**1. What are different ways to apply styles to a Web page?**

There are four ways to apply style to a Web page.

* 1. **Inline CSS**: HTML elements may have CSS applied to them via the STYLE attribute.For Example: If You have <p> element into webpage, you can apply inline style likeshows in example.<p style=”font-size: 12px;  color: #000000;”>Test </p>
* **Embedded CSS**: CSS may be embedded in a Web page by placing the code    in a STYLE element within the HEAD element.For Example: If You have <h2> element into webpage, you can apply embedded style like shows in example.

<head>  
<style type=”text/css”>  
h2 {  
font-size: 16px;  
color: #2d2d2d;  
font-weight: 900;  
}  
</style>  
</head>

* **Linked CSS**: CSS can be placed in an external file (a simple text file containing         CSS) and linked via the link element.You can apply style to webpage using external file as shown in example.  
  <link rel=”stylesheet” href=”custom/custom.css” type=”text/css” media=”screen, projection” />
* **Imported CSS**: Another way to utilize external CSS files via @import.<style>  
  @import url(‘/css/styles.css’);  
  </style>  
  Put then your “styles.css” document can contain calls to any number of additional  
  style sheets:  
  @import url(‘/css/typography.css’);  
  @import url(‘/css/layout.css’);  
  @import url(‘/css/color.css’);

**2. How do CSS precedence/cascading rules work? How does the !important directive affect the rules?**

CSS style rules “cascade” in the sense that they follow an order of precedence. Global style rules apply first to HTML elements, and local style rules override them. For example, a style defined in a style element in a webpage overrides a style defined in an external style sheet. Similarly, an inline style that is defined in an HTML element in the page overrides any styles that are defined for that same element elsewhere.

The !important rule is a way to make your CSS cascade but also have the rules you feel are most crucial always be applied. A rule that has the !important property will always be applied no matter where that rule appears in the CSS document. So if you wanted to make sure that a property always applied, you would add the !important property to the tag. So, to make the paragraph text always red, in the above example, you would write:

p { color: #ff0000 !important; }

p { color: #000000; }

**3. What is a class? What is an ID?**

**A class** is a style (i.e., a group of CSS attributes) that can be applied to one or more HTML elements. This means it can apply to instances of the same element or instances of different elements to which the same style can be attached. Classes are defined in CSS using a period followed by the class name. It is applied to an HTML element via the class attribute and the class name.

The following snippet shows a class defined, and then it being applied to an HTML DIV element.

.test {font-family: Helvetica; font-size: 20; background: black;}

<div class =”test”><p>test</p></div>

Also, you could define a style for all elements with a defined class. This is demonstrated with the following code that selects all P elements with the column class specified.

p.column {font-color: black;}

**An ID selector** is a name assigned to a specific style. In turn, it can be associated with one HTML element with the assigned ID. Within CSS, ID selectors are defined with the # character followed by the selector name.

The following snippet shows the CSS example1 defined followed by the use of an HTML element’s ID attribute, which pairs it with the CSS selector.

#example1: {background: blue;}

<div id=”example1″></div>

**4. What is the difference between an ID selector and CLASS?**

An ID selector identifies and sets style to only one occurrence of an element, while CLASS can be attached to any number of elements.

**5. What is Contextual Selector?**

Contextual selector addresses specific occurrence of an element. It is a string of individual selectors separated by white space (search pattern), where only the last element in the pattern is addressed providing it matches the specified contex

**6. What is Grouping?**

When more than one selector shares the same declaration, they may be grouped together via a comma-separated list; this allows you to reduce the size of the CSS (every bit and byte is important) and makes it more readable. The following snippet applies the same background to the first three heading elements.

h1, h2, h3 {background: red;}

**7. What are Child Selectors?**

A child selector is used when you want to match an element that is the child of another specific element. The parent and child selectors are separated by spaces. The following selector locates an unordered list element within a paragraph element and makes a text within that element bold.

p > ul {font-weight: bold;}

**8. What are Pseudo Classes?**

Pseudo classes allow you to identify HTML elements on characteristics (as opposed to their name or attributes). The classes are specified using a colon to separate the element name and pseudo class. A good example is the :link and :visited pseudo classes for the HTML A element. Another good example is first-child, which finds an element’s first child element.

The following CSS makes all visited links red and green, the actual link text becomes yellow when the mouse pointer is positioned over it, and the text of the first element of a paragraph is bold.

a:link {font-color: red;}  
a:visited {font-color: green;}  
a:hover {font-color: yellow;}

p.first-child {font-weight: bold;}

**9. What are some ways you might target IE (or IE6) only, without affecting other browsers?**

Below are the example for browser specific Style Sheet, which targets defined browser.

#### Target ALL VERSIONS of IE

<!--[if IE]>

<link rel="stylesheet" type="text/css" href="all-ie-only.css" />

<![endif]-->

#### Target everything EXCEPT IE

<!--[if !IE]><!-->

<link rel="stylesheet" type="text/css" href="not-ie.css" />

<!--<![endif]-->

#### Target IE 6 ONLY

<!--[if IE 6]>

<link rel="stylesheet" type="text/css" href="ie6.css" />

<![endif]-->

**10. What’s the difference between an inline element and a block element?**

A block-level element is an element that creates large blocks of content like  paragraphs or page divisions. They start new lines of text when you use them, and can contain other blocks as well as inline elements and text or data.

An inline element is an element that define text or data in the document like STRONG makes the enclosed text strongly emphasized and Q says the enclosed text is a quotation. They don’t start new lines when you use them, and they generally only contain other inline tags and text or data. Or they include nothing at all, like the BR tag.

Some properties that inline elements ignore include:

* width and height
* max-width and max-height
* min-width and min-height

The CSS property display lets you change an inline property to block or a block to inline or either of them to not display at all.

display: block;

display:inline;

display:none;

### ****When to Change the Display Property****

* **Horizontal list menus** - Lists are block-level elements, but if you want your menu to display horizontally, you need to convert the list to an inline element, so that newlines aren’t added between each list item.
* **Headers in the text** - Sometimes you might want a header to remain in the text, but have the HTML header values. Changing the h1-h6 values to inline will let the following text flow next to it.
* **Removing the element** - And of course, if you want to remove the element completely from the document flow, you can set the display to none.

**11. Explain how the CSS box model works. Draw a box and then explain what the border is, what the margin is, what the padding is, etc.? If you assign a width: or height: to something, what specifically does that refer to?**

All HTML elements can be considered as boxes. In CSS, the term “box model” is used when talking about design and layout.

The CSS box model is essentially a box that wraps around HTML elements, and it consists of: margins, borders, padding, and the actual content.

The box model allows us to place a border around elements and space elements in relation to other elements.

[](http://www.gophoto.it/view.php?i=http://dglobaltech.com/wp-content/uploads/2013/05/boxdim.png)

Explanation of the different parts:

* **Margin** - Clears an area around the border. The margin does not have a background color, it is completely transparent
* **Border** - A border that goes around the padding and content. The border is affected by the background color of the box
* **Padding** - Clears an area around the content. The padding is affected by the background color of the box
* **Content** - The content of the box, where text and images appear

Here is the snippet to draw box model to your web page.

<head>  
<style>  
div.test  
{  
width:220px;  
padding:10px;  
border:5px solid gray;  
margin:0px;  
}  
</style>  
</head>

<body>

<div class=”test”>The picture above is 250px wide. The total width of this element is   also 250px.</div>

</body>

**12. Explain vertical margin collapsing.**

Top and bottom margins of blocks are sometimes combined (collapsed) into a single margin whose size is the largest of the margins combined into it, a behavior known as **margin collapsing**.

Margin collapsing occurs in three basic cases:

Adjacent siblings

The margins of adjacent siblings are collapsed (except when the later sibling needs to be [cleared](https://developer.mozilla.org/en-US/docs/CSS/clear) past floats).

**Parent and first/last child**

If there is no border, padding, inline content, or clearance to separate the margin-top of a block with the margin-top of its first child block, or no border, padding, inline content, height, min-height, or max-height to separate the margin-bottom of a block with the margin-bottom of its last child, then those margins collapse. The collapsed margin ends up outside the parent.

Empty blocks

If there is no border, padding, inline content, height, or min-height to separate a block’s margin-top from its margin-bottom, then its top and bottom margins collapse.

vertical margins collapse between certain boxes.

To understand how vertical margins collapse lets start with a basic html and CSS example.

## Adjacent Elements With Positive Margin

Vertical adjoining margins collapses. If two elements has positive vertically adjoining margin than only the maximum of both will take effect.

### CSS Code

|  |  |
| --- | --- |
| 1  2  3 | #parent{ background-color:yellow; width:300px; }  #red {background-color:red; height:50px; margin-bottom:30px;}  #blue {background-color:blue; height:50px; margin-top:20px;} |

### Browser Result

[](http://www.gophoto.it/view.php?i=http://960development.com.s142956.gridserver.com/wp-content/uploads/2010/08/02_margin.png)

Margins of [floating](https://developer.mozilla.org/en-US/docs/CSS/float) and [absolutely positioned](https://developer.mozilla.org/en-US/docs/CSS/position) elements never collapse.

There are other situations where elements do not have their margins collapsed:

* inline-block elements
* elements with overflow set to anything other than visible (They do not collapse margins with their children.)
* cleared elements (They do not collapse their top margins with their parent block’s bottom margin.)
* the root element

**13. How do you float an element, and what does that mean? What’s Clearing?**

With CSS float, an element can be pushed to the left or right, allowing other elements to wrap around it.

Float is very often used for images, but it is also useful when working with layouts.

Elements are floated horizontally, this means that an element can only be floated left or right, not up or down.

A floated element will move as far to the left or right as it can. Usually this means all the way to the left or right of the containing element.

Below is the example, which shows using css float property you can set image to the right of your web page.

<head>  
<style>  
img  
{  
float:right;  
}  
</style>  
</head>

<body>

<p>

<img src=”logocss.gif” width=”95″ height=”84″ />

This is some text. This is some text. This is some text.This is some text. This is some text. This is some text.This is some text. This is some text. This is some text.This is some text. This is some text. This is some text.This is some text. This is some text. This is some text.This is some text. This is some text. This is some text.This is some text. This is some text. This is some text.This is some text. This is some text. This is some text.This is some text. This is some text. This is some text.This is some text. This is some text. This is some text.</p></body>

**14. What’s the difference between position: relative, position: fixed & position: absolute?**

Here, I’ve listed basic difference between css positioning properties.

**Position-Relative**. This type of positioning is probably the most confusing and misused. What it really means is “relative to itself”. If you set position: relative;on an element but no other positioning attributes (top, left, bottom or right), it will no effect on it’s positioning at all, it will be exactly as it would be if you left it as position: static; But if you DO give it some other positioning attribute, say, top: 10px;, it will shift it’s position 10 pixels DOWN from where it would NORMALLY be.

**Position-Absolute**. This is a very powerful type of positioning that allows you to literally place any page element exactly where you want it. You use the positioning attributes top, left bottom and right to set the location. Remember that these values will be relative to the next parent element. If there is no such parent, it will default all the way back up to the <html> element itself meaning it will be placed relatively to the page itself.

**Position-Fixed**. This type of positioning is fairly rare but certainly has its uses. A fixed position element is positioned relative to the viewport, or the browser window itself. The viewport doesn’t change when the window is scrolled, so a fixed positioned element will stay right where it is when the page is scrolled.

**15. What does z-index do?**

The z-index property specifies the stack order of an element.

An element with greater stack order is always in front of an element with a lower stack order.

Here is the Example, where using z-index property you can display in front of image.

<head>  
<style>  
img  
{  
position:absolute;  
left:0px;  
top:0px;  
z-index:-1;  
}  
</style>  
</head>

<body>  
<h1>This is a heading</h1>  
<img src=”w3css.gif” width=”100″ height=”140″ />  
<p>Because the image has a z-index of -1, it will be placed behind the text.</p>  
</body>