

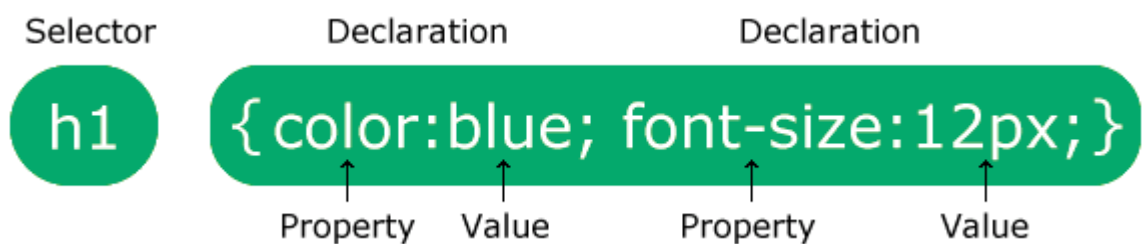


CSS

What is CSS?

- CSS stands for Cascading Style Sheets
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
- CSS saves a lot of work. It can control the layout of multiple web pages all at once
- External stylesheets are stored in CSS files

Syntax



Example :-

```
p {  
  color: red;  
}
```

- **p** is a selector in CSS (it points to the HTML element you want to style: <p>).
- **color** is a property, and **red** is the property value

• Selectors

CSS selectors are used to "find" (or select) the HTML elements you want to style.

• element Selector

```
<html>
<head>
<style>
p { color: red; }
</style>    </head>
<body>
<p>Every paragraph will be affected by the style.</p>
<p>And me!</p>
</body>    </html>
```

• Id Selector

1. The id selector uses the id attribute of an HTML element to select a specific element.

2. The id of an element is unique within a page, so the id selector is used to select one unique element!
3. To select an element with a specific id, write a hash (#) character, followed by the id of the element.

Example : -

```
#para1 {  
  color: red;  
}  
  
<body>  
  
  <p id="para1">Hello World!</p>  
  
  <p>This paragraph is not affected by the style.</p>
```

• class Selector

1. The class selector selects HTML elements with a specific class attribute.
2. To select elements with a specific class, write a period (.) character, followed by the class name.

Example :-

```
.center {  
  text-align: center;  
  color: red;  
}  
  
<body>  
  
  <h1 class="center">Red and center-aligned heading</h1>  
  
  <p class="center">Red and center-aligned paragraph.</p>  
  
</body>
```

Selector	Example	Example description
#id	#firstname	Selects the element with id="firstname"
.class	.intro	Selects all elements with class="intro"
Element.class	p.intro	Selects only <p> elements with class="intro"
*	*	Selects all elements

Three Ways to Insert CSS

There are three ways of inserting a style sheet:

- External CSS
- Internal CSS
- Inline CSS

External CSS

With an external style sheet, you can change the look of an entire website by changing just one file!

Each HTML page must include a reference to the external style sheet file inside the <link> element, inside the head section.

Example :-

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" href="mystyle.css">
</head>
<body>
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

Internal CSS

An internal style sheet may be used if one single HTML page has a unique style.

The internal style is defined inside the <style> element, inside the head section.

Example :-

```
<html>
<head>
<style>
body {
    background-color: linen;
}
h1 {
    color: maroon;
    margin-left: 40px;
}
```

```
}  
</style>  
</head>  
<body>  
<h1>This is a heading</h1>  
<p>This is a paragraph.</p>  
</body>  
</html>
```

Inline CSS

An inline style may be used to apply a unique style for a single element.

To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

Example :-

```
<html>  
<body>  
  
<h1 style="color:blue;text-align:center;">This is a heading</h1>  
<p style="color:red;">This is a paragraph.</p>  
  
</body>  
</html>
```

➤ Backgrounds Properties

1. background-color

The **background-color** property specifies the background color of an element.

Example

```
body {  
  background-color: lightblue;  
}
```

With CSS, a color is most often specified by:

- a valid color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

2. background-image

The **background-image** property specifies an image to use as the background of an element.

By default, the image is repeated so it covers the entire element.

Example

```
body {  
  background-image: url("paper.gif");  
}
```

3. background-repeat

Showing the background image only once is also specified by the **background-repeat** property:

Example

```
body {  
  background-image: url("img_tree.png");  
  background-repeat: no-repeat;  
}
```

4. background-attachment

The **background-attachment** property specifies whether the background image should scroll or be fixed (will not scroll with the rest of the page):

Example

```
body {  
  background-image: url("img_tree.png");  
  background-repeat: no-repeat;  
  background-position: right top;  
  background-attachment: fixed;  
}
```

5. background-position

6. background (shorthand property)

➤ Borders

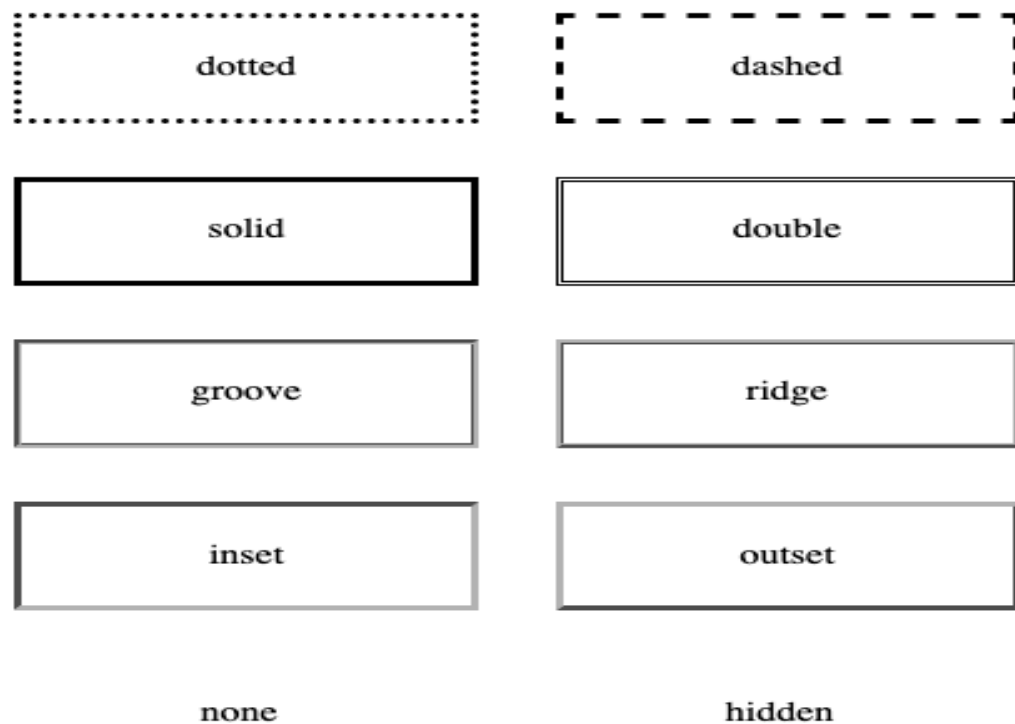
The CSS border properties allow you to specify the style, width, and color of an element's border.

• Border Style

The `border-style` property specifies what kind of border to display.

The following values are allowed:

- `dotted` - Defines a dotted border
- `dashed` - Defines a dashed border
- `solid` - Defines a solid border
- `double` - Defines a double border
- `groove` - Defines a 3D grooved border. The effect depends on the border-color value
- `ridge` - Defines a 3D ridged border. The effect depends on the border-color value
- `inset` - Defines a 3D inset border. The effect depends on the border-color value
- `outset` - Defines a 3D outset border. The effect depends on the border-color value
- `none` - Defines no border
- `hidden` - Defines a hidden border



The **border-style** property can have from one to four values (for the top border, right border, bottom border, and the left border).

Example :-

```
p.dotted {border-style: dotted;}
p.dashed {border-style: dashed;}
p.solid {border-style: solid;}
p.double {border-style: double;}
p.inset {border-style: inset;}
p.outset {border-style: outset;}

p.none {border-style: none;}
p.hidden {border-style: hidden;}

p.groove {border-style: groove;}
p.ridge {border-style: ridge;}
```

• Border Width

The `border-width` property specifies the width of the four borders.

The width can be set as a specific size (in px)

Example :-

```
p.one {  
  border-width: 5px;  
}
```

• Border Color

The `border-color` property is used to set the color of the four borders.

The color can be set by:

- name - specify a color name, like "red"
- HEX - specify a HEX value, like "#ff0000"
- RGB - specify a RGB value, like "rgb(255,0,0)"

Example :-

```
p.one {  
  border-style: solid;  
  border-color: red;  
}
```

• Border - Individual Sides

In CSS, there are also properties for specifying each of the borders (top, right, bottom, and left):

Example :-

```
p {  
  border-top-style: dotted;
```

```
border-right-style: solid;
border-bottom-style: dotted;
border-left-style: solid;
}
```

• CSS Border - Shorthand Property

To shorten the code, it is also possible to specify all the individual border properties in one property.

The `border` property is a shorthand property for the following individual border properties:

- `border-width`
- `border-style` (required)
- `border-color`

Example :-

```
p {
  border: 5px solid red;
}
```

You can also specify all the individual border properties for just one side:

Left Border

```
p {
  border-left: 6px solid red;
  background-color: lightgrey;
}
```

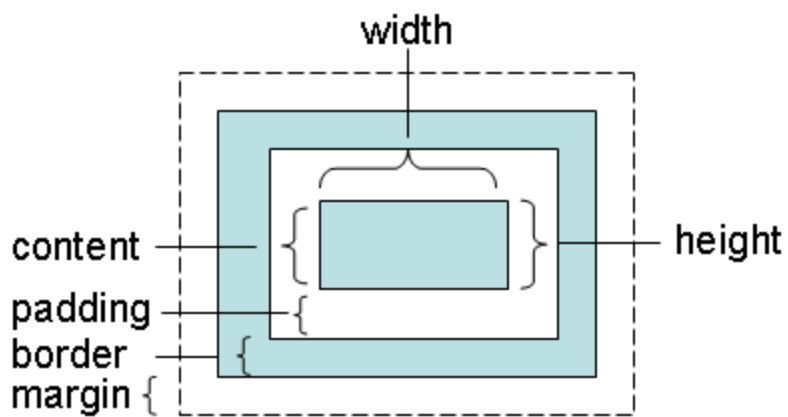
• CSS Rounded Borders

The `border-radius` property is used to add rounded borders to an element:

Example :-

```
p {  
  border: 2px solid red;  
  border-radius: 5px;  
}
```

➤ Height and Width



The CSS `height` and `width` properties are used to set the height and width of an element.

Example

```
div {  
  height: 200px;  
  width: 50%;  
  background-color: powderblue;  
}
```



Margins

The CSS `margin` properties are used to create space around elements, outside of any defined borders.

With CSS, you have full control over the margins. There are properties for setting the margin for each side of an element (top, right, bottom, and left).

SS has properties for specifying the margin for each side of an element:

- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`

Example

```
p {  
  margin-top: 100px;  
  margin-bottom: 100px;  
  margin-right: 150px;  
  margin-left: 80px;  
}
```

• The auto Value

You can set the margin property to `auto` to horizontally center the element within its container.

The element will then take up the specified width, and the remaining space will be split equally between the left and right margins.

```
div {  
  width: 300px;  
  margin: auto;  
  border: 1px solid red;  
}
```

➤ Padding

The CSS **padding** properties are used to generate space around an element's content, inside of any defined borders.

With CSS, you have full control over the padding. There are properties for setting the padding for each side of an element (top, right, bottom, and left)

CSS has properties for specifying the padding for each side of an element:

- **padding-top**
- **padding-right**
- **padding-bottom**
- **padding-left**

Example

```
div {  
  padding-top: 50px;  
  padding-right: 30px;  
  padding-bottom: 50px;  
  padding-left: 80px;  
}
```

• Padding and Element Width

The CSS **width** property specifies the width of the element's content area. The content area is the portion inside the padding, border, and margin of an element

So, if an element has a specified width, the padding added to that element will be added to the total width of the element. This is often an undesirable result.

Example

Here, the <div> element is given a width of 300px. However, the actual width of the <div> element will be 350px (300px + 25px of left padding + 25px of right padding):

```
div {  
  width: 300px;  
  padding: 25px;  
}
```

➤ float Property

The **float** property specifies how an element should float.

Example

Let an image float to the left:

```
img {  
  float: right;  
}
```

Example

Let an image float to the left:

```
img {  
  float: left;  
}
```


➤ Text

CSS has a lot of properties for formatting text.

1. Text Color

The `color` property is used to set the color of the text. The color is specified by:

- a color name - like "red"
- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"

Example

```
body {  
  color: blue;  
}  
  
h1 {  
  color: green;  
}
```

2. Text Alignment

The `text-align` property is used to set the horizontal alignment of a text.

A text can be left or right aligned, centered, or justified.

Example

```
h1 {  
  text-align: center;  
}
```

```
h2 {  
  text-align: left;  
}  
  
h3 {  
  text-align: right;  
}
```

3. Vertical Alignment

The `vertical-align` property sets the vertical alignment of an element.

This example demonstrates how to set the vertical alignment of an image in a text:

Example

```
img.top {  
  vertical-align: top;  
}  
  
img.middle {  
  vertical-align: middle;  
}  
  
img.bottom {  
  vertical-align: bottom;  
}
```

4. Text Decoration

The `text-decoration` property is used to set or remove decorations from text.

The value `text-decoration: none;` is often used to remove underlines from links:

Example

```
a {  
  text-decoration: none;  
}
```

The other `text-decoration` values are used to decorate text:

Example

```
h1 {  
  text-decoration: overline;  
}  
  
h2 {  
  text-decoration: line-through;  
}  
  
h3 {  
  text-decoration: underline;  
}
```

5. Text Transformation

The `text-transform` property is used to specify uppercase and lowercase letters in a text.

It can be used to turn everything into uppercase or lowercase letters, or capitalize the first letter of each word:

Example

```
p.uppercase {  
  text-transform: uppercase;  
}
```

```
}
```

```
p.lowercase {  
  text-transform: lowercase;  
}
```

```
p.capitalize {  
  text-transform: capitalize;  
}
```

• Text Spacing

I. Text Indentation

The `text-indent` property is used to specify the indentation of the first line of a text:

Example

```
p {  
  text-indent: 50px;  
}
```

II. Letter Spacing

The `letter-spacing` property is used to specify the space between the characters in a text.

The following example demonstrates how to increase or decrease the space between characters:

Example

```
h1 {  
  letter-spacing: 3px;  
}  
  
h2 {  
  letter-spacing: -3px;  
}
```

III. Line Height

The `line-height` property is used to specify the space between lines:

Example

```
p.small {  
  line-height: 0.8;  
}  
  
p.big {  
  line-height: 1.8;  
}
```

IV. Word Spacing

The `word-spacing` property is used to specify the space between the words in a text.

The following example demonstrates how to increase or decrease the space between words:

Example

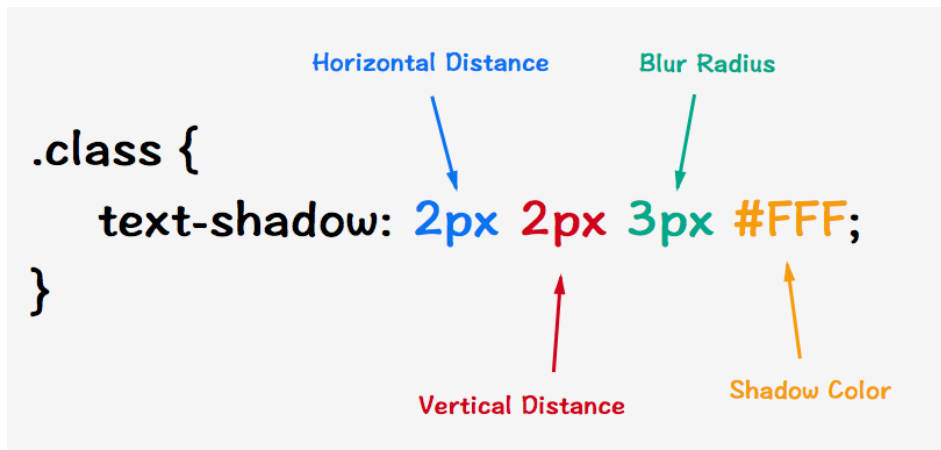
```
h1 {  
  word-spacing: 10px;  
}
```

```
h2 {  
  word-spacing: -5px;  
}
```

V. Text Shadow

The `text-shadow` property adds shadow to text.

In its simplest use, you only specify the horizontal shadow (2px) and the vertical shadow (2px):



Example

```
h1 {  
  text-shadow: 2px 2px;  
}
```

Next, add a color (red) to the shadow:

Example

```
h1 {  
  text-shadow: 2px 2px red;  
}
```

Then, add a blur effect (5px) to the shadow:

Example

```
h1 {  
  text-shadow: 2px 2px 5px red;  
}
```

➤ display Property

- ✓ The **display** property is the most important CSS property for controlling layout.
- ✓ The **display** property specifies if/how an element is displayed.
- ✓ Every HTML element has a default display value depending on what type of element it is. The default display value for most elements is **block** or **inline**.

Display: none;

display: none; is commonly used with JavaScript to hide and show elements without deleting and recreating them. Take a look at our last example on this page if you want to know how this can be achieved.

Override The Default Display Value

Changing an inline element to a block element, or vice versa, can be useful for making the page look a specific way, and still follow the web standards.

A common example is making inline `` elements for horizontal menus:

Example

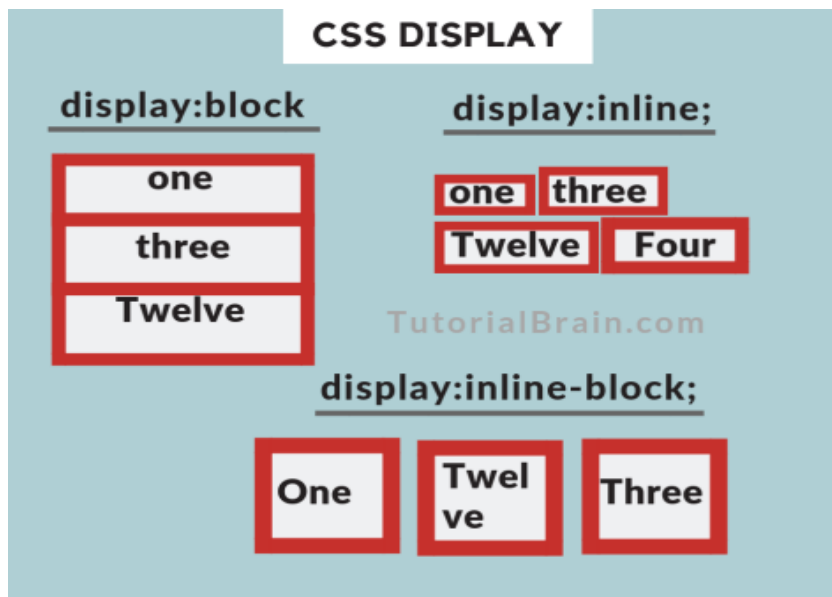
```
li {  
  display: inline;  
}
```

Hiding an element can be done by setting the `display` property to `none`. The element will be hidden, and the page will be displayed as if the element is not there:

Example

```
h1.hidden {  
  display: none;  
}
```

- `display: inline-block`



Compared to **display: inline**, the major difference is that **display: inline-block** allows to set a width and height on the element.

Also, with **display: inline-block**, the top and bottom margins/paddings are respected, but with **display: inline** they are not.

Compared to **display: block**, the major difference is that **display: inline-block** does not add a line-break after the element, so the element can sit next to other elements.

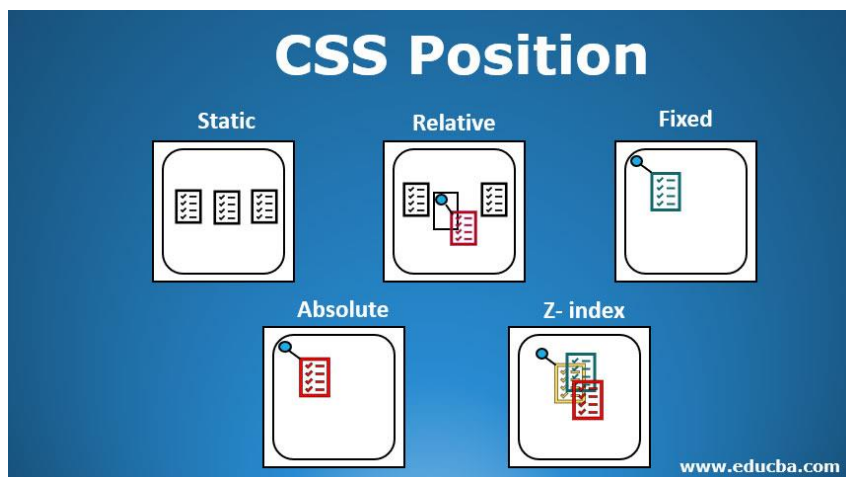
Example

```
span.a {  
  display: inline; /* the default for span */  
  width: 100px;  
  height: 100px;  
  padding: 5px;  
  border: 1px solid blue;  
  background-color: yellow;  
}  
  
span.b {
```

```
display: inline-block;
width: 100px;
height: 100px;
padding: 5px;
border: 1px solid blue;
background-color: yellow;
}

span.c {
display: block;
width: 100px;
height: 100px;
padding: 5px;
border: 1px solid blue;
background-color: yellow;
}
```

➤ The position



The **position** property specifies the type of positioning method used for an element (relative, fixed, absolute or sticky).

The position Property

The **position** property specifies the type of positioning method used for an element.

There are five different position values:

- **relative**
- **fixed**
- **absolute**
- **sticky**

1) **position: relative;**

An element with **position: relative;** is positioned relative to its normal position.

Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

This <div> element has position: relative;

Here is the CSS that is used:

Example

```
div.relative {  
    position: relative;  
    left: 30px;  
    border: 3px solid #73AD21;  
}
```

2) **position: fixed;**

An element with `position: fixed;` is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

A fixed element does not leave a gap in the page where it would normally have been located.

Example

```
div.fixed {  
    position: fixed;  
    bottom: 0;  
    right: 0;  
    width: 300px;  
    border: 3px solid #73AD21;  
}
```

3) **position: absolute;**

An element with `position: absolute;` is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed).

However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

Example

```
div.relative {  
    position: relative;  
    width: 400px;
```

```
height: 200px;
border: 3px solid #73AD21;
}

div.absolute {
  position: absolute;
  top: 80px;
  right: 0;
  width: 200px;
  height: 100px;
  border: 3px solid #73AD21;
}
```

4) **position: sticky;**

An element with `position: sticky;` is positioned based on the user's scroll position.

A sticky element toggles between `relative` and `fixed`, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like `position: fixed`).

Example

```
div.sticky {
  position: sticky;
  top: 0;
  background-color: green;
  border: 2px solid #4CAF50;
}
```

• Overlapping Elements

When elements are positioned, they can overlap other elements.

The **z-index** property specifies the stack order of an element (which element should be placed in front of, or behind, the others).

An element can have a positive or negative stack order:

Example

```
img {  
  position: absolute;  
  left: 0px;  
  top: 0px;  
  z-index: -1;  
}
```



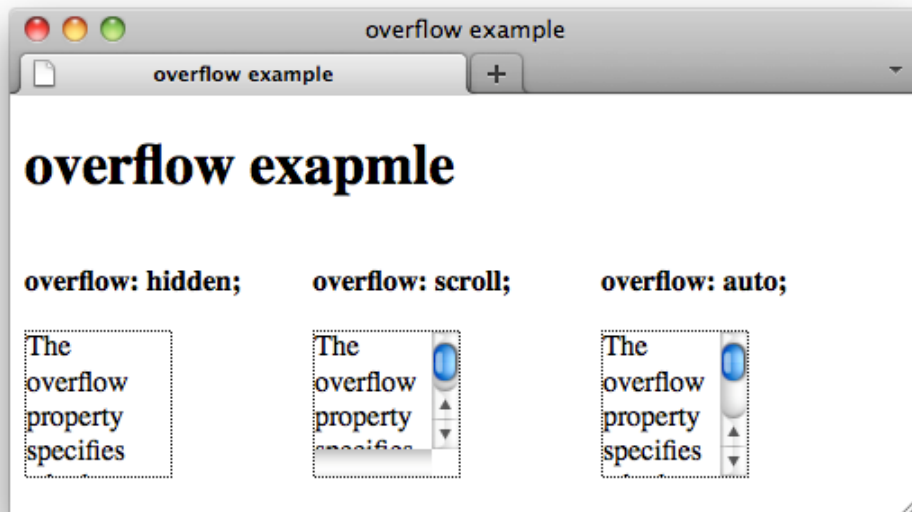
clear Property

The **clear** property specifies on which sides of an element floating elements are not allowed to float.

Example

```
img {  
  float: left;  
}  
  
p.clear {  
  clear: both;  
}
```

➤ Layout – Overflow



The CSS `overflow` property controls what happens to content that is too big to fit into an area.

The `overflow` property has the following values:

- ❖ **visible** - Default. The overflow is not clipped. The content renders outside the element's box
- ❖ **hidden** - The overflow is clipped, and the rest of the content will be invisible
- ❖ **scroll** - The overflow is clipped, and a scrollbar is added to see the rest of the content
- ❖ **auto** - Similar to **scroll**, but it adds scrollbars only when necessary

• overflow: visible

By default, the overflow is **visible**, meaning that it is not clipped and it renders outside the element's box:

Example

```
div {  
  width: 200px;  
  height: 50px;  
  background-color: #eee;  
  overflow: visible;  
}
```

• overflow: hidden

With the **hidden** value, the overflow is clipped, and the rest of the content is hidden:

Example

```
div {  
  width: 200px;  
  height: 50px;  
  background-color: #eee;  
  overflow: hidden;  
}
```

• overflow: scroll

Setting the value to **scroll**, the overflow is clipped and a scrollbar is added to scroll inside the box. Note that this will add a scrollbar both horizontally and vertically (even if you do not need it):

Example

```
div {  
  width: 200px;  
  height: 50px;  
  background-color: #eee;  
  overflow: scroll;  
}
```

- **overflow: auto**

The `auto` value is similar to `scroll`, but it adds scrollbars only when necessary:

Example

```
div {  
  width: 200px;  
  height: 50px;  
  background-color: #eee;  
  overflow: auto;  
}
```



Horizontal & Vertical Align

❖ Center Align Elements

To horizontally center a block element (like `<div>`), use `margin: auto;`

Setting the width of the element will prevent it from stretching out to the edges of its container.

The element will then take up the specified width, and the remaining space will be split equally between the two margins:

Example

```
center {  
  margin: auto;  
  width: 50%;  
  border: 3px solid green;  
  padding: 10px;  
}
```

❖ Center Align Text

To just center the text inside an element, use `text-align: center;`

Example

```
.center {  
  text-align: center;  
  border: 3px solid green;  
}
```

❖ Center an Image

To center an image, set left and right margin to `auto` and make it into a `block` element:

Example

```
img {  
  display: block;  
  margin-left: auto;  
  margin-right: auto;  
  width: 40%;  
}
```

❖ Left and Right Align - Using position

One method for aligning elements is to use `position: absolute;`

Example

```
.right {  
  position: absolute;  
  right: 0px;  
  width: 300px;  
  border: 3px solid #73AD21;  
  padding: 10px;  
}
```

❖ Left and Right Align - Using float

Another method for aligning elements is to use the `float` property:

Example

```
.right {  
  float: right;  
  width: 300px;  
  border: 3px solid #73AD21;  
  padding: 10px;  
}
```

❖ Center Vertically - Using position & transform

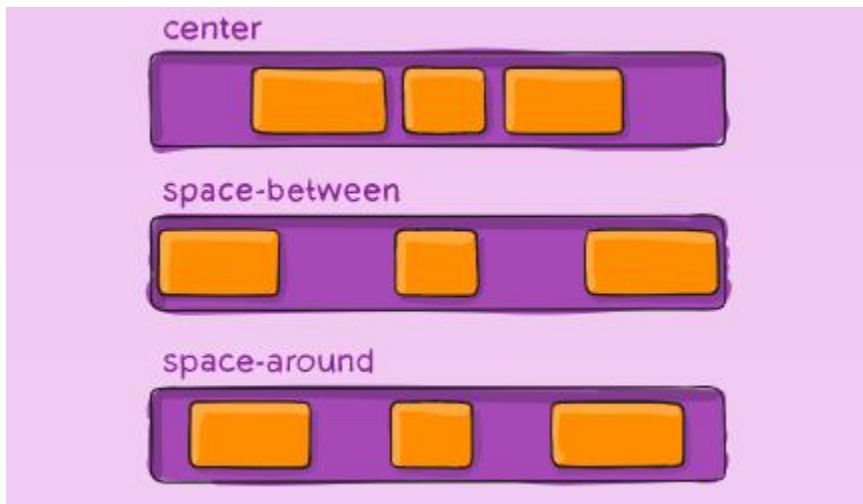
If `padding` and `line-height` are not options, another solution is to use positioning and the `transform` property:

Example

```
.center p {  
  margin: 0;  
  position: absolute;  
  top: 50%;  
  left: 50%;  
  transform: translate(-50%, -50%);  
}
```



Flex



Flexbox Elements

To start using the Flexbox model, you need to first define a flex container.

Example

```
<style>

.flex-container {

  display: flex;

  background-color: DodgerBlue;

}

.flex-container div {

  background-color: #f1f1f1;

  margin: 10px;

  padding: 20px;

  font-size: 30px;

}

</style>
```

• The flex-direction Property

The `flex-direction` property defines in which direction the container wants to stack the flex items.

Example

```
.flex-container {  
  display: flex;  
  flex-direction: column;  
}
```

Example

The `row` value stacks the flex items horizontally (from left to right):

```
.flex-container {  
  display: flex;  
  flex-direction: row;  
}
```

• The flex-wrap Property

The `flex-wrap` property specifies whether the flex items should wrap or not.

Example

The `wrap` value specifies that the flex items will wrap if necessary:

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

• The justify-content Property

The `justify-content` property is used to align the flex items:

Example

The `center` value aligns the flex items at the center of the container:

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

Example

The `space-around` value displays the flex items with space before, between, and after the lines:

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

Example

The `space-between` value displays the flex items with space between the lines:

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

• The align-items Property

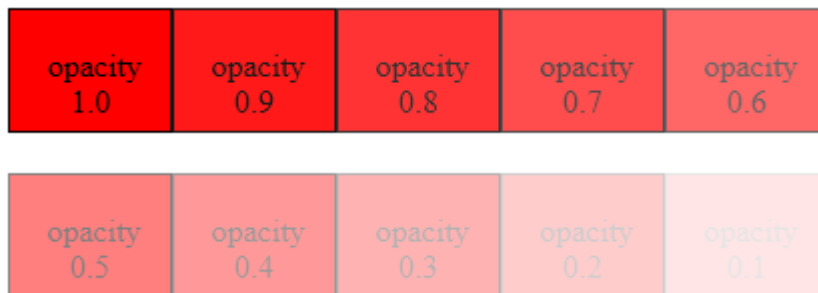
The `align-items` property is used to align the flex items.

Example

The `center` value aligns the flex items in the middle of the container:

```
.flex-container {  
  display: flex;  
  height: 200px;  
  align-items: center;  
}
```

➤ Opacity



The `opacity` property specifies the opacity/transparency of an element.

Example

```
img {  
  opacity: 0.5;  
}
```

➤ CSS :hover Selector

- The `:hover` selector is used to select elements when you mouse over them.

- **Tip:** The `:hover` selector can be used on all elements, not only on links.

Example

```
a:hover {  
  background-color: yellow;  
}
```



Transitions

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

- `transition`
- `transition-delay`
- `transition-duration`

How to Use CSS Transitions?

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

Example

```
div {  
  width: 100px;  
  height: 100px;  
  background: red;  
  transition: width 2s;  
}
```

Change Several Property Values

Example

```
div {  
  transition: width 2s, height 4s;  
}
```

Delay the Transition Effect

The **transition-delay** property specifies a delay (in seconds) for the transition effect.

























Example

```
div {  
  transition-delay: 1s;  
  
  transition-duration: 2s;  
}
```



cursor Property

The **cursor** property specifies the mouse cursor to be displayed when pointing over an element.

 auto	 move	 no-drop	 col-resize
 all-scroll	 pointer	 not-allowed	 row-resize
 crosshair	 progress	 e-resize	 ne-resize
 default	 text	 n-resize	 nw-resize
 help	 vertical-text	 s-resize	 se-resize
 inherit	 wait	 w-resize	 sw-resize

```

.alias {cursor: alias;}
.all-scroll {cursor: all-scroll;}
.auto {cursor: auto;}
.cell {cursor: cell;}
.context-menu {cursor: context-menu;}
.col-resize {cursor: col-resize;}
.copy {cursor: copy;}
.crosshair {cursor: crosshair;}
.default {cursor: default;}
.e-resize {cursor: e-resize;}
.ew-resize {cursor: ew-resize;}
.grab {cursor: grab;}
.grabbing {cursor: grabbing;}
.help {cursor: help;}
.move {cursor: move;}
.n-resize {cursor: n-resize;}
.ne-resize {cursor: ne-resize;}
.nesw-resize {cursor: nesw-resize;}
.ns-resize {cursor: ns-resize;}
.nw-resize {cursor: nw-resize;}
.nwse-resize {cursor: nwse-resize;}
.no-drop {cursor: no-drop;}
.none {cursor: none;}
.not-allowed {cursor: not-allowed;}
.pointer {cursor: pointer;}
.progress {cursor: progress;}
.row-resize {cursor: row-resize;}

```

```
.s-resize {cursor: s-resize;}
.se-resize {cursor: se-resize;}
.sw-resize {cursor: sw-resize;}
.text {cursor: text;}
.url {cursor: url(myBall.cur), auto;}
.w-resize {cursor: w-resize;}
.wait {cursor: wait;}
.zoom-in {cursor: zoom-in;}
.zoom-out {cursor: zoom-out;}
```

➤ outline Property

An outline is a line that is drawn around elements, outside the borders, to make the element "stand out".

The **outline** property is a shorthand property for:

- outline-width
- Outline-style (required)
- Outline-color

Example

```
h2 {
  outline: 5px dotted green;
}

div.a {
  outline: 2px dashed blue;
}
```



Gradients

CSS gradients let you display smooth transitions between two or more specified colors.

CSS defines two types of gradients:

- **Linear Gradients (goes down/up/left/right/diagonally)**
- **Radial Gradients (defined by their center)**

• Linear Gradients



Example

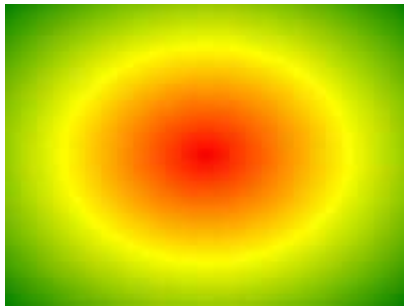
```
#grad {  
  background: linear-gradient(red, yellow);  
}
```

Direction - Left to Right

Example

```
#grad {  
  background-image: linear-gradient(to right, red ,  
yellow);  
}
```

• Radial Gradients



A radial gradient is defined by its center.

To create a radial gradient you must also define at least two color stops.

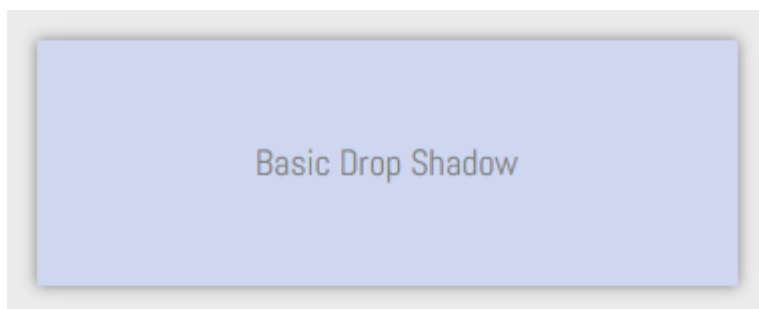
Example

```
#grad {  
  background-image: radial-gradient(red, yellow, green);  
}
```



Box Shadow

CSS box-shadow Property



The CSS `box-shadow` property applies shadow to elements.

In its simplest use, you only specify the horizontal shadow and the vertical shadow:

Example

```
div {  
  box-shadow: 10px 10px;  
}
```

Next, add a color to the shadow:

Example

```
div {  
  box-shadow: 10px 10px grey;  
}
```

Next, add a blur effect to the shadow:

Example

```
div {  
  box-shadow: 10px 10px 5px grey;  
}
```



Forms

• Styling Input Fields

Use the `width` property to determine the width of the input field:

Example

```
input {  
  width: 100%;  
}
```

The example above applies to all `<input>` elements. If you only want to style a specific input type, you can use attribute selectors:

- `input[type=text]` - will only select text fields
- `input[type=password]` - will only select password fields
- `input[type=number]` - will only select number fields

• Padded Inputs

Use the `padding` property to add space inside the text field.

Example

```
input[type=text] {  
  width: 100%;  
  padding: 12px 20px;  
  margin: 8px 0;  
}
```

If you only want a bottom border, use the `border-bottom` property:

Example

```
input[type=text] {  
  border: none;  
  border-bottom: 2px solid red;  
}
```

• Focused Inputs

By default, some browsers will add a blue outline around the input when it gets focus (clicked on). You can remove this behavior by adding `outline: none;` to the input.

Use the `:focus` selector to do something with the input field when it gets focus:

Example

```
input[type=text]:focus {  
  background-color: lightblue;  
}
```

• Animated Search Input

we use the CSS **transition** property to animate the width of the search input when it gets focus. You will learn more about the **transition** property later,

Example

```
input[type=text] {  
  transition: width 0.4s;  
}  
  
input[type=text]:focus {  
  width: 100%;  
}
```

➤ 2D Transforms

2D Transforms Methods

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

- **translate()**
- **rotate()**
- **scaleX()**
- **scaleY()**

- `scale()`
- `skewX()`
- `skewY()`
- `skew()`

• The `translate()` Method

The `translate()` method moves an element from its current position (according to the parameters given for the X-axis and the Y-axis).

Example

```
div {  
  transform: translate(50px, 100px);  
}
```

• The `rotate()` Method

The `rotate()` method rotates an element clockwise or counter-clockwise according to a given degree.

Example

```
div {  
  transform: rotate(20deg);  
}
```

• The `scale()` Method

The `scale()` method increases or decreases the size of an element (according to the parameters given for the width and height).

Example

```
div {  
  transform: scale(2, 3);  
}
```

• The scaleX() Method

The `scaleX()` method increases or decreases the width of an element.

Example

```
div {  
  transform: scaleX(2);  
}
```

• The scaleY() Method

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

Example

```
div {  
  transform: scaleY(3);  
}
```

• The skew() Method

The `skew()` method skews an element along the X and Y-axis by the given angles.

Example

```
div {  
  transform: skew(20deg, 10deg);  
}
```

• The skewX() Method

The `skewX()` method skews an element along the X-axis by the given angle.

Example

```
div {  
  transform: skewX(20deg);  
}
```

• The skewY() Method

The `skewY()` method skews an element along the Y-axis by the given angle.

Example

```
div {  
  transform: skewY(20deg);  
}
```

➤ 3D Transforms

3D Transforms Methods

With the CSS `transform` property you can use the following 3D transformation methods:

- `rotateX()`
- `rotateY()`
- `rotateZ()`

• The `rotateX()` Method



The `rotateX()` method rotates an element around its X-axis at a given degree:

Example

```
#myDiv {  
  transform: rotateX(150deg);  
}
```

• The rotateY() Method



The `rotateY()` method rotates an element around its Y-axis at a given degree:

Example

```
#myDiv {  
  transform: rotateY(150deg);  
}
```

• The rotateZ() Method

The `rotateZ()` method rotates an element around its Z-axis at a given degree:

Example

```
#myDiv {  
  transform: rotateZ(90deg);  
}
```

➤ Animations

CSS allows animation of HTML elements without using JavaScript or Flash!

- `@keyframes`
- `animation-name`
- `animation-duration`
- `animation-delay`
- `animation-iteration-count`
- `animation-direction`

Example

```
/* 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-duration

Example

```
/* The animation code */
@keyframes example {
  0% {background-color: red;}
  25% {background-color: yellow;}
  50% {background-color: blue;}
  100% {background-color: green;}
}

/* The element to apply the animation to */
div {
  width: 100px;
  height: 100px;
  background-color: red;
  animation-name: example;
}
```

```
    animation-duration: 4s;
}
```

❖ Delay an Animation

The `animation-delay` property specifies a delay for the start of an animation.

Example

```
div {
    width: 100px;
    height: 100px;
    position: relative;
    background-color: red;
    animation-name: example;
    animation-duration: 4s;
    animation-delay: 2s;
}
```

❖ Set How Many Times an Animation Should Run

The `animation-iteration-count` property specifies the number of times an animation should run.

Example

```
div {
    width: 100px;
    height: 100px;
    position: relative;
    background-color: red;
    animation-name: example;
    animation-duration: 4s;
    animation-iteration-count: 3;
}
```


❖ Run Animation in Reverse Direction or Alternate Cycles

The `animation-direction` property specifies whether an animation should be played forwards, backwards or in alternate cycles.

The animation-direction property can have the following values:

- `normal` - The animation is played as normal (forwards). This is default
- `reverse` - The animation is played in reverse direction (backwards)
- `alternate` - The animation is played forwards first, then backwards
- `alternate-reverse` - The animation is played backwards first, then forwards

Example

```
div {  
  width: 100px;  
  height: 100px;  
  position: relative;  
  background-color: red;  
  animation-name: example;  
  animation-duration: 4s;  
  animation-direction: reverse;  
}
```

➤ :nth-child() Selector

- The `:nth-child(n)` selector matches every element that is the *n*th child, regardless of type, of its parent.
- *n* can be a number, a keyword, or a formula.

Example

```
p:nth-child(2) {  
  background: red;  
}
```

➤ Border Images

border-image Property



The CSS `border-image` property allows you to specify an image to be used instead of the normal border around an element.

The property has three parts:

1. The image to use as the border
2. Where to slice the image
3. Define whether the middle sections should be repeated or stretched

Example

```
#borderimg {  
  border: 10px solid transparent;
```

```
padding: 15px;
border-image: url(border.png) 30 stretch;
}
```

➤ Media Queries

CSS3 Introduced Media Queries

Media queries in CSS3 extended the CSS2 media types idea: Instead of looking for a type of device, they look at the capability of the device.

Example

```
@media (min-width: 480px) {
  body {
    background-color: lightgreen;
  }
}
```

➤ Background Image Opacity with Before Selector

Example:-

```
header::before{
    background: url
('https://source.unsplash.com/collection/190727/1600x900') no-repeat
center center/cover;
```

```
content: "";  
position: absolute;  
top:0;  
left: 0;  
width: 100%;  
height: 100%;  
z-index: -1;  
opacity: 0.3;  
}
```

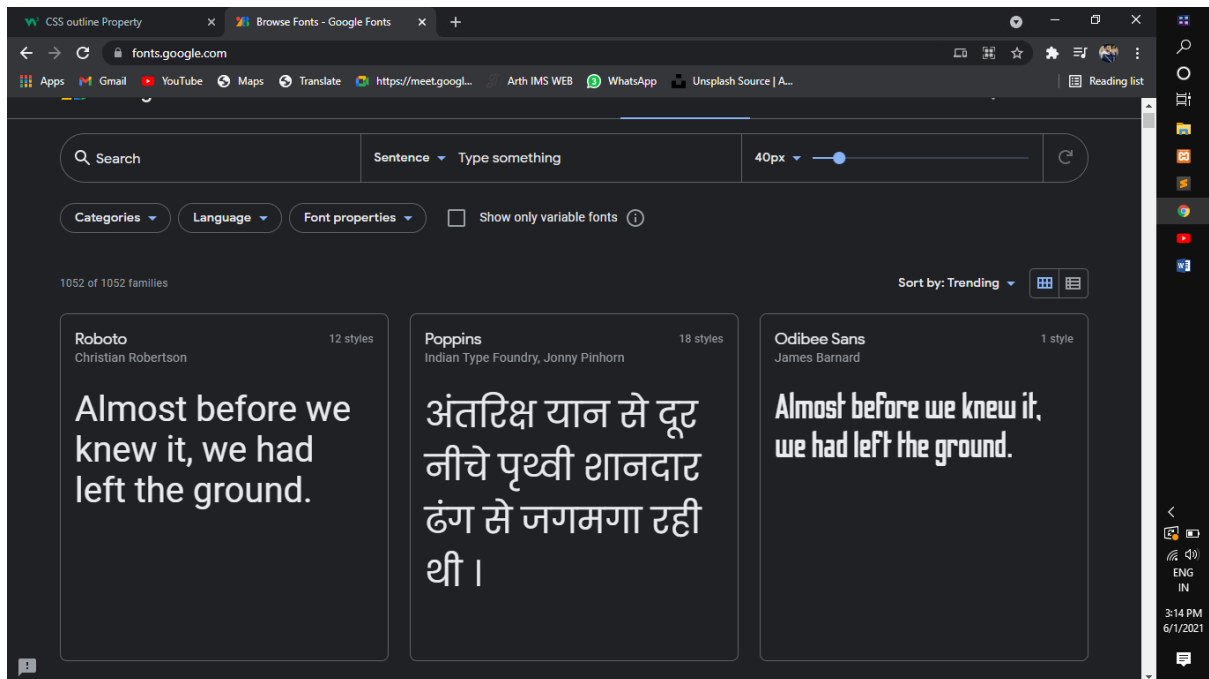
```
<header>  
  <nav class="navbar">  
    <ul class="navigation">  
      <li class="item">Home</li>  
      <li class="item">About</li>  
      <li class="item">Services</li>  
      <li class="item">Contact Us</li>  
    </ul>  
  </nav>  
</header>
```

➤ Google Font

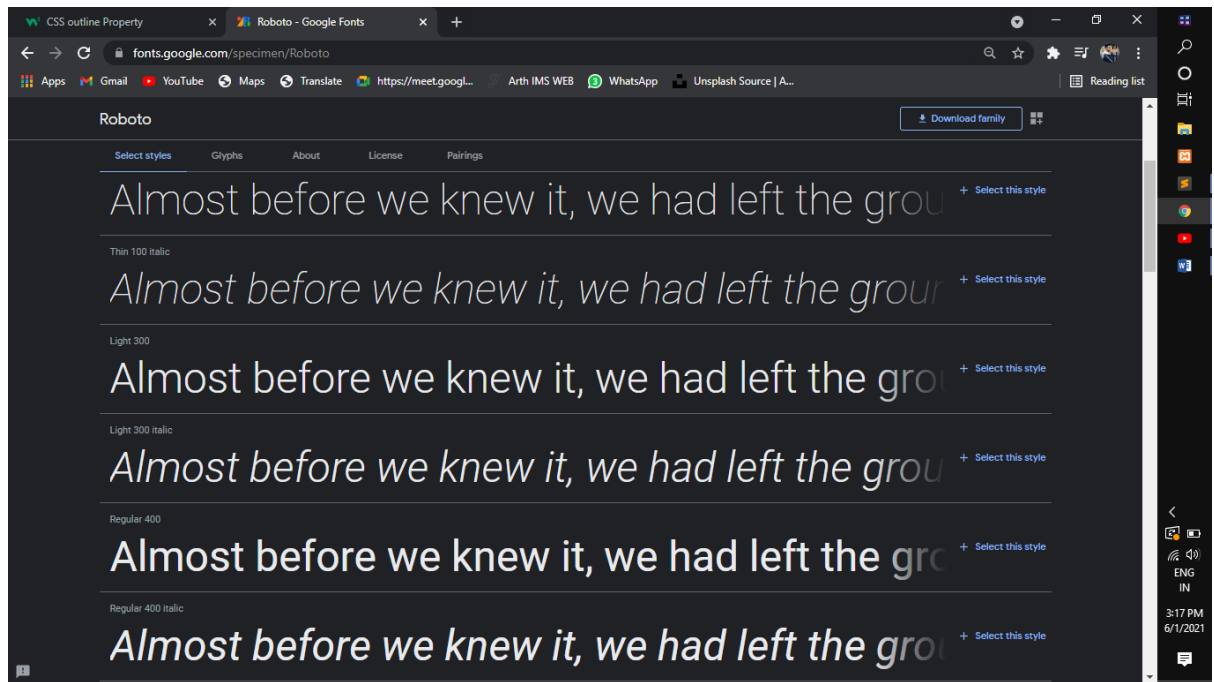
Steps To Use :-

1. Go to site :-

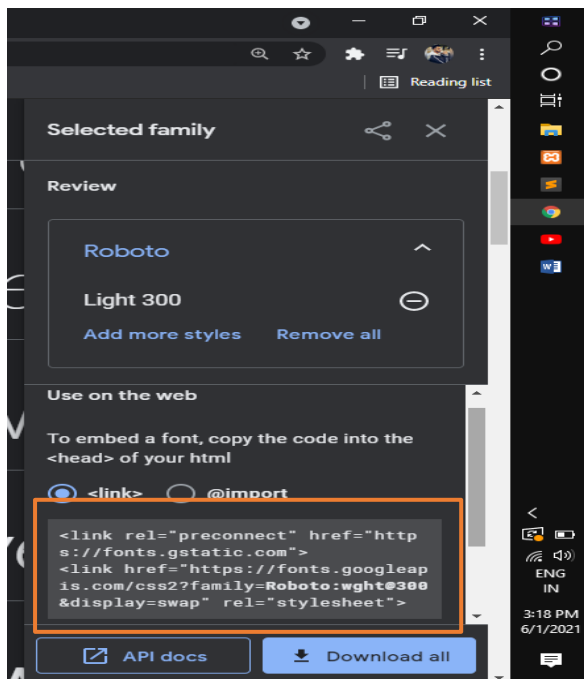
<https://fonts.google.com/>



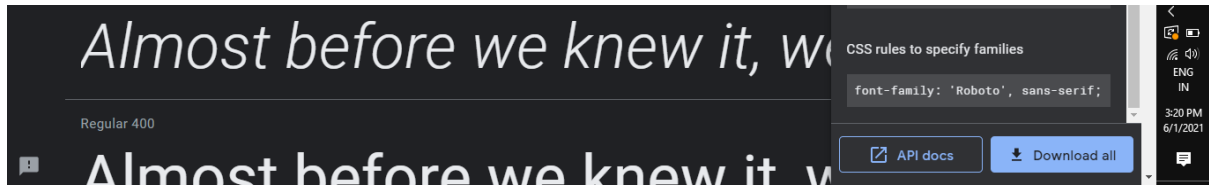
2. Select Font style



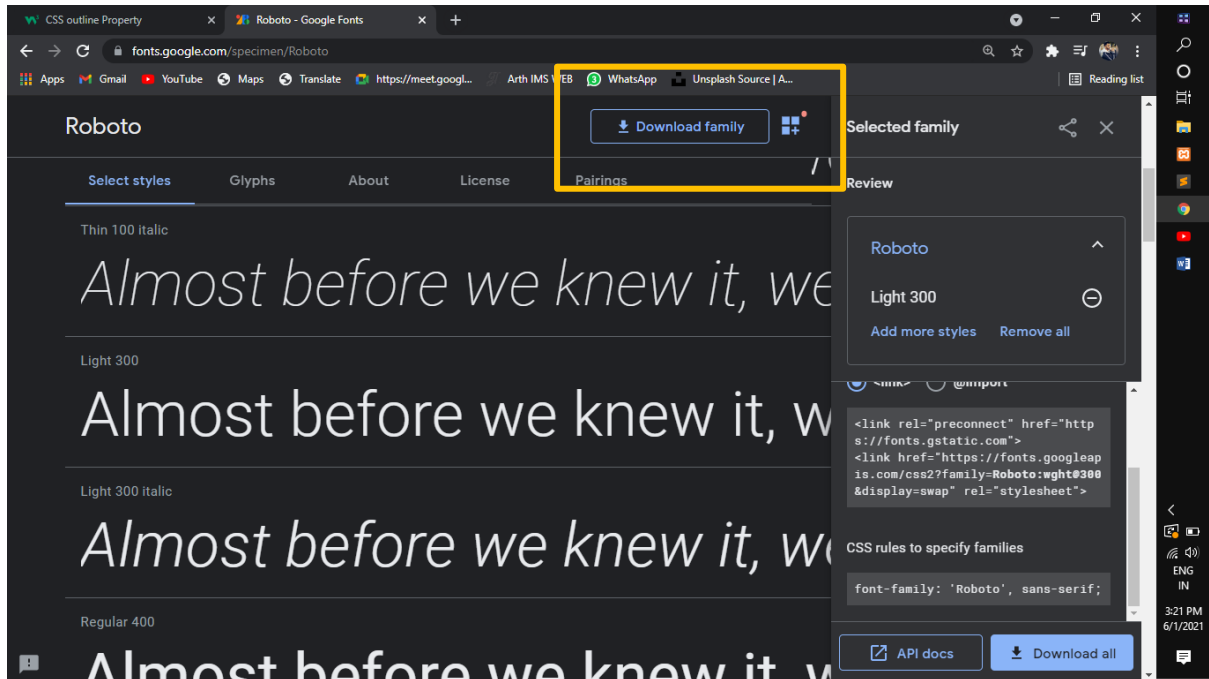
3. Copy link and place in your
<head> tag :-



4. Then Copy Css Property:-



5. Download Method :- Simple Click Download Family icon



BE SAFE AND KEEP PRACTICING

