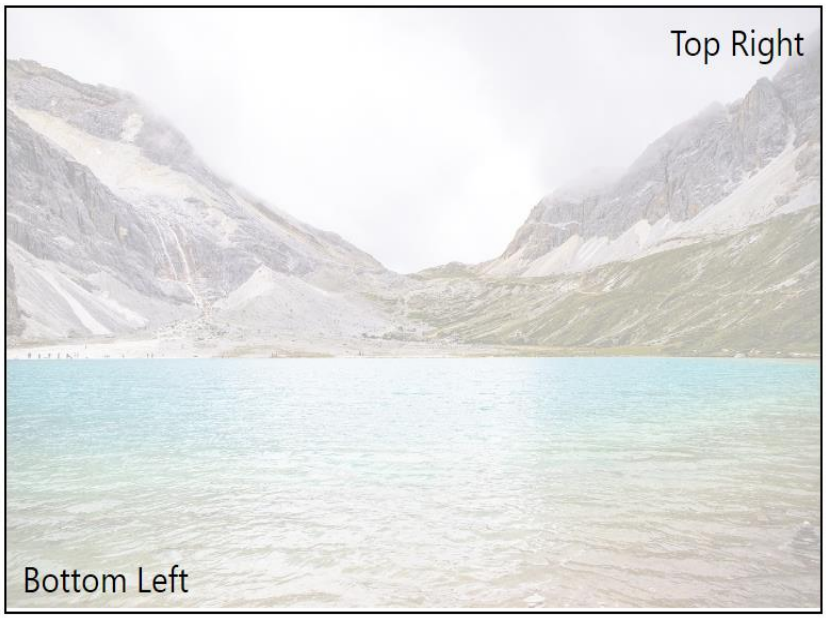
```
<html> <head> <style>
#img1{ max-width:100%; height:auto; }
</style>
</head>
<body>
   <h1>Responsive image</h1>
   <h2>You can resize the window to see the effect</h2>
   <img src="a1.jpg" id="img1"></img>
   <img src="a2.jpg" id="img2"></img>
</body> </html>
```

Example:

Write down HTML/CSS code that add some text to an image in the bottom left corner and in top right corner. Also, use opacity property of image to make an image transparent.

```
<html> <head> <style>
.container { position: relative; border: 2px solid;}
.bottomleft { position: absolute; bottom: 8px; left: 15px; font-size: 30px; }
.topright { position: absolute; top: 8px; right: 15px; font-size: 30px; }
img{
   width:100%; height:500px;opacity:0.3;
}
 </style> </head> <body>
   <h2>Transparent image and position property example</h2>
   <div class="container">
      <img src="a2.jpg" alt="Hello">
      <div class="bottomleft">Bottom Left</div>
      <div class="topright">Top Right</div>
   </div>
</body> </html>
```

Transparent image and position property example



CSS 2D Transforms

CSS transforms allow you to move, rotate, scale, and skew elements. Mouse over the element below to see a 2D transformation:

With the CSS transform property you can use the following 2D transformation methods:

- translate()
- rotate()
- scale()
- skew()

The listed CSS transform functions allow you to apply transformations to HTML elements, such as translating, rotating, scaling, and skewing. Here's a concise explanation for each:

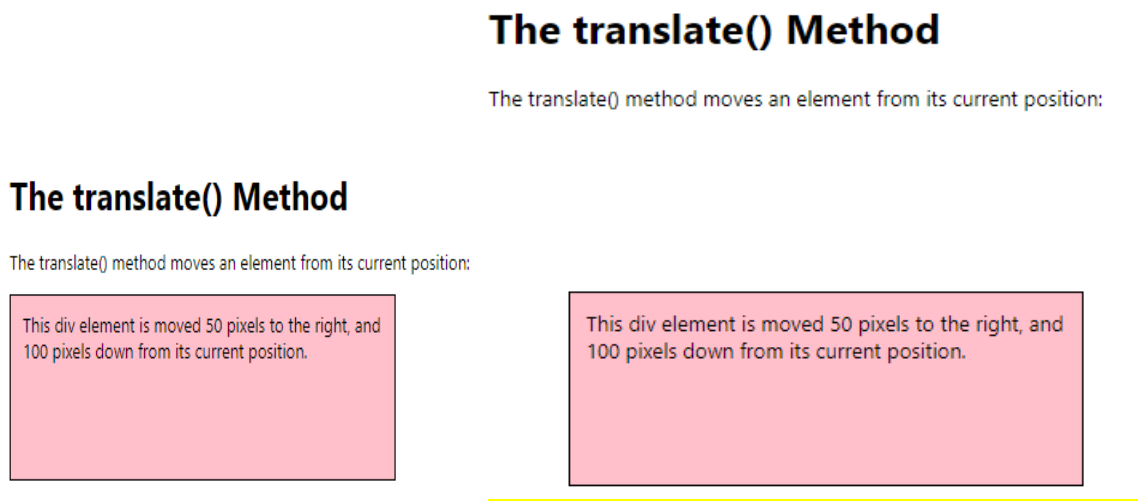
1. **translate(x, y)**:
   Moves an element horizontally by x and vertically by y.
2. **translateX(n)**:
   Moves an element horizontally by n units.
3. **translateY(n)**:
   Moves an element vertically by n units.
4. **rotate(angle)**:
   Rotates an element by the specified angle (in degrees, radians, etc.).
5. **scale(x, y)**:
   Resizes an element horizontally by x and vertically by y. Values greater than 1 increase size; values between 0 and 1 decrease it.
6. **scaleX(n)**:
   Resizes an element's width by n.
7. **scaleY(n)**:
   Resizes an element's height by n.
8. **skew(x, y)**:
   Distorts an element by slanting it horizontally by x and vertically by y.
9. **skewX(angle)**:
   Skews an element horizontally by the specified angle.
10. **skewY(angle)**:
    Skews an element vertically by the specified angle.

translate()

- ✓ The translate() method moves an element from its current position (according to the parameters given for the X-axis and the Y-axis).
- ✓ The following example moves the element 50 pixels to the right, and 100 pixels down from its current position.

EXAMPLE

```
<html><head><style>
   div {
      width: 300px;
      height: 100px;
      background-color: pink;
      border: 1px solid black;
      padding:10px;
      }
   div:hover { transform: translate(50px,100px); }
   </style></head>
<body>
   <h1>The translate() Method</h1>
   <p>The translate() method moves an element from its current position:</p>
   <div> This div element is moved 50 pixels to the right, and 100 pixels down from its current position.
</div>
</body></html>
```



- **translateX(50px):** It moves the element 50px along X-axis.
- **translateY(100px):** It moves the element 100px along Y-axis.

rotate()

- ✓ The rotate() method rotates an element clockwise or counter-clockwise according to a given degree.
- ✓ For counter-clockwise give value in (-)negative.
- ✓ The following example rotates the one image clockwise with 20 degrees and the other image anticlockwise with -40deg.

EXAMPLE

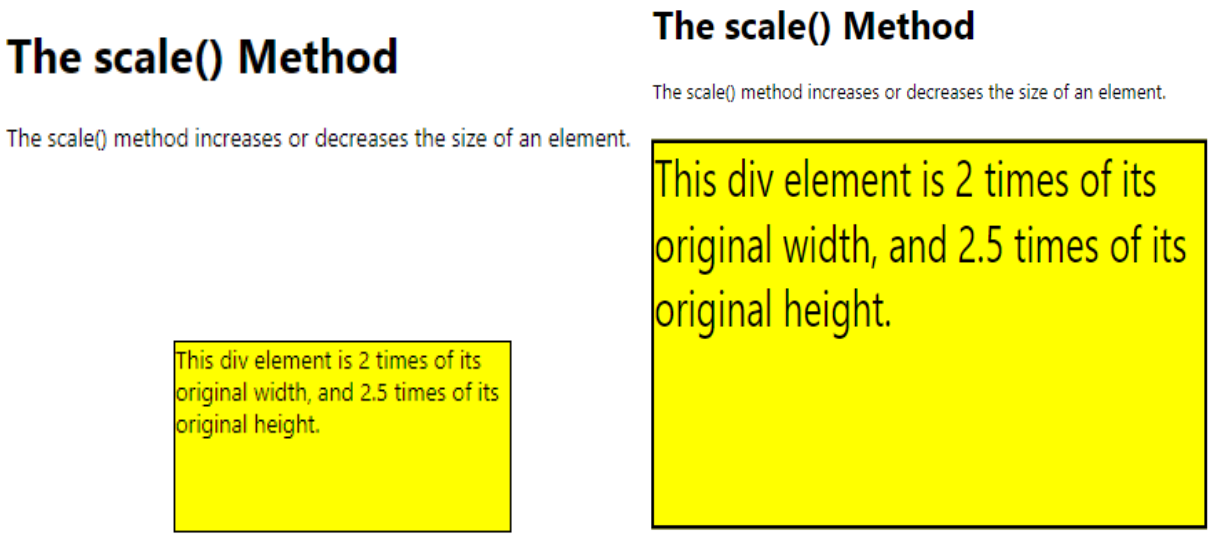
```
<head>
<style>
.i1 {
width: 600px;
height: 300px;
border: 1px solid black;
}
.i1:hover
{ transform: rotate(20deg);
}
.i2 {
width: 600px;
height: 300px;
border: 1px solid black;
}
.i2:hover { transform: rotate(-40deg); }
</style></head>
<body>
<h1>The rotate() Method</h1>
<p>The rotate() method rotates an element clockwise or counter-clockwise.</p>
<img src="a3.jpg" class="i1"/>
<img src="a2.jpg" class="i2"/>
</body>
```

scale()

- ✓ The scale() method increases or decreases the size of an element (according to the parameters given for the width and height).
- ✓ The following example increases the element to be two times of its original width, and three times of its original height:

EXAMPLE

```
<html> <head> <style>
div {
   margin: 100px;
   width: 200px;
   height: 100px;
   background-color:yellow;
   border: 1px solid black; }
   div:hover { transform: scale(2,2.5); }
   </style>
   </head>
<body>
<h1>The scale() Method</h1>
<p>The scale() method increases or decreases the size of an element.</p>
<div> This div element is two times of its original width, and 2.5 times of its original height. </div>
</body>
</html>
```



scaleX()

- ✓ The scaleX() method increases or decreases the width of an element.
- ✓ The following example decreases width of the element by half of its original width:
  - **div { transform: scaleX(0.5); }**
- ✓ The following example increases the element to be 2 times of its original width:
  - **div { transform: scaleX(2); }**

scaleY()

- ✓ The scaleY() method increases or decreases the height of an element.

- ✓ The following example increases the element to be three times of its original height:
  - **div { transform: scaleY(3); }**
- ✓ The following example decreases the element to be half of its original height:
  - **div { transform: scaleY(0.5); }**

How To Flip an Image(Add a mirror effect)

- ✓ The image is flipped to create a mirror image. Flip an image means create the mirror image horizontally or vertically.
  - The transform: scaleX(-1) property is used to flip the image horizontally.
  - The transform property is used to rotate the image and scalex(-1) rotates the image to axial symmetry. Hence the original image is flipped to its mirror image.
  - The transform: scaleX(-2) - double the mirror image horizontally.
  - The transform: scaleY(-2) - double the mirror image vertically.
  - The transform: scaleY(-1) - mirror image vertically as well as horizontally.

Note: The transform: scale(-1) property create a mirror image vertically as well as horizontally.

Example

```
<head>
<style>
.i1:hover {
  transform: scaleY(-1);
}

</style>
</head>
<body>
<h3>Hover over the image below:</h3>
  <div>
     <img src="nature.jpg" alt="hello" style="width:300px;height:200px" class="i1">

   </div>
</body>
```

Before

Hover over the image below:

After scale(-1)

Hover over the image below:

After scaleX(-1)

Hover over the image below:

After scaleY(-1)

Hover over the image below:

skew()

- ✓ The skew() method skews an element along the X and Y-axis by the given angles.
- ✓ The `skewX()` method skews an element along the X-axis by the given angle.
- ✓ The following example skews the element 20 degrees along the X-axis:

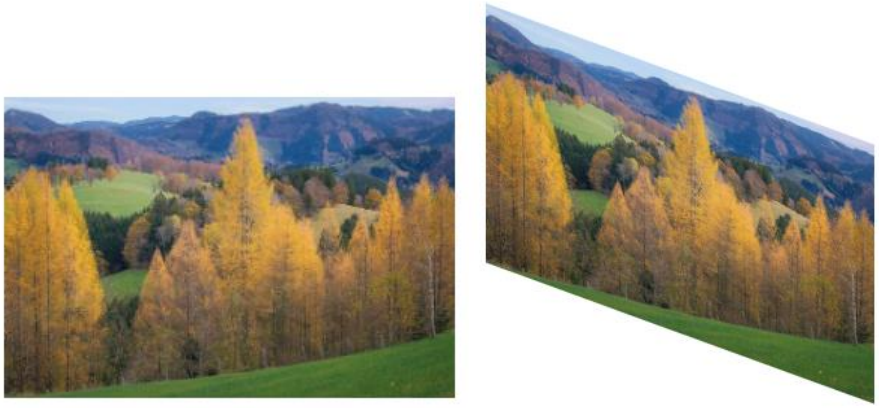
```
<head>
<style>
.i1{
  margin: 100px;
  width:300px;height:200px
}
.i1:hover {
 transform: skewX(20deg);
}
</style>
</head>
<body>
   <div>
     <img src="a1.jpg" alt="hello" class="i1">
   </div>
</body>
```

transform: skew(20deg,0deg); also gives the same effect as skewX(20deg). Keep the angle from y axis 0deg

- ✓ The skewY() method skews an element along the Y-axis by the given angle.
- ✓ The following example skews the element 20 degrees along the Y-axis:

```
<head>
<style>
.i1{
  margin: 100px;
  width:300px;height:200px
}
.i1:hover {
 transform: skewY(20deg);
}
</style>
</head>
<body>
   <div>
```

```
    <img src="a1.jpg" alt="hello" class="i1">
  </div>
</body>
```



transform: skew(0deg,20deg); also gives the same effect as skewY(20deg). Keep the angle from x axis 0deg

Example:

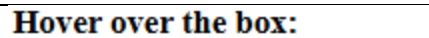
```
<head>
<style>
.i1{
  margin: 100px;
  width:300px;height:200px
}
.i1:hover {
 transform: skew(20deg,20deg);
}
</style>
</head>
<body>
   <div>
     <img src="a1.jpg" alt="hello" class="i1">
   </div>
</body>
```

These transformations can be combined in a single transform property to create complex effects.

For example:

transform: translate(100px, 100px) rotate(90deg) scale(1.5, 1.5);

```
<head>
<style>
.box {
  width: 100px;
  height: 100px;
  background-color: steelblue;
  color: white;
  display: flex;
  align-items: center;
  justify-content: center;
  font-family: sans-serif;
  border-radius: 10px;
}

/* When hovered */
.box:hover {
  transform: translate(100px, 100px) rotate(90deg) scale(1.5, 1.5);
  background-color: tomato; /* changes color */
}
</style>
</head>
<body>

<h3>Hover over the box:</h3>
<div class="box">Box</div>

</body>
```



Example:

```
<head>
<style>
div {
  margin:100px;
  width: 100px;
  height: 100px;
  background-color: lightblue;
```

```
}
div:hover {
  transform: rotate(45deg) scale(1.5) translate(50px, 50px);
}
</style>
</head>
<body>
<div></div>
</body>
```

Before **After**

- ✓ The square starts as a 100px by 100px blue box.
- ✓ **Hover Effect (div:hover)**: When hovered, the box rotates 45 degrees, scales up by 1.5 times, and shifts 50px from x and y axis.

Transform Function	Example	Description / Effect
Translate (move element)		
translate(20px)	Moves element **20px right**	Equivalent to translateX(20px) — moves along **X-axis** only.
translate(20px, 20px)	Moves element **20px right and 20px down**	First value = **X**, second = **Y**.
translateX(20px)	Moves element **20px right**	Positive → right, negative → left.
translateY(20px)	Moves element **20px down**	Positive → down, negative → up.
Scale (resize element)		
scale(1)	No change	1 means 100% original size.
scale(2, 1.5)	Width ×2, Height ×1.5	First = **X scale**, second = **Y scale**.
scale(2)	Doubles size both horizontally and vertically	Equivalent to scale(2, 2).
scaleX(2)	Doubles width only	Height remains unchanged.
scaleY(2)	Doubles height only	Width remains unchanged.
scale(-1)	Flips element **horizontally and vertically**	Same as scale(-1, -1) → mirrored both axes.
scaleX(-1)	Flips **horizontally** (mirror image left–right)	Common trick to flip images or icons.
scaleY(-1)	Flips **vertically** (mirror image top–bottom)	Often used for reflection effects.
Rotate (turn element)		

Transform Function	Example	Description / Effect
rotate(45deg)	Rotates element **45° clockwise**	Rotation center is by default the element's center.
rotate(-45deg)	Rotates element **45° counterclockwise**	Negative values rotate opposite direction.
Skew (tilt element)		
skew(45deg)	Tilts element **45° along X-axis**	Equivalent to skewX(45deg).
skew(45deg, 20deg)	Tilts element **45° on X**, **20° on Y**	Creates a combined shearing effect.
skewX(15deg)	Tilts only along **X-axis**	Top edge stays still, bottom edge moves.
skew(15deg, 0deg)	Tilts **15° along X**, none along Y	Same as skewX(15deg).
skewY(15deg)	Tilts only along **Y-axis**	Left edge stays still, right edge moves.

CSS Transitions

- ✓ CSS transitions allows you to change property values smoothly, over a given duration.

Transition

- ✓ Transitions in CSS allow us to control the way in which transition takes place between the two states of the element.
- ✓ For example, when hovering your mouse over a button, you can change the background color of the element with help of CSS selector and pseudo-class.
- ✓ We can change any other or combination of properties also.
- ✓ The transition allows us to determine how the change in color takes place. Hence, it gives a better user experience and interactivity.
- ✓ To create a transition effect, you must specify two things:
  - the CSS property you want to add an effect to
  - the duration of the effect

Note: If the duration part is not specified, the transition will have no effect, because the default value is 0s.

1. transition-property

Specifies which CSS properties will be animated during the transition.

Syntax:

transition-property: all | property | property1, property2, ..., propertyN;

Values:

- all → Animate all animatable properties.
- property → Animate only a specific property (e.g., background-color, width).
- property1, property2, ... → Animate multiple comma-separated properties.

Example:

transition-property: background-color, transform;

2. transition-duration

Specifies how long the transition should take to complete.

Syntax:

transition-duration: time;

Values:

- Time can be in seconds (s) or milliseconds (ms).

Example:

transition-duration: 0.5s;

3. transition-delay

Specifies how long to wait before starting the transition.

Syntax:

transition-delay: time;

Example:

transition-delay: 0.3s;

4. transition-timing-function

Controls the speed curve of the transition how the animation progresses over time.

Syntax:

transition-timing-function: linear | ease | ease-in | ease-out | ease-in-out

Common Values:

- linear → Same speed from start to end
- ease → Default – starts slow, speeds up, then slows down
- ease-in → Starts slow, then speeds up
- ease-out → Starts fast, then slows down
- ease-in-out → Starts and ends slow

Example:

transition-timing-function: ease-in-out;

Shorthand

Syntax:

transition: property duration timing-function delay;

Example:

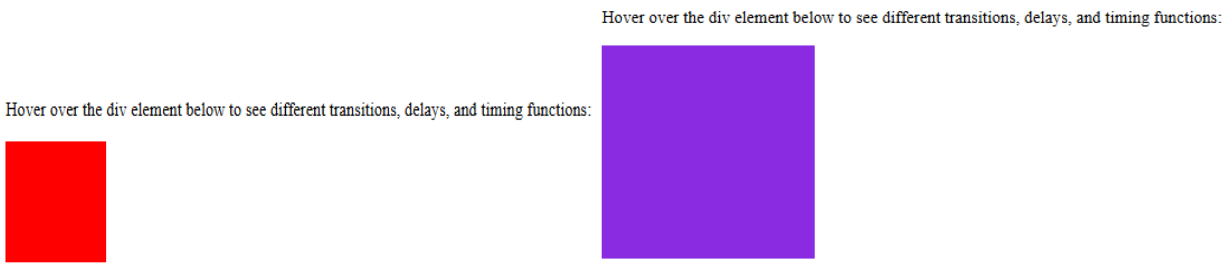
transition: background-color 0.5s ease-in-out 0.2s;

Property	Required in shorthand?	Default value
transition-property	No (defaults to all)	all
transition-duration	Yes **(must specify)**	0s
transition-timing-function	No	ease
transition-delay	No	0s

Example:

```
<!DOCTYPE html>
<html>
<head>
<style>
div {
  width: 100px;
  height: 100px;
  background: red;

  /* Multiple transitions with custom duration, delay, and timing */
  transition: width 5s ease-in 1s,height 6s ease-out 0.5s,background-color 7s ease-in-out 2s;
}

div:hover {
  width: 200px;
  height: 200px;
```

```
  background-color: blueviolet;
}
</style>
</head>
<body>
  <p>Hover over the div element below to see different transitions, delays, and timing functions:</p>
  <div></div>
</body>
</html>
```



Before Hover **After Hover**

Example:

Create a CSS program that demonstrates the use of multiple transitions on a single element.
When the user hovers over the div, it should:

- **Increase in size,**
- **Change background color,**
- **Rotate 360 degrees,**
- **Become circular,**
  all with smooth timing, delay, and different transition durations.

```
<head>
<style>
div {
  width: 100px;
  height: 100px;
  background: rgb(142, 7, 233);

  /* Transition for multiple properties including transform */
  transition-property: width, height, transform, background-color,border-radius;
  transition-duration: 3s, 5s, 4s,6s,6s;
  transition-delay: 2s; /* Wait 2s before starting transition */
  transition-timing-function: ease-in-out; /* Smooth start and end */
}
div:hover {
  width: 150px;
  height: 150px;
  transform: rotate(360deg);
  background-color: yellow;
  border-radius:50%;
}
</style>
</head>
<body>
```

```
 <div></div>
</body>
```

Example:

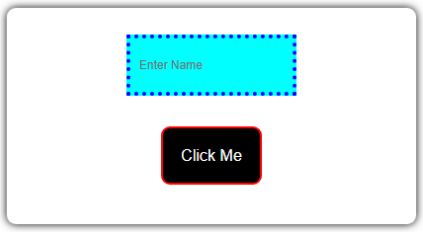
Create an interactive HTML form that demonstrates the use of **CSS transitions** to enhance user experience.
When the user interacts with the form elements:

- The **input field** should smoothly change its background color, text color, and border style when hovered.
- The **button** should smoothly change its background color, text color, padding, and border thickness when hovered.
- The transitions should be visually appealing and use easing for smooth motion.

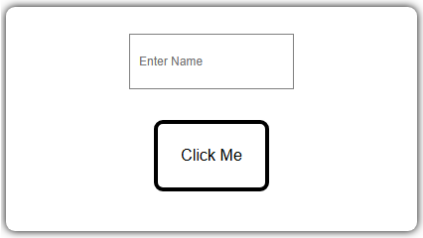
```
<head>
<style>
form {
  display: block;
  margin: 30px auto;
  width: 400px;
  padding: 30px;
  border-radius: 10px;
  box-shadow: 0 0 10px black;
  text-align: center;
}
/* Input field styling */
.i1 {
  height: 40px;
  border: 2px solid gray;
  padding: 10px;
  transition: all 0.5s ease;
}

/* Input hover effect */
.i1:hover {
  height: 40px;
  background-color: aqua;
  color: blue;
  border: 5px dotted blue;
}

/* Button styling */
.b1 {
  background-color: black;
  color: white;
  font-size: 18px;
  padding: 20px;
  border: 3px solid red;
  border-radius: 10px;
  margin: 15px;
  transition: all 0.4s ease;
}
```

```
/* Button hover effect */
.b1:hover {
  background-color: white;
  color: black;
  padding: 25px;
  border: 5px solid black;
}
</style>
</head>
<body>
<form>
  <input type="text" class="i1" placeholder="Enter Name"/><br><br>
  <input type="submit" class="b1" value="Click Me"/>
</form>
</body>
```

Before



After hovering field



After hovering button

CSS Animation

- ✓ CSS allows animation of HTML elements without using JavaScript or Flash!
- ✓ An animation lets an element gradually change from one style to another.
- ✓ You can change as many CSS properties you want, as many times as you want.
- ✓ To use CSS animation, you must first specify some keyframes for the animation.
- ✓ Keyframes hold what styles the element will have at certain times.

The @keyframes Rule

- ✓ When you specify CSS styles inside the **@keyframes rule**, the animation will gradually change from the current style to the new style at certain times.
- ✓ To get an animation to work, you must bind the animation to an element.

Properties

- **animation-name**: It is used to specify the name of the @keyframes describing the animation.
- **animation-duration**: It is used to specify the time duration it takes animation to complete one cycle.
  - If the animation-duration property is not specified, no animation will occur, because the default value is 0s (0 seconds).
- **animation-delay:** It specifies the delay of the start of an animation.
- **animation-timing-function:** Controls **how the animation progresses over time** i.e., the *speed curve* of the animation.
  Syntax: animation-timing-function: linear | ease | ease-in | ease-out | ease-in-out
- **animation-iteration-count:** This specifies the number of times the animation will be repeated.
  - The animation-iteration-count property can be set to **infinite** to let the animation run for ever.
- **animation-direction:** It defines the direction of the animation. animation direction can be **normal**, **reverse**, **alternate**, and **alternate-reverse**.
  - **normal**: **default** forward direction (clockwise/left-right)
  - **reverse:** backward direction (anti clockwise/right-left)
  - **alternate:** start with forward and then backward. Minimum 2 iteration-count required. To perform alternate animation.
  - **alternate-reverse:** start with backward and then forward. Minimum 2 iteration-count required. To perform alternate-reverse animation.

Property	Purpose
animation-name	Links to the @keyframes definition
animation-duration	Sets total time for one cycle
animation-delay	Sets start delay before animation begins
animation-iteration-count	Sets how many times animation repeats
animation-direction	Defines forward/reverse play direction
animation-timing-function	Controls speed curve of animation

The following example binds the "example" animation to the element. The animation will last for 4 seconds, and it will gradually change the background-color of the element from "red" to "yellow". Animation will start after 2s of delay. Also, it will start in backward direction and performs 5 times.

```
/* The animation code */
@keyframes example {
  from {background-color: red;}
  to {background-color: yellow;}
}
/* The element to apply the animation to */
div {
  width: 100px;
  height: 100px;
  background-color: red;
  animation-name: example;
  animation-duration: 4s;
  animation-direction: reverse;
  animation-iteration-count:5;
  animation-delay:2s;
  animation-timing-function: ease-in-out; /* Added timing for smooth start & end */
}
```

In the above example we have specified when the style will change by using the keywords "from" and "to" (which represents 0% (start) and 100% (complete)).

It is also possible to use percent. By using percent, you can add as many style changes as you like.

Example: Simple Animation

```
<html>
<head>
<style>
div {
animation-name: example;
width: 100px;
height: 100px;
background-color: red;
animation-duration: 4s;
animation-delay: 2s;
animation-timing-function: ease-in-out; /* Added timing for smooth start & end */
}
@keyframes example {
from {background-color: red;}
to {background-color: green;}
}
</style>
</head>
<body>
<div></div>
<p><b>Note:</b> When an animation is finished, it goes back to its original style.</p>
</body>
</html>
```

Output (before animation):

Note: When an animation is finished, it goes back to its original style.

Output (after animation)

Note: When an animation is finished, it goes back to its original style.

Example (Using percentage)

```
<html>
<head>
<style>
div {
width: 100px; height: 100px;
background-color: purple;
animation-name: example;
animation-duration: 4s;
animation-iteration-count: 2;
animation-timing-function: ease-in-out; /* Added timing for smooth start & end */
}
@keyframes example {
0%  {background-color: red;}
25% {background-color: yellow;}
50% {background-color: blue;}
100% {background-color: green;}
}
</style>
</head>
<body>
<div></div>
<p><b>Note:</b> When an animation is finished, it goes back to its original style.</p>
</body>
</html>
```

Output:

Note: When an animation is finished, it goes back to its original style.

Animation starts after 2s and then color will be start changing from red to mentioned colors in above example.

Example: position and color change

The following example will change both the background-color and the position of the element when the animation is 25% completed, 50% completed, 75% completed and again when the animation is 100% completed:

```
<html>
<head>
<style>
div {
width: 100px;
height: 100px;
background-color: pink;
position: relative;
animation-name: example;
animation-duration: 10s;
animation-delay: 2s;
animation-iteration-count: infinite;
}
@keyframes example {
0%  {background-color:red; left:0px; top:0px;}
25% {background-color:yellow; left:200px; top:0px;}
50% {background-color:blue; left:200px; top:200px;}
75% {background-color:green; left:0px; top:200px;}
100% {background-color:red; left:0px; top:0px;}
}
</style>
</head>
<body>
<div></div>
</body>
</html>
```

- ✓ Animation starts after 2s of delay.
- ✓ Initial color pink.
- ✓ Starts in forward direction clockwise with red color at position left 0 and top 0. (0%)
- ✓ Moves to right (left:200px,top:0px) and changes to yellow color. (25%)
- ✓ Moves to bottom (left:200px,top:200px) and changes to blue color. (50%)
- ✓ Moves to left (left:0px,top:200px) and changes to green color. (75%)
- ✓ Moves to original position top 0 and left 0 and changes to red color. (100%)

Example (Negative animation-delay value)

- ✓ Negative values are also allowed. If using negative values, the animation will start as if it had already been playing for N seconds.
- ✓ In the following example, the animation will start as if it had already been playing for 2 seconds:

```
<html>
<head>
<style>
```

```
div {
width: 100px;
height: 100px;
background-color: red;
position: relative;
animation-name: example;
animation-duration: 10s;
animation-delay: -2s;
}
@keyframes example {
0%  {background-color:red; left:0px; top:0px;}
25% {background-color:yellow; left:200px; top:0px;}
50% {background-color:blue; left:200px; top:200px;}
75% {background-color:green; left:0px; top:200px;}
100% {background-color:red; left:0px; top:0px;}
}
</style>
</head>
<body>
<p>Using negative values in the animation-delay property: Here, the animation will start as if it had already been playing for 2 seconds:</p>
<div></div>
</body>
</html>
```

Example: Animation in reverse direction with infinite iteration count

```
<html>
<head>
<style>
div {
width: 100px;

height: 100px;
background-color: red;
position: relative;
animation-name: example;
animation-duration: 4s;
animation-direction: reverse;
border-radius: 50%;
animation-iteration-count: infinite;
}
@keyframes example {
0%  {background-color:red; left:0px; top:0px;}
25% {background-color:yellow; left:200px; top:0px;}
50% {background-color:blue; left:200px; top:200px;}
75% {background-color:green; left:0px; top:200px;}
100% {background-color:red; left:0px; top:0px;}
}
</style>
</head>
```

```
<body>
<p>The animation-direction property specifies whether an animation should be played forwards, backwards or in alternate cycles. This example will run the animation in reverse direction (backwards):</p>
<div></div>
</body>
</html>
```

CSS Gradients

- ✓ The **Gradient** in CSS is a special type of image that is made up of progressive & smooth transition between two or more colors.
- ✓ CSS is the way to add style to various web documents.
- ✓ By using the gradient in CSS, we can create variants styling of images which can help to make an attractive webpage.
- ✓ CSS gradients let you display smooth transitions between two or more specified colors.

CSS defines three types of gradients:

- **Linear Gradients (goes down/up/left/right/diagonally)**
- **Radial Gradients (defined by their center)**
- **Conic Gradients (rotated around a center point)**

Linear Gradients

- ✓ It includes the smooth color transitions to going up, down, left, right, and diagonally.
- ✓ The minimum two-color required to create a linear gradient.
- ✓ More than two color elements can be possible in linear gradients.
- ✓ Color stops are the colors you want to render smooth transitions among.
- ✓ You can also set a starting point and a direction (or an angle) along with the gradient effect.

Syntax:

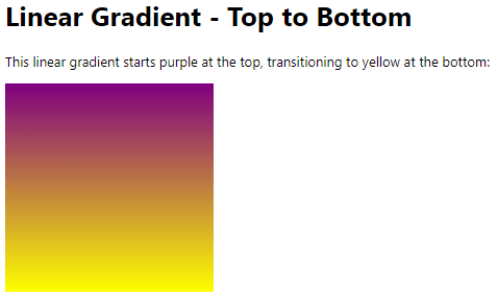
```
background-image: linear-gradient(direction, color-stop1, color-stop2, ...);
or
background: linear-gradient(direction, color-stop1, color-stop2, ...);
```

If you **don't specify a direction**, the gradient defaults to **top → bottom** (i.e., **to bottom**).

The linear-gradient can be implemented in the following ways:

Top to Bottom: The transition starts from top and end at bottom.

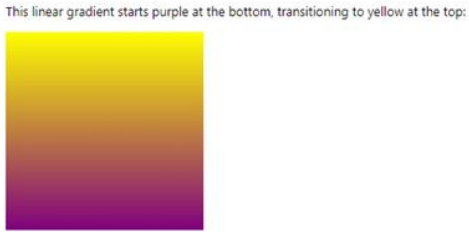
```
<html>
<head>
<style>
#grad1 {
  height: 200px;
  width: 200px;
  background-color: red; /* For browsers that do not support gradients */
  background-image: linear-gradient(purple, yellow);
}
</style>
</head>
<body>
<h1>Linear Gradient - Top to Bottom</h1>
<p>This linear gradient starts red at the top, transitioning to yellow at the bottom:</p>
<div id="grad1"></div>
</body>
</html>
```



- ✓ In this image, the transition started with purple color and ended with yellow color.
- ✓ On exchanging the color sequence, the transition will start with yellow and will end with purple.
- ✓ This is default. If you don't define any direction then default direction will be **to bottom.**

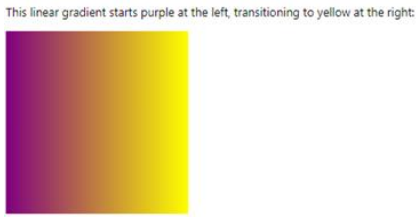
Bottom to Top: The transition starts from bottom and end at top.

```
background-image: linear-gradient(to top,purple, yellow);
```



Left to Right: In this, the transition starts from left to right.

```
background-image: linear-gradient(to right,purple, yellow);
```



Left to Right: In this, the transition starts from right to left.

```
background-image: linear-gradient(to left,purple, yellow);
```

Diagonal: For the diagonal gradient, need to specify both horizontal and vertical starting positions. This transition started from one of the below directions.

- ✓ **to top left / to left top**

```
background-image: linear-gradient(to top left, purple, yellow);
                                  or
background-image: linear-gradient(to left top, purple, yellow);
```

This linear gradient starts purple at the right bottom, transitioning to yellow at the left top:

✓ **to top right / to right top**

```
background-image: linear-gradient(to top right, purple, yellow);
                                or
background-image: linear-gradient(to right top, purple, yellow);
```

This linear gradient starts purple at the left bottom, transitioning to yellow at the right top:

✓ **to bottom right / to right bottom**

```
background-image: linear-gradient(to bottom right, purple, yellow);
                                or
background-image: linear-gradient(to right bottom, purple, yellow);
```

This linear gradient starts purple at the left top, transitioning to yellow at the right bottom:

✓ **to bottom left / to left bottom**

```
background-image: linear-gradient(to bottom left, purple, yellow);
                                or
background-image: linear-gradient(to left bottom, purple, yellow);
```

This linear gradient starts purple at the right top, transitioning to yellow at the left bottom:

Example (Using angle)

- ✓ If you want more control over the direction of the gradient, you can define an angle, instead of the predefined directions (to bottom, to top, to right, to left, to bottom right, etc.).
- ✓ A value of 0deg is equivalent to "to top".
- ✓ A value of 90deg is equivalent to "to right".
- ✓ A value of 180deg is equivalent to "to bottom".

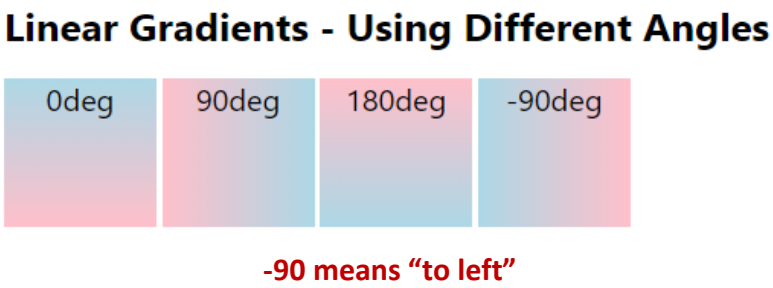
- ✓ A value of 270deg is equivalent to "to left".

Syntax:

background-image: linear-gradient(*angle*, *color-stop1*, *color-stop2…*);

Example:

```
<html>
<head>
<style>
#grad1,#grad2,#grad3,#grad4 {
  height: 100px;
  width: 100px;
  display: inline-block;
  font-size: 20px;
text-align:center;
}
#grad1{ background-image: linear-gradient(0deg,pink,lightblue); }
#grad2 { background-image: linear-gradient(90deg,pink,lightblue); }
#grad3 { background-image: linear-gradient(180deg,pink,lightblue); }
#grad4 { background-image: linear-gradient(-90deg,pink,lightblue); }
</style>
</head>
<body>
<h1>Linear Gradients - Using Different Angles</h1>
<div class="d1">
<div id="grad1">0deg</div>
<div id="grad2">90deg</div>
<div id="grad3">180deg</div>
<div id="grad4">-90deg</div>
</div>
</body>
</html>
```



-90 means "to left"

Example:

Evenly spaced/not evenly spaced

```
<html>
<head>
<style>
#grad1 {
  height: 100px;
  width:100px;
  background-image: linear-gradient(red, yellow, green);
}
```

```
#grad2 {
  height: 100px;
  width:100px;
  background-image: linear-gradient(red, orange, yellow, green, blue, indigo, violet);
}
#grad3 {
  height: 100px;
  width:100px;
  background-image: linear-gradient(pink 20%, lightgreen 30%, lightblue 50%);
}
</style>
</head>
<body>
<h2>3 Color Stops (evenly spaced):</h2>
<div id="grad1"></div>
<h2>7 Color Stops (evenly spaced):</h2>
<div id="grad2"></div>
<h2>3 Color Stops using percentage (not evenly spaced):</h2>
<div id="grad3"></div>
<p><strong>Note:</strong> Color stops are spaced evenly when no percent is specified.</p>
</body>
</html>
```

3 Color Stops (evenly spaced):

7 Color Stops (evenly spaced):

3 Color Stops using percentage (not evenly spaced):

Note: Color stops are spaced evenly when no percents are specified.

Example: Rainbow Background

```
<html>
<head>
<style>
#grad1 {
height: 50px;
background-image: linear-gradient(to right, red, orange, yellow, green, blue, indigo, violet);
text-align: center;
color: white;
font-size: 20px;
margin-top: 20px;
}
</style>
</head>
```

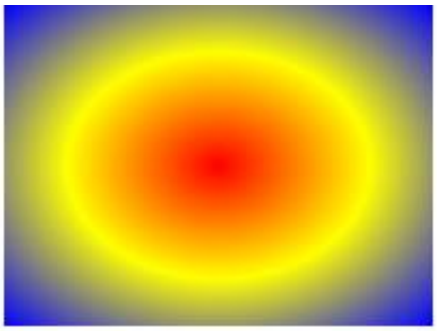
```
<body>
<div id="grad1">Rainbow Background</div>
</body>
</html>
```

CSS Radial Gradients

- ✓ The radial-gradient() function sets a radial gradient as the background image.
- ✓ A radial gradient is defined by its center.
- ✓ To create a radial gradient you must define at least two color stops.

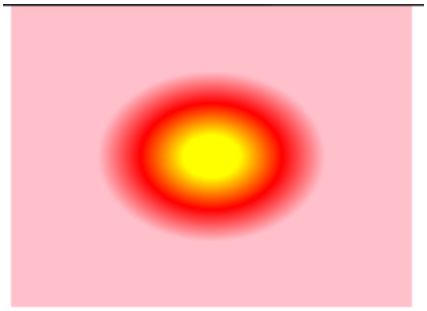
Example (Evenly Spaced color stops by default):

```
<html>
<head>
<style>
#grad1 {
height: 150px;
width: 200px;
background-color: red; /* For browsers that do not support gradients */
background-image: radial-gradient(red,yellow,blue);
}
</style>
</head>
<body>
<div id="grad1"></div>
</body>
</html>
```



Example (Differently Spaced color stops)

```
<html>
<head>
<style>
#grad1 {
height: 150px;
width: 200px;
background-color: red; /* For browsers that do not support gradients */
background-image: radial-gradient(yellow 10%,red 25%, pink 40%);
}
</style>
</head>
<body>
<div id="grad1"></div>
</body>
</html>
```



Radial gradient using shape

Unlike a linear gradient, which moves in a straight line, a radial gradient spreads out in circles or ellipses.

Ellipse (default):
The gradient is stretched to fill the element's shape.
background-image: radial-gradient(purple, yellow, pink);
Creates an elliptical gradient (default behavior).

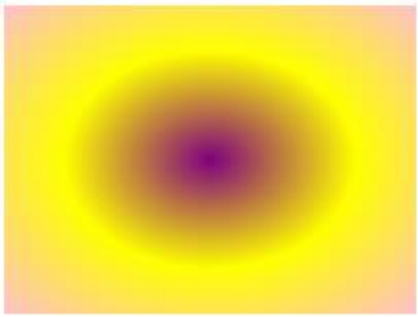
Circle:
Forces the gradient to be perfectly round.
background-image: radial-gradient(circle, blue, yellow, pink);
Creates a circular gradient.

Example (Define shape)

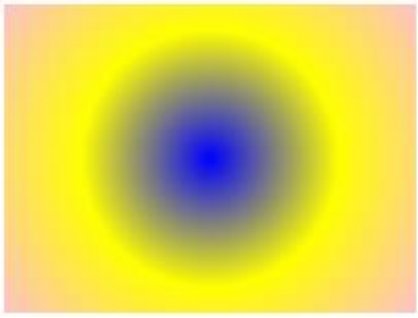
```
<html>
<head>
<style>
#grad1 {
height: 150px;
width: 200px;
background-color: red; /* For browsers that do not support gradients */
background-image: radial-gradient(purple, yellow, pink);
}
#grad2 {
height: 150px;
```

```
width: 200px;
background-color: red; /* For browsers that do not support gradients */
background-image: radial-gradient(circle, blue, yellow, pink);
}
</style>
</head>
<body>
<h2>Ellipse (this is default):</h2>
<div id="grad1"></div>
<h2><strong>Circle:</strong></h2>
<div id="grad2"></div>
</body>
</html>
```

Ellipse (this is default):



Circle:



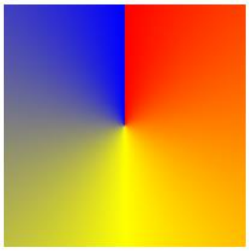
CSS Conic Gradients

- ✓ A conic gradient is a gradient with color transitions rotated around a center point. **Default** started from **0deg**.
- ✓ To create a conic gradient, you must define at least two colors.

Example:

```
<html>
<head>
<style>
#grad1 {
height: 200px;
width: 200px;
background-color: red; /* For browsers that do not support gradients */
background-image: conic-gradient(red, yellow, blue);
}
</style>
</head>
<body>
<h1>Conic Gradient - Three Colors</h1>
<div id="grad1"></div>
</body>
</html>
```

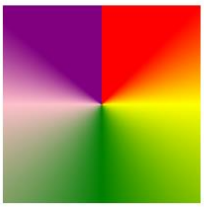
Conic Gradient - Three Colors



Example (colors with degrees)

```
<html>
<head>
<style>
#grad1 {
height: 200px;
width: 200px;
background-color: red; /* For browsers that do not support gradients */
background-image: conic-gradient(red 45deg, yellow 90deg, green 180deg,pink 270deg, purple
310deg);
}
</style>
</head>
<body>
<h1>Conic Gradient - Defined degree for each color</h1>
<div id="grad1"></div>
</body>
</html>
```

Conic Gradient - Defined degree for each color



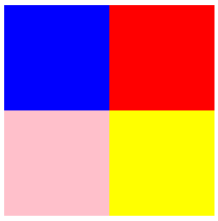
- ✓ till 45 deg red color,
- ✓ 45 to 90 deg transition of red to yellow,
- ✓ 90 to 180 deg transition of yellow to green,
- ✓ 180 to 270 deg transition of green to pink,
- ✓ 270 to 310 deg transition of pink to purple,
- ✓ Then purple color upto 0 deg.

Example: Defined degrees for all the colors

If we mention color with start and end degree then it will give an output as shown below.

```
<html>
<head>
<style>
#grad1 {
height: 200px;
width: 200px;
background-color: red; /* For browsers that do not support gradients */
background-image: conic-gradient(red 0deg, red 90deg, yellow 90deg, yellow 180deg, pink
180deg, pink 270deg, blue 270deg);
}
</style>
</head>
<body>
<h1>Conic Gradient - Pie Chart</h1>
<div id="grad1"></div>
</body>
</html>
```

Conic Gradient - Pie Chart



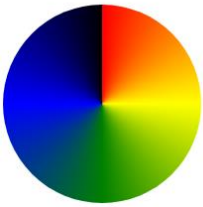
Example (Pie Charts)

- ✓ Just add border-radius: 50% to make the conic gradient look like a pie chart:

```
<head>
<style>
#grad1 {
```

```
height: 200px;
width: 200px;
background-color: red; /* For browsers that do not support gradients */
background-image: conic-gradient(red, yellow, green, blue, black);
border-radius: 50%;
}
</style>
</head>
<body>
<h1>Conic Gradient - Pie Chart</h1>
<div id="grad1"></div>
</body>
```

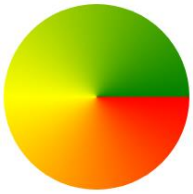
Conic Gradient - Pie Chart



Example (conic gradient with specified from angle):

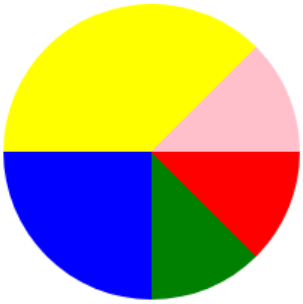
```
<head><style>
#grad1 {
height: 200px;
width: 200px;
background-color: red; /* For browsers that do not support gradients */
background-image: conic-gradient(from 90deg, red, yellow, green);
border-radius: 50%;
}
</style></head>
<body>
<h1>Conic Gradient - With a from angle</h1>
<div id="grad1"></div>
</body>
```

Conic Gradient - With a from angle



Example (conic gradient with specified from angle to create distinct colored segments rather than a smooth gradient transition.

```
<html>
<head>
<style>
#grad1 {
height: 200px;
width: 200px;
background-color: red; /* For browsers that do not support gradients */
background-image: conic-gradient(from 90deg, red 0deg, red 45deg, green 45deg, green 90deg, blue 90deg, blue 180deg, yellow 180deg, yellow 315deg, pink 315deg);
border-radius: 50%;
}
</style>
</head>
<body>
<div id="grad1"></div>
</body>
</html>
```



Here's how the gradient is applied:

The circle's gradient starts at 90 degrees (to the right of the center).

The circle is divided into angular sections based on the specified color stops:

- ✓ Red: 0°–45°
- ✓ Green: 45°–90°
- ✓ Blue: 90°–180°
- ✓ Yellow: 180°–315°
- ✓ Pink: 315°–360°

This type of gradient is often used in:

- Pie charts or circular data visualizations.
- Decorative backgrounds.
- Highlighting specific sections of a circular design.

CSS Variable

- ✓ The **variables** in CSS are just like simple variables of any other programming language.
- ✓ These variables are used to store values and have a scope in which the variables can be used. A variable is defined by using two dashes(–) at the beginning and then the name which is case-sensitive.
- ✓ The benefit of variables is that it allows the same values to be reused at multiple places and updated/modified from one place.
- ✓ Also, the variable names are easier to understand and use.
- ✓ The var() function can be used to take the values of the variables in CSS.
- ✓ CSS variables can have a global or local scope.
- ✓ Global variables can be accessed/used through the entire document, while local variables can be used only inside the selector where it is declared.
- ✓ To create a variable with global scope, declare it inside the :root selector. The :root selector matches the document's root element.
- ✓ To create a variable with local scope, declare it inside the selector that is going to use it.

Syntax:

Define variable globally

```
:root{   --b: blue; }
```

Use of variable

```
P{ color:var(--b) }
```

All the elements of the document can use the variable.

Define variable locally and use it

```
P{ --b: blue; color:var(--b) }
```

It has a scope only for the p element.

Example:

```
<html>
<head>
<style>
:root
{
--b:#1234EE;
--all:solid dashed double dotted;
--p20:20px;
}
p
{
  --c:center;
  --b:brown;
  border-style: var(--all);
  padding: var(--p20);
  border-color: var(--b);
}
```

```
h1{
  color: var(--b);
  text-align: var(--c);
}
</style>
</head>
<body>
<h1>Hello</h1>
<p>Good Morning!</p>
</body>
</html>
```

- ✓ Here, the variables declared inside the root can be accessed by all the elements of the documents.
- ✓ The –b var is also defined with brown value inside the p element. So, here if I use –b variable value inside the p then brown value will be applied inside the p element.
- ✓ Also, --c variable is define in side the p element. Its effect will not get reflected ouside the element. For Example, In above example I have tried to access that –c variable inside the h1 element.

What is The Viewport?

- ✓ The viewport is the user's visible area of a web page.
- ✓ The viewport varies with the device, and will be smaller on a mobile phone than on a computer screen.
- ✓ HTML5 introduced a method to let web designers take control over the viewport, through the <meta> tag.
- ✓ You should include the following <meta> viewport element in all your web pages:
  - **<meta name="viewport" content="width=device-width, initial-scale=1.0">**
- ✓ This gives the browser instructions on how to control the page's dimensions and scaling.
- ✓ The width=device-width part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).
- ✓ The initial-scale=1.0 part sets the initial zoom level when the page is first loaded by the browser.

Meta tag with all other content attribute values

```
<meta name="viewport" content="width=device-width,initial-scale=1.0,user-scalable=no,maximum-scale=0.5,minimum-scale=0.1"/>
```

- ✓ minimum-scale[0.0-10.0] gives minimum size to allow to zoom-out.
- ✓ Maximum-scale[0.0-10.0] gives maximum size to allow to zoom-in.
- ✓ The user-scalable="no" parameter inside the content attribute of <meta name="viewport"> element disables zooming on a page. The maximum-scale parameter limits the amount the user can zoom.

CSS Media queries

The @media rule in CSS is a powerful tool for applying **conditional styles** based on the characteristics of the user's device, such as screen size, resolution, and orientation. It's a cornerstone of **responsive web design**, allowing a website to adapt to various devices and screen conditions.

Key Concepts of @media Rule:

1. **Purpose**:
   The @media rule applies different CSS rules depending on the device's properties.
2. **Media Features**:
   - **width and height**: Define styles based on viewport or device dimensions.
   - **orientation**: Adjust styles based on whether the screen is in landscape or portrait mode.
3. **Logical Operators/Keywords**:
   - **not**: Negates a media query (styles apply if the condition is *not* met).
   - **and**: Combines multiple conditions (all must be true).
   - **or**: Includes either of multiple conditions (any can be true).
   - **only**: Targets specific media types to prevent older browsers from applying styles.
4. **Media Types**:
   Common media types include:
   - screen: For screens such as desktops, tablets, and phones.
   - all: Applies to all devices (default).

Syntax

```
@media not|only mediatype and (mediafeature) and|or (mediafeature) {
  /* CSS rules go here */
}
```

Examples

1. **Target a Specific Width Range:**

```
@media only screen and (min-width: 500px) and (max-width: 700px) {
  body {
    background-color: lightblue;
  }
}
```

2. **Apply Styles for Viewports at Least 500px Wide:**

```
@media (min-width: 500px) {
  body {
    font-size: 16px;
  }
}
```

3. **Exclude Landscape Orientation:**

```
@media not (orientation: landscape) {
  body {
    background-color: lightpink;
  }
}
```

4. **Apply Styles for Viewports up to 500px Wide:**

```
@media screen and (max-width: 500px) {
  body {
    font-size: 14px;
```

```
    }
}
```

not,only,and are all optional.

However, if you use **only**, you must also **specify a media type**.

Media Types

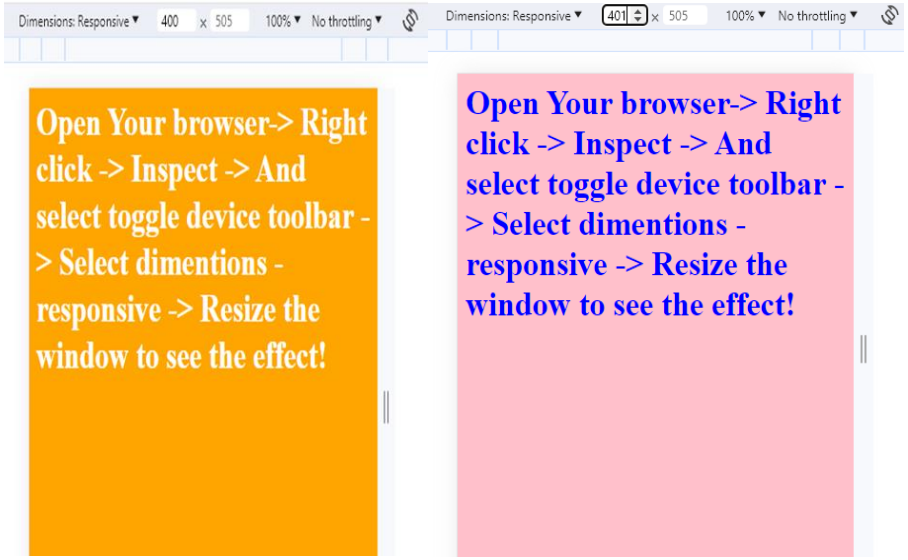
- ✓ **all :** Suitable for all devices.
- ✓ **screen :** Used for computer screens, tablets, smart-phones etc.

EXAMPLE:

```
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
body { background-color: lightblue;}
@media screen and (max-width: 400px) {
   body { background-color: orange;}
}
@media only screen and (min-width: 400px) and (max-width: 1024px) {
   body { background-color: pink; }
}
</style></head>
<body>
<h1>Open Your browser-> Right click -> Inspect -> And select toggle device toolbar -> Select dimentions - responsive -> Resize the window to see the effect! </h1>
</body>
</html>
```

Here,

1) **For up to 400px width** - background-color will be orange and font color will be white.
2) **From 401px to 1000px px width** - background-color will be pink and font color will be blue.
3) **Above 1000px width** - background-color will be lightblue and font color will be red.



Media Queries and Orientation

These allow you to adjust a webpage's layout and styles depending on the **orientation** of the device or browser. This feature is especially useful for creating responsive designs that work seamlessly on both portrait and landscape modes.

Orientation Keyword Values

1. **portrait**:
   - Applies when the viewport's height is greater than or equal to its width.
   - Common for mobile phones held vertically.
2. **landscape**:
   - Applies when the viewport's width is greater than its height.
   - Common for tablets and desktops, or phones held horizontally.

EXAMPLE:

```
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
body {
  background-color: lightgreen;
}
/* Apply styles when NOT in landscape orientation */
@media not (orientation: landscape) {
  body {
    background-color: lightblue;
  }
}
</style>
</head>
<body>
<p>
Resize the browser window.
When the width of this document is larger than the height (landscape),
the background color is <b>lightgreen</b>.
Otherwise (portrait), it is <b>lightblue</b>.
</p>
<b>
Here, the not keyword inverts the media query condition
if the orientation is <u>not landscape</u>, the background becomes lightblue;
else, it remains lightgreen.
</b>
</body>
```

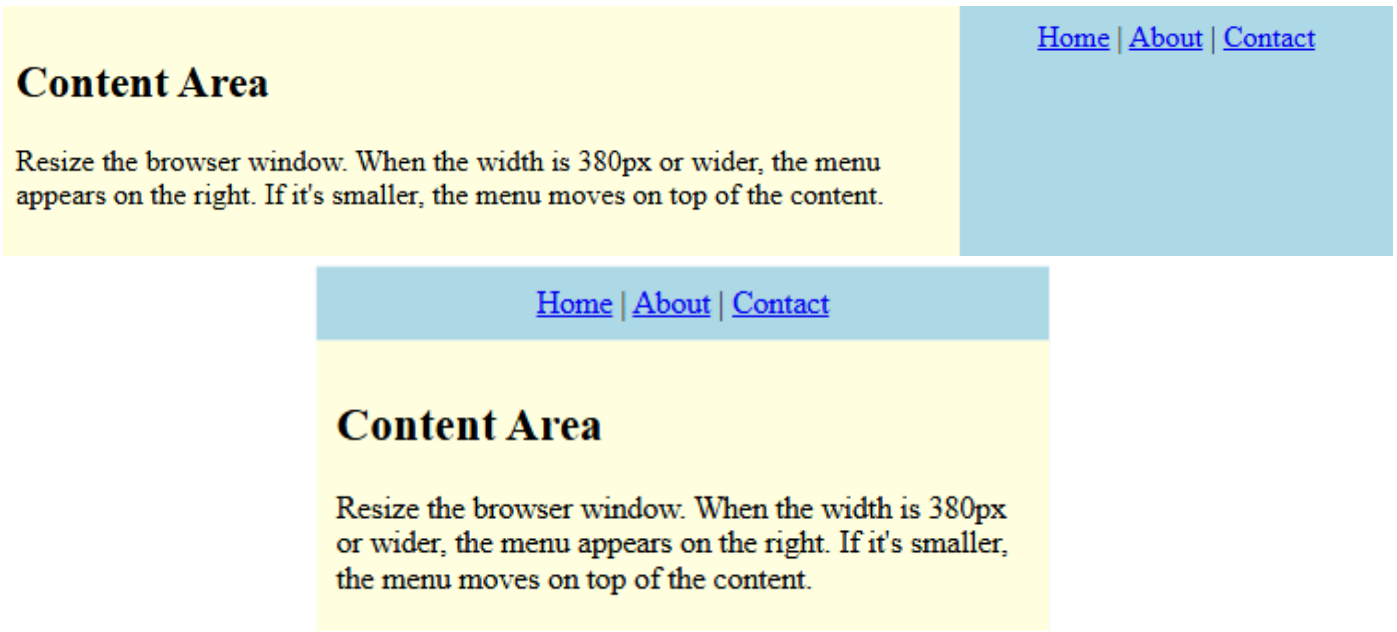
Here, we have added **not** keyword which **inverts the media query**. If **not landscape** then **lightblue** else lightgreen.

Example:

Write down HTML/CSS code that shows a menu that will float to the right of the page if the viewport is 380 pixels wide or wider (if the viewport is less than 380 pixels, the menu will be on top of the content)

```
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
/* Default: row layout (menu on right) */
.container {
  display: flex;
}sss
/* Menu styling */
nav {
  background: lightblue;
  padding: 10px;
  width: 30%;
  text-align: center;
}
/* Content styling */
section {
  background: lightyellow;
  padding: 10px;
  width: 70%;
}
/* When viewport < 380px → stack vertically */
@media (max-width: 380px) {
  .container {
    flex-direction: column;
  }
  nav{order:-1;}
  nav, section {
    width: 100%;
  }
}
</style>
</head>
<body>
<div class="container">
  <section>
    <h2>Content Area</h2>
    <p>
      Resize the browser window.
      When the width is 380px or wider, the menu appears on the right.
      If it's smaller, the menu moves on top of the content.
    </p>
```

```
 </section>
 <nav>
  <a href="#">Home</a> |
  <a href="#">About</a> |
  <a href="#">Contact</a>
 </nav>
</div>
</body>
```



Example:

Write down HTML/CSS code that will display an image (img2.png) and hides an image (img1.png) if viewport is maximum 650 pixels wide. If viewport is greater than 650px then display (img1.png) and hides (img2.png).

```
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
.i1{ display: none;}
@media(max-width:650px) {
.i1{display: block;}
.i2{display: none;}
}
</style></head>
<body>
<ul>
   <div>
      <img src="a.jpg" class="i1" width="200">
      <img src="smiley.png" class="i2" width="200">
  </div>
</ul>
</body></html>
```

Note this for Reference

- Use **em** or **rem** units instead of pixels for better scalability.
- Combine media queries with a **mobile-first approach**, starting with base styles for smaller screens and adding styles for larger ones.

Media queries enhance the adaptability of your designs across various devices, ensuring a consistent user experience.