Internal Assessment Test 1 – May 2022
Solution

| Sub: | Web Technology & its Applications | | | | | Sub Code: | 18CS63 | Branch: | CSE |
|---|---|---|---|---|---|---|---|---|---|
| Date: | 10-05-2022 | Duration: | 90 min's | Max Marks: | 50 | Sem / Sec: | A, B & C | | OBE |

Answer any FIVE FULL Questions

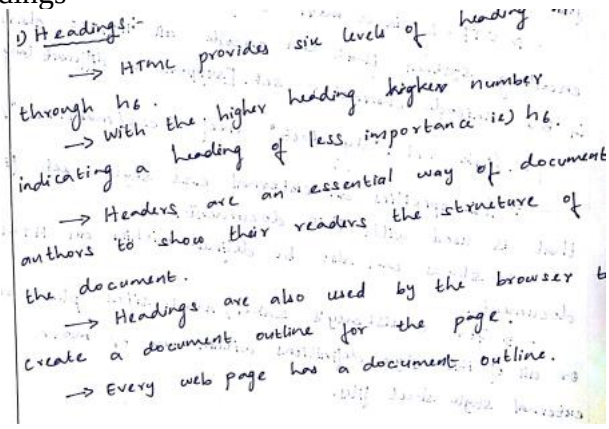
| | | MARKS | CO | RBT |
|---|---|---|---|---|
| 1 (a) | Briefly explain the history of markup languages.<br>HTML books invariably begin with a brief history of HTML. Such a history might begin with the ARPANET of the late 1960s, jump quickly to the first public specification of the HTML by Tim Berners-Lee in 1991, and then to HTML's codification by the World Wide Web Consortium (better known as the W3C) in<br>1997. Some histories of HTML might also tell stories about the Netscape Navigator and Microsoft Internet Explorer of the early and mid-1990s, a time when intrepid developers working for the two browser manufacturers ignored the W3C and brought forward a variety of essential new tags (such as, for instance, the <table>tag), and features such as CSS and JavaScript, all of which have been essential to the growth and popularization of the web. Perhaps in reaction to these manufacturer innovations, in 1998 the W3C froze<br>the HTML specification at version 4.01. This specification begins by stating: To publish information for global distribution, one needs a universally understood language, a kind of publishing mother tongue that all computers may potentially understand. The publishing language used by the World Wide Web is HTML<br>(from HyperText Markup Language).As one can see from the W3C quote, HTML is defined as a markup language.A markup language is simply a way of annotating a document in such a way as to make the annotations distinct from the text being annotated. Markup languages such as HTML, Tex, XML, and XHTML allow users to control how text and visual elements will be laid out and displayed. The term comes from the days of print, when editors would write instructions on manuscript pages that might be revision<br>instructions to the author or copy editor. You may very well have been the recipient of markup from caring parents or concerned teachers at various points in your past,At its simplest, markup is a way to indicate information about the content that is distinct from the content. This "information about content" in HTML is implemented via tags (or more formally, HTML elements, but more on that later). The markup in Figure 2.1 consists of the red text and the various circles and arrows and the little yellow sticky notes. HTML does the same thing but uses textual tags. In addition to specifying "information about content" many markup languages are able to encode information how to display the content for the end user. These presentation semantics can be as simple as specifying a bold weight font for certain words, and were a part of the earliest HTML specification. Although combining semantic markup with presentation markup is no longer permitted in HTML5,"formatting the content" for display remains a key reason why HTML was widely adopted. | [05] | CO1 | L2 |
| (b) | Write short notes on XHTML and HTML5.<br>Instead of growing HTML, the W3C turned its attention in the late 1990s to a new specification called XHTML 1.0, which was a version of HTML that used stricter XML (extensible markup language) syntax rules (see Background next). But why was "stricter" considered a good thing? Perhaps the best | [05] | CO1 | L2 |

analogy might
be that of a strict teacher. When one is prone to bad habits and is learning something difficult in school, sometimes a teacher who is more scrupulous about the need to finish daily homework may actually in the long run be more beneficial than a more permissive and lenient teacher. As the web evolved in the 1990s, web browsers evolved into quite permissive and lenient programs. They could handle sloppy HTML, missing or malformed tags, and other syntax errors. However, it was somewhat unpredictable how each browser would
handle such errors. The goal of XHTML with its strict rules was to make page rendering more predictable by forcing web authors to create web pages without syntax errors. To help web authors, two versions of XHTML were created: XHTML 1.0 Strict and XHTML 1.0 Transitional. The strict version was meant to be rendered by a browser using the strict syntax rules and tag support described by the W3C XHTML 1.0 Strict specification; the transitional recommendation is a more forgiving flavor of XHTML, and was meant to act as a temporary transition to the eventual global adoption of XHTML Strict. The payoff of XHTML Strict was to be predictable and standardized web documents. Indeed, during much of the 2000s, the focus in the professional web development community was on standards: that is, on limiting oneself to the W3C specification for XHTML. A key part of the standards movement in the web development community of the 2000s was the use of HTML validators (see Figure 2.2) as a means of verifying that a web page's markup followed the rules for XHTML Transitional or Strict. Web developers often placed proud images on their sites to tell the world at large that their site followed XHTML rules (and also to communicate their support for web standards).
Yet despite the presence of XHTML validators and the peer pressure from book authors, actual web browsers tried to be forgiving when encountering badly formed HTML so that pages worked more or less how the authors intended regardless of whether a document was XHTML valid or not. In the mid-2000s, the W3C presented a draft of the XHTML 2.0 specification.
It proposed a revolutionary and substantial change to HTML. The most important
was that backwards compatibility with HTML and XHTML 1.0 was dropped. Browsers would become significantly less forgiving of invalid markup. The XHTML 2.0 specification also dropped familiar tags such as <img>, <a>, <br>, and numbered headings such as <h1>. Development on the XHTML 2.0 specification dragged on HTML5
At around the same time the XHTML 2.0 specification was being developed, a group of developers at Opera and Mozilla formed the WHATWG (Web Hypertext Application Technology Working Group) group within the W3C. This group was not convinced that the W3C's embrace of XML and its abandonment of backwards-compatibility was the best way forward for the web. Thus the WHATWG charter announced:
"The Web Hypertext Applications Technology working group therefore intends to address the need for one coherent development environment for Web applications, through the creation of technical specifications that are intended to be implemented in mass-market Web browsers." That is, WHATWG was focused less on semantic purity and more on the web as it actually existed. As well, unlike the large membership of the W3C, the WHATWG group was very small and led by Ian Hickson. As a consequence, the work at WHATWG progressed quickly, and eventually, by 2009, the W3C stopped work on XHTML 2.0 and instead adopted the work done by WHATWG and named it HTML5.
There are three main aims to HTML5:

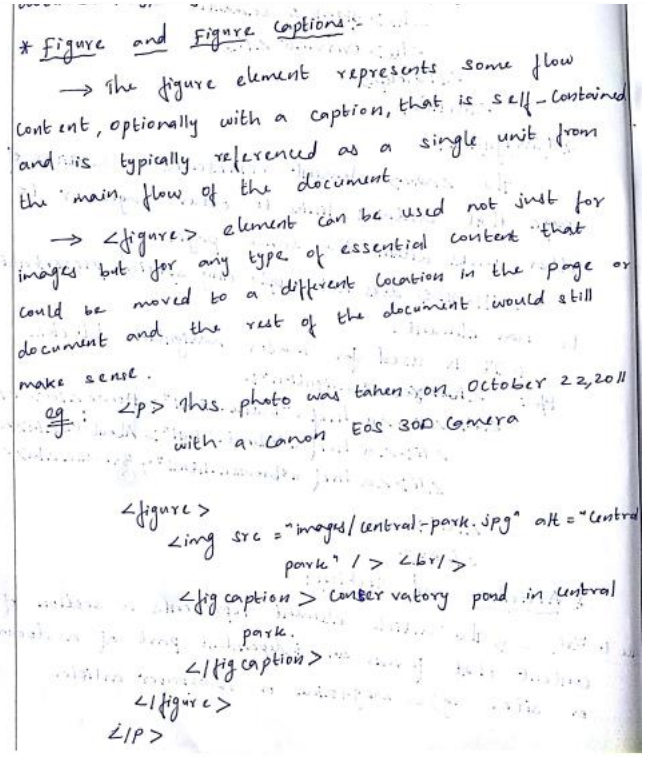
1. Specify unambiguously how browsers should deal with invalid markup.
2. Provide an open, nonproprietary programming framework (via JavaScript) for creating rich web applications.
3. Be backwards compatible with the existing web. While parts of the HTML5 are still being finalized, all of the major browser manufacturers have at least partially embraced HTML5. Certainly not all browsers
and all versions support every feature of HTML5. This is in fact by design. HTML in HTML5 is now a living language: that is, it is a language that evolves and develops over time. As such, every browser will support a gradually increasing subset of HTML5 capabilities. In late September 2012, the W3C announced that they
planned to have the main elements of the HTML5 specification moved to Recommendation status (i.e., the specification would be finalized in terms of features) by late 2014, and the less stable parts of HTML5 moved to HTML5.1 (with a tentative completion date of 2016). This certainly creates complications for web developers. Does one only use HTML elements that are universally supported by all browsers, or all the newest elements supported only by the most recent browsers, or . . . something in between?
This is an interesting question as well for the authors of this textbook. Should we cover only what is supported by the XHTML 1.0 standard or should we cover more of the features in HTML5?
In this text, we have taken the position that HTML5 is not only the future but the present as well. As such, this book assumes that you are using an HTML5 browser. This is not an unreasonable assumption since as of February 2013, a very large majority of web requests are from browsers that have at least partial support
of the main features of HTML5

| | | | | |
|---|---|---|---|---|
| 2 (a) | Explain the following HTML5 tags with example.<br>(i) headings (ii) figure and figure caption (iii)Inline elements (iv)image (v) division | [10] | CO2 | L2 |

(i) headings




(ii) figure and figure caption



* Figure and Figure captions :-

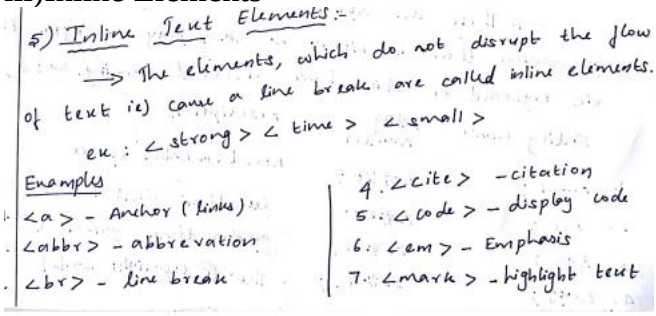
→ The figure element represents some flow content, optionally with a caption, that is self-contained and is typically referenced as a single unit from the main flow of the document.

→ <figure> element can be used not just for images but for any type of essential content that could be moved to a different location in the page or document and the rest of the document would still make sense.

eg:  This photo was taken on October 22, 2011 with a Canon EOS 30D Camera

<figure>
<img src = "images/central-park.jpg" alt = "Central park" /> <br/>
<figcaption> Conservatory pond in central park.
</figcaption>
</figure>

iii)Inline Elements



5) Inline Text Elements :-

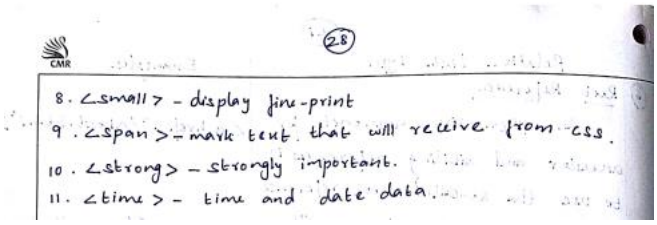
→ The elements, which do not disrupt the flow of text ie) cause a line break are called inline elements.

ex :  <time> <small>

Examples

<a> - Anchor (links)
<abbr> - abbreviation
<br> - line break
4. <cite> - citation
5. <code> - display code
6.  - Emphasis
7. <mark> - highlight text

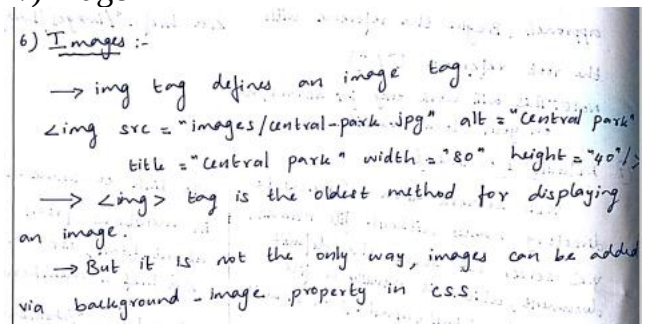






8. <small> - display fine-print
9.  - mark text that will receive from CSS.
10.  - strongly important.
11. <time> - time and date data.

iv)Image



6) Images :-

→ img tag defines an image tag.

<img src = "images/central-park.jpg" alt = "Central park" title = "Central park" width = "80" height = "40"/>

→ <img> tag is the oldest method for displaying an image.

→ But it is not the only way, images can be added via background-image property in CSS.

v) 

This element is a container element and is used to create a logical grouping of content (text and other HTML elements, including containers such as  and other  elements). The  element has no intrinsic presentation; it is frequently used in contemporary CSS-based layouts to mark out sections.

Example:

By Ricardo on <time>September 15, 2015</time>

```
<p>Easy on the HDR buddy.</p>
</div>
<div>
<p>By Susan on <time>October 1, 2015</time></p>
<p>I love Central Park.</p>
</div>
<p><small>Copyright &copy; 2015 Share Your Travels</small></p>
</body>
```

| | | | | |
|---|---|---|---|---|
| 3 (a) | List the different selectors available in CSS and explain in detail | [10] | CO3 | L2 |

## 1 Element Selectors

**Element selectors** select all instances of a given HTML elementYou can select all elements by using the **universal element selector**, which is the * (asterisk) character. You can select a group of elements by separating the different element names with commas. This is a sensible way to reduce the size and complexity of your CSS files, by combining multiple identical rules into a single rule.

### Class Selectors

A **class selector** allows you to simultaneously target different HTML elements regardless of their position in the document tree. If a series of HTML elements have been labeled with the same class attribute value, then you can target them for styling by using a class selector, which takes the form: period (.) followed by the class name.

Listing 3.5 illustrates an example of styling using a class selector. The result in the browser is shown in Figure 3.4.

```
/* commas allow you to group selectors */
p, div, aside {
   margin: 0;
   padding: 0;
}
/* the above single grouped selector is equivalent to the
   following: */
p {
   margin: 0;
   padding: 0;
}
div {
   margin: 0;
   padding: 0;
}
aside {
   margin: 0;
   padding: 0;
}
```

LISTING 3.4 Sample grouped selector

then you can target it for styling by using an id selector, which takes the form: pound/hash (#) followed by the id name. Listing 3.6 illustrates an example of styling using an id selector. The result in the browser is shown in Figure 3.5.

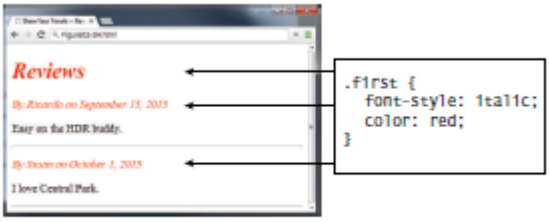



FIGURE 3.4 Class selector example in browser

```
<head lang="en">
   <meta charset="utf-8">
   <title>Share Your Travels -- New York - Central Park</title>
   <style>
         #latestComment {
         font-style: italic;
         color: red;
         }
</style>
</head>
<body>
   <h1>Reviews</h1>
   <div id="latestComment">
      <p>By Ricardo on <time>September 15, 2015</time></p>
      <p>Easy on the HDR buddy.</p>
   </div>
   <hr/>
   <div>
      <p>By Susan on <time>October 1, 2015</time></p>
      <p>I love Central Park.</p>
   </div>
   <hr/>
</body>
```

LISTING 3.6 Id selector example

## 3 Id Selectors

An **id selector** allows you to target a specific element by its id attribute regardless of its type or position. If an HTML element has been labeled with an id attribute, then you can target it for styling by using an id selector, which takes the form: pound/hash (#) followed by the id name.

## 4 Attribute Selectors

An **attribute selector** provides a way to select HTML elements either by the presence of an element attribute or by the value of an attribute. This can be a very powerful technique, but because of uneven support by some of the browsers, not all web authors have used them. Attribute selectors can be a very helpful technique in the styling of hyperlinks and images. For instance, perhaps we want to make it more obvious to the user when a pop-up tooltip is available for a link or image. We can do this by using the following attribute selector: [title] { … } This will match any element in the document that has a title attribute.

## 5 Pseudo-Element and Pseudo-Class Selectors

A **pseudo-element selector** is a way to select something that does not exist explicitly as an element in the HTML document tree but which is still a recognizable selectable object. For instance, you can select the first line or first letter of any HTML element using a pseudo-element selector. A **pseudo-class selector** does apply to an HTML element, but targets either a particular state or, in CSS3, a variety of family relationships. Table 3.5 lists some of the more common pseudo-class and pseudoelement selectors. The most common use of this type of selectors is for targeting link states. By default, the browser displays link text blue and visited text links purple. Listing 3.8
illustrates the use of pseudo-class selectors to style not only the visited and unvisited link colors, but also the hover color, which is the color of the link when the mouse is over the link. Do be aware that this state does not occur on touch screen devices.
Note the syntax of pseudo-class selectors: the colon (:) followed by the pseudo-class selector name. Do be aware that a space is *not* allowed after the colon. Believe it or not, the order of these pseudo-class elements is important. The :link and :visited pseudo-classes should appear before the others. Some developers use a mnemonic to help them remember the order. My favorite is “Lord Vader,

Former Handle Anakin" for Link, Visited, Focus, Hover, Active.

| Selector | Type | Description |
|---|---|---|
| a:link | pseudo-class | Selects links that have not been visited |
| a:visited | pseudo-class | Selects links that have been visited |
| :focus | pseudo-class | Selects elements (such as text boxes or list boxes) that have the input focus. |
| :hover | pseudo-class | Selects elements that the mouse pointer is currently above. |
| :active | pseudo-class | Selects an element that is being activated by the user. A typical example is a link that is being clicked. |
| :checked | pseudo-class | Selects a form element that is currently checked. A typical example might be a radio button or a check box. |
| :first-child | pseudo-class | Selects an element that is the first child of its parent. A common use is to provide different styling to the first element in a list. |
| :first-letter | pseudo-element | Selects the first letter of an element. Useful for adding drop-caps to a paragraph. |
| :first-line | pseudo-element | Selects the first line of an element. |

TABLE 3.5 Common Pseudo-Class and Pseudo-Element Selectors

```
<head>
   <title>Share Your Travels</title>
   <style>
      a:link {
      text-decoration: underline;
      color: blue;
    }
      a:visited {
      text-decoration: underline;
      color: purple;
    }
      a:hover {
      text-decoration: none;
      font-weight: bold;
    }
      a:active {
      background-color: yellow;
    }
   </style>
</head>
<body>
    <p>Links are an important part of any web page. To learn more about
       links visit the <a href="#">W3C</a> website.</p>
   <nav>
     <ul>
       <li><a href="#">Canada</a></li>
       <li><a href="#">Germany</a></li>
       <li><a href="#">United States</a></li>
     </ul>
   </nav>
</body>
```

LISTING 3.8 Styling a link using pseudo-class selectors

## 6 Contextual Selectors

A **contextual selector** (in CSS3 also called **combinators**) allows you to select elements based on their *ancestors*, *descendants*, or *siblings*. That is, it selects elements based on their context or their relation to other elements in the document tree. While some of these contextual selectors are used relatively infrequently, almost all web authors find themselves using descendant selectors.A **descendant selector** matches all elements that are contained within another element. The character used to indicate descendant selection is the space character.

**Descendant** A specified element that is contained somewhere within another specified element. div p Selects a <p> element that is contained somewhere within a  element. That is, the <p> can be any descendant, not just a child. **Child** A specified element that is a direct child of the specified element.

div>h2

Selects an <h2> element that is a child of a  element.
**Adjacent sibling** A specified element that is the next sibling (i.e., comes directly after) of the specified element.

h3+p

Selects the first <p> after any <h3>. **General sibling** A specified element that shares the same parent as the specified element.

h3~p

Selects all the <p> elements that share the same parent as the <h3>.

4 (a) Explain Location of styles with examples. [05] CO3 L2

CSS style rules can be located in three different locations.

**1. Inline Styles**
**Inline styles** are style rules placed within an HTML element via the style attribute,An inline style only affects the element it is defined within and overrides any other style definitions for properties used in the inline style.
Using inline styles is generally discouraged since they increase bandwidth and decrease maintainability (because presentation and content are intermixed and because it can be difficult to make consistent inline style changes across multiple files.

Internal styles example:

```
<h1>Share Your Travels</h1>
<h2>style="font-size: 24pt"Description</h2>
...
<h2>style="font-size: 24pt; font-weight: bold;">Reviews</h2>
```

**2. Embedded Style Sheet**
**Embedded style sheets** (also called **internal styles**) are style rules placed within the <style> element (inside the <head> element of an HTML document), While better than inline styles, using embedded styles is also by and large discouraged. Since each HTML document has its own <style> element, it is more difficult to consistently style multiple documents when using embedded styles.
Just as with inline styles, embedded styles can, however, be helpful when quickly testing out a style that is used in multiple places within a single HTML document.

Example:

```
<title>Share Your Travels -- New York - Central Park</title>
<style>
h1 { font-size: 24pt; }
h2 {
font-size: 18pt;
font-weight: bold;
}
</style>
```

```
</head>
<body>
<h1>Share Your Travels</h1>
<h2>New York - Central Park</h2>
...
```

**3. External Style Sheet**

**External style sheets** are style rules placed within a external text file with the **.css** extension. This is by far the most common place to locate style rules because it provides the best maintainability. When you make a change to an external style sheet, all HTML documents that reference that style sheet will automatically use the updated version. The browser is able to cache the external style sheet, which can improve the performance of the site as well.
To reference an external style sheet, you must use a <link> element (within the <head> element),We can link to several style sheets at a time; each linked style sheet will require its own <link> element.

Example:

```
<head lang="en">
<meta charset="utf-8">
<title>Share Your Travels -- New York - Central Park</title>
<link rel="stylesheet" href="styles.css" />
</head>
```

(b) Explain how CSS styles interact. [05] CO3 L2

**1. Inheritance**

**Inheritance** is the first of these cascading principles. Many (but not all) CSS properties affect not only themselves but their descendants as well. Font, color, list, and text properties are inheritable; layout, sizing, border, background, and spacing properties are not.

In the below example, without using inherit keyword, we have a single style rule that styles *all* the  elements. The <p> and <time> elements within the  inherit the bold font-weight property but not the margin or border styles.

Example:

```
div {
font-weight: bold;
margin: 50px;
border: 1pt solid green;
}
p {
border: inherit;
margin: inherit;
}
<h3>Reviews</h3>
<div>
<p>By Ricardo on <time>September 15, 2015</time></p>
<p>Easy on the HDR buddy.</p>
</div>
<hr/>
<div>
<p>By Susan on <time>October 1, 2015</time></p>
<p>I love Central Park.</p>
```

<hr/>
However, it is possible to tell elements to inherit properties that are normally not inheritable by using the <p> elements nested within the  elements now inherit the border and margins of their parent.

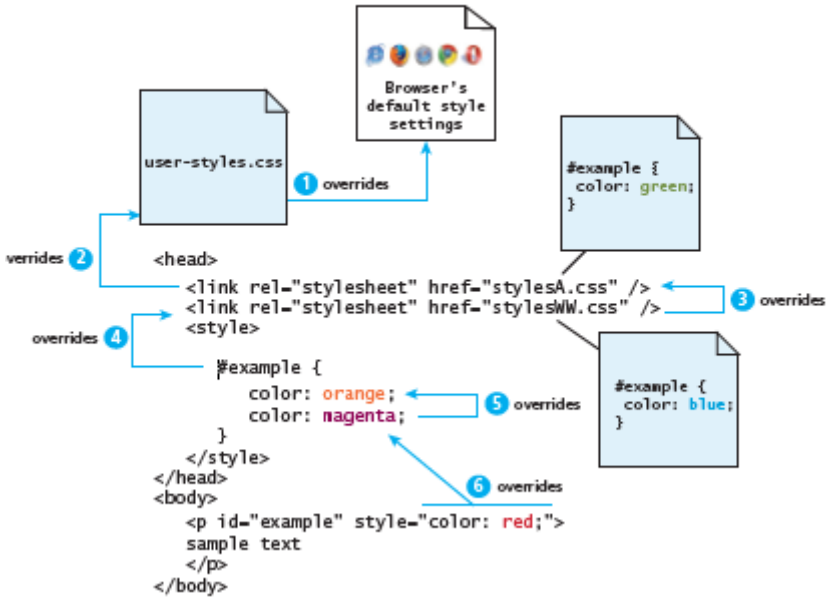
**2. Specificity**

**Specificity** is how the browser determines which style rule takes precedence when more than one style rule could be applied to the same element. In CSS, the more specific the selector, the more it takes precedence (i.e., overrides the previous definition).

**3. Location**

When inheritance and specificity cannot determine style precedence, the principle of **location** will be used. The principle of location is that when rules have the same specificity, then the latest are given more weight.

For instance, an inline style will override one defined in an external author style sheet or an embedded style sheet.




In the above example paragraph uses red color

5 (a) Explain the need of cascade in CSS. Explain the 3 principles of cascade with suitable CSS script segments. [05] CO3 L2

The "Cascade" in CSS refers to how conflicting rules are handled. The visual metaphor behind the term **cascade** is that of a mountain stream progressing downstream over rocks (and not that of a popular dishwashing detergent). The downward movement of water down a cascade is meant to be analogous to how a given style rule will continue to take precedence with child elements (i.e., elements "below" in a document outline as shown in Figure 3.3).

CSS uses the following cascade principles to help it deal with conflicts: inheritance, specificity, and location.

### 3.5.1 Inheritance

**Inheritance** is the first of these cascading principles. Many (but not all) CSS properties affect not only themselves but their descendants as well. Font, color, list, and text properties (from Table 3.1) are inheritable; layout, sizing, border, background, and spacing properties are not.

Figures 3.9 and 3.10 illustrate CSS inheritance. In the first example, only some of the property rules are inherited for the <body> element. That is, only the body element (thankfully!) will have a thick green border and the 100-px margin;

however, all the text in the other elements in the document will be in the Arial font and colored red.

In the second example in Figure 3.10, you can assume there is no longer the body styling but instead we have a single style rule that styles *all* the  elements. The <p> and <time> elements within the  inherit the bold font-weight property but not the margin or border styles. However, it is possible to tell elements to inherit properties that are normally not inheritable, as shown in Figure 3.11. In comparison to Figure 3.10, notice how the <p> elements nested within the  elements now inherit the border and margins of their parent.

### 3.5.2 Specificity

**Specificity** is how the browser determines which style rule takes precedence when more than one style rule could be applied to the same element. In CSS, the more specific the selector, the more it takes precedence (i.e., overrides the previous definition).

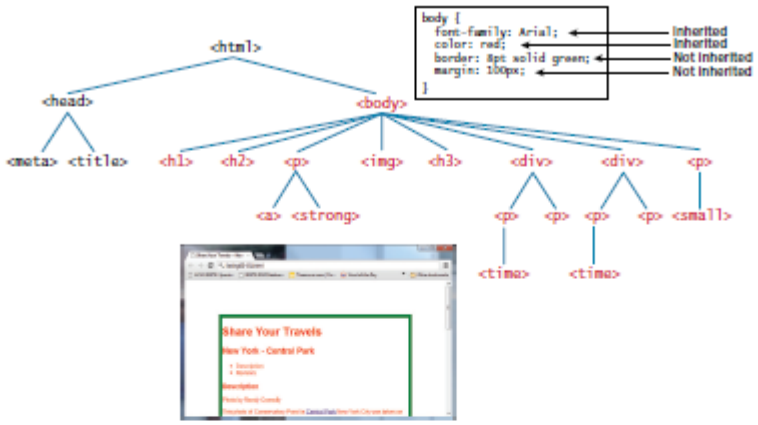



FIGURE 3.9 Inheritance

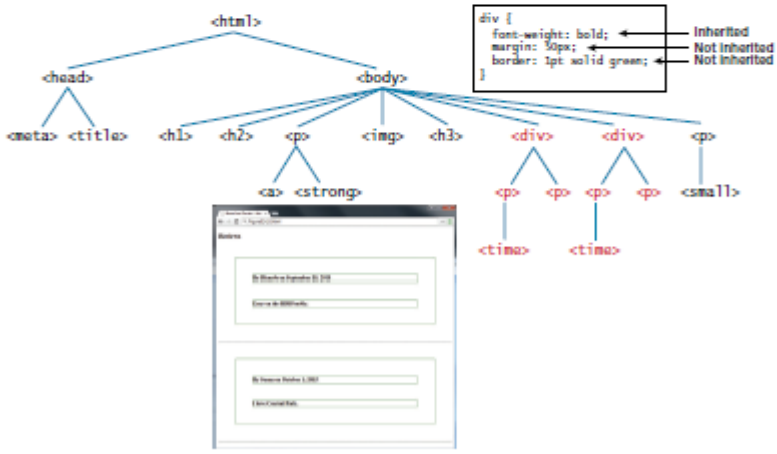



FIGURE 3.10 More Inheritance

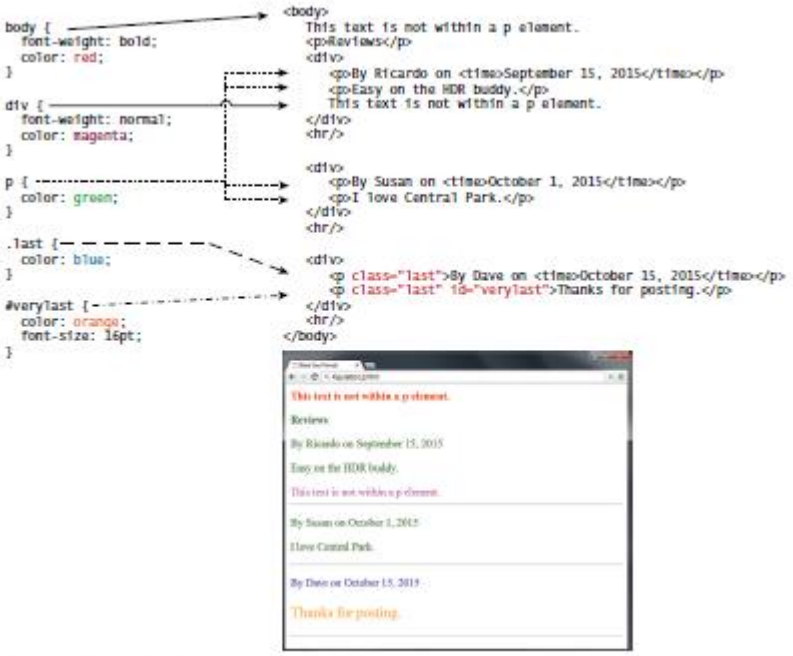



FIGURE 3.12 Specificity

As you can see in Figure 3.12, class selectors take precedence over element selectors, and id selectors take precedence over class selectors. The precise algorithm the browser is supposed to use to determine specificity is quite complex.6 A simplified version is shown in Figure 3.13.

### 3.5.3 Location

Finally, when inheritance and specificity cannot determine style precedence, the principle of **location** will be used. The principle of location is that when rules have the same specificity, then the latest are given more weight. For instance, an inline style will override one defined in an external author style sheet or an embedded style sheet.

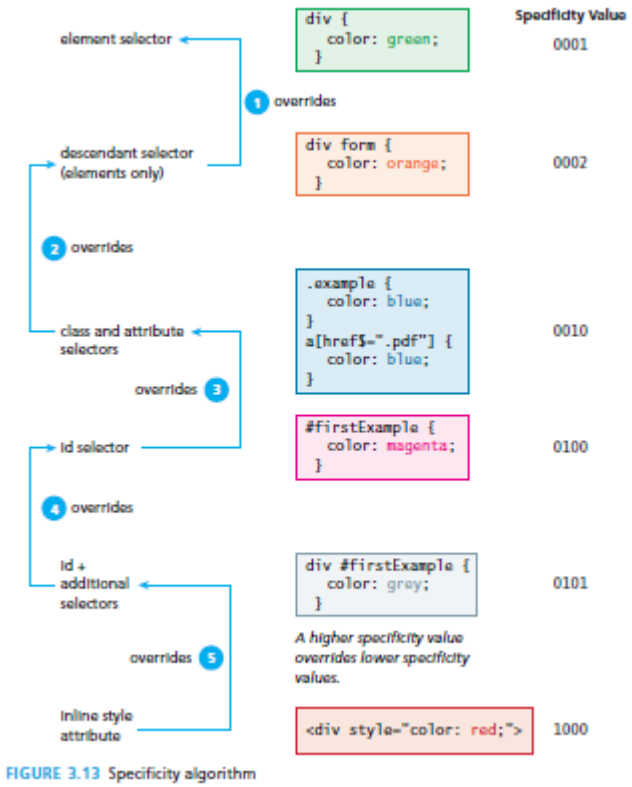



FIGURE 3.13 Specificity algorithm

5 (b) Explain the role of <ul> and <ol> HTML tags with syntax and examples. [05] CO1 L2

- **Unordered lists**. Collections of items in no particular order; these are by

default rendered by the browser as a bulleted list. However, it is common in CSS to style unordered lists without the bullets. Unordered lists have become the conventional way to markup navigational menus.

■ **Ordered lists**. Collections of items that have a set order; these are by default rendered by the browser as a numbered list.

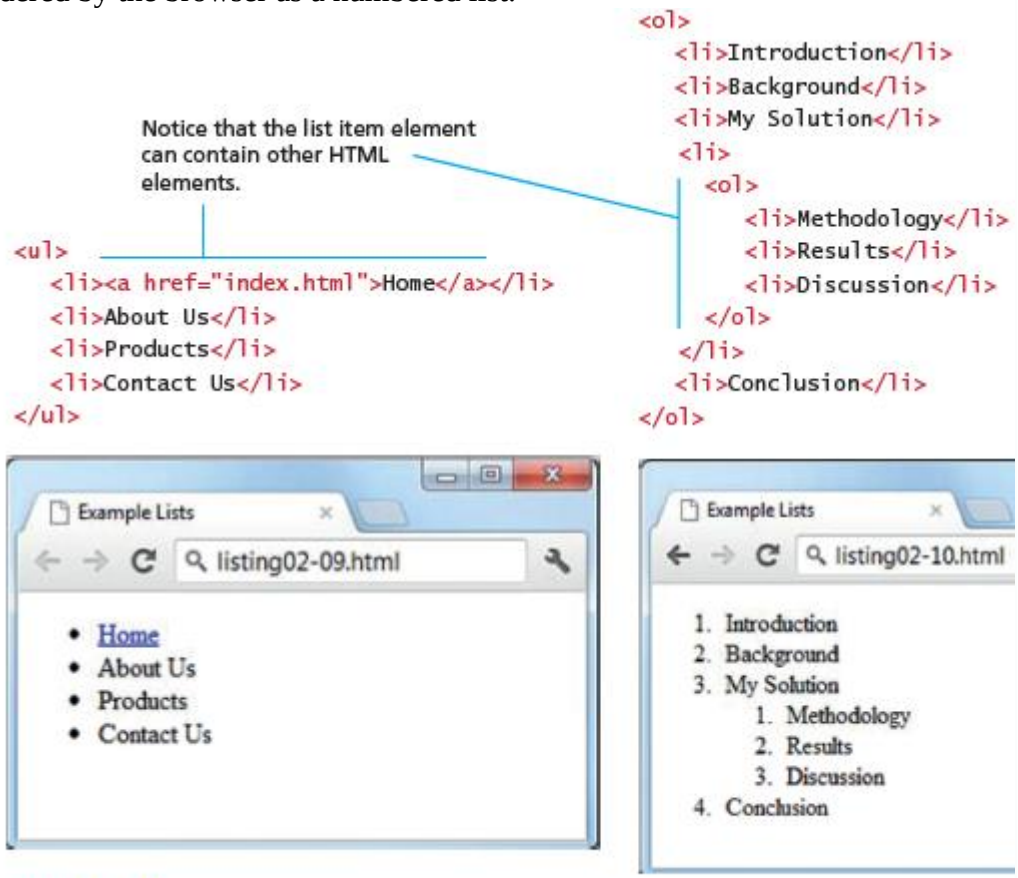



**FIGURE 2.19** List elements and their default rendering

6 (a) Explain different form widgets created with the <input> tag. [05] CO2 L2

A <form> element, which is a container for other elements that represent the various input elements within the form as well as plain text and almost any other HTML element.

Most forms need to gather text information from the user. Whether it is a search box, or a login form, or a user registration form, some type of text input is usually necessary. Table 4.3 lists the different text input controls.

While some of the HTML5 text elements are not uniformly supported by all browsers, they still work as regular text boxes in older browsers.

| Type | Description |
|---|---|
| **text** | Creates a single-line text entry box.<br>`<input type="text" name="title" />` |
| **textarea** | Creates a multiline text entry box. You can add content text or if using an HTML5 browser, placeholder text (hint text that disappears once user begins typing into the field).<br>`<textarea rows="3" ... />` |
| **password** | Creates a single-line text entry box for a password (which masks the user entry as bullets or some other character)<br>`<input type="password" ... />` |
| **search** | Creates a single-line text entry box suitable for a search string. This is an HTML5 element. Some browsers on some platforms will style search elements differently or will provide a clear field icon within the text box.<br>`<input type="search" ... />` |
| **email** | Creates a single-line text entry box suitable for entering an email address. This is an HTML5 element. Some devices (such as the iPhone) will provide a specialized keyboard for this element. Some browsers will perform validation when form is submitted.<br>`<input type="email" ... />` |
| **tel** | Creates a single-line text entry box suitable for entering a telephone. This is an HTML5 element. Since telephone numbers have different formats in different parts of the world, current browsers do not perform any special formatting or validation. Some devices may, however, provide a specialized keyboard for this element.<br>`<input type="tel" ... />` |
| **url** | Creates a single-line text entry box suitable for entering a URL. This is an HTML5 element. Some devices may provide a specialized keyboard for this element. Some browsers also perform validation on submission.<br>`<input type="url" ... />` |

TABLE 4.3 Text Input Controls

```
<input type="text" ... />
```

```
<textarea>
  enter some text
</textarea>
```

```
<textarea placeholder="enter some text">
</textarea>
```

```
<input type="password" ... />
```

```
<input type="search" placeholder="enter search text" ... />
```

```
<input type="email" ... />
```

*In Opera*

*In Chrome*

```
<input type="url" ... />
```

```
<input type="tel" ... />
```

FIGURE 4.16 Text input controls

(b) Write a HTML5 program for the following table. [05] CO3 L3

| DAY | SEMINAR | | |
|---|---|---|---|
| | SCHEDULE | | TOPIC |
| | BEGIN | END | |
| Monday | 8.00 am | 5.00 pm | Introduction to XML |
| | | | Validity: DTD & NG |
| TUESDAY | 11.00 am | 2.00 pm | XPAT4 |
| | 11.00 pm | 2.00 pm | |
| | 2.00 pm | 5.00 pm | XSL Transformations |
| WEDNESDAY | 8.00 am | 5.00 pm | XSL Formatting Objects |

```
<!DOCTYPE html>
<html>
<head>
	<title>Table Program</title>
</head>
<body>
	<table border="2">
		<tr>
			<th rowspan="3">DAY</th><th colspan="3">SEMINAR</th>
		</tr>
		<tr>
			<th colspan="2">SCHEDULE</th><th rowspan="2">TOPIC</th>
		</tr>
		<tr><th>BEGIN</th><th>END</th></tr>
		<tr>
			<td rowspan="2">Monday</td>
			<td rowspan="2">8.00 am</td>
			<td rowspan="2">5.00 pm</td>
			<td>Introduction to XML</td>
		</tr>
		<tr><td>Validity: DTD & NG</td></tr>
		<tr><td rowspan="3">TUESDAY</td><td>11.00 am</td><td>2.00 pm</td><td
rowspan="2">XPAT4</td></tr>
		<tr><td>11.00 pm</td><td>2.00 pm</td></tr>
		<tr><td>2.00 pm</td><td>5.00 pm</td><td>XSL Transformations</td></tr>
		<tr><td>WEDNESDAY</td><td>8.00 am</td><td>5.00 pm</td><td>XSL Formatting
Objects</td></tr>
	</table>
</body>
</html>
```