UNIT - 1

WEB FUNDAMENTALS AND HTML

A BRIEF INTRODUCTION TO THE INTERNET

1. Origins of the Internet:

- In the 1960s, the U.S. Department of Defense wanted a large computer network.
- ARPAnet, the first network, was created in 1969 for defense-related research.
- Other networks like BITNET and CSNET followed but weren't widely used.
- NSFnet, established in 1986, replaced ARPAnet and connected millions of computers.

2. What Is the Internet?

- The Internet is a vast network connecting computers of various sizes and types.
- TCP/IP (Transmission Control Protocol/Internet Protocol) is a crucial protocol for communication.
- It's a network of networks, not just a network of individual computers.

3. Internet Protocol Addresses:

- Computers on the Internet have unique numeric addresses (IP addresses).
- IP addresses are written as four 8-bit numbers, like 191.57.126.0.
- A new standard, IPv6, with larger addresses (128 bits), was introduced in 1998.

4. Domain Names:

- People use names (like www.ual.com) instead of numbers.
- Domain names have hierarchical structures, indicating the host's location and organization type.
- Fully Qualified Domain Names (e.g., movies.marxbros.comedy.com) help identify locations on the Internet.
- DNS (Domain Name System) and name servers convert human-readable names into IP addresses.

5. Protocols and the World Wide Web:

- Initially, different protocols (telnet, ftp, Usenet, mailto) had separate interfaces, limiting Internet growth.
- The World Wide Web emerged as a better approach, unifying access through a single interface.

In simpler terms, the Internet started as a military research project, grew into a network of networks using TCP/IP, assigned unique IP addresses to each device, introduced domain names for easier human understanding, and eventually embraced the World Wide Web for a more user-friendly experience.

THE WORLD WIDE WEB

1. Origins:

- In 1989, a group at CERN proposed a new system for the Internet, which they named the World Wide Web (Web).
- The goal was to let scientists globally share documents about their work.

2. System Design:

- The Web was designed to let anyone on the Internet search and retrieve documents from various servers.
- It used hypertext, allowing nonsequential browsing through text with embedded links.

3. Terminology:

- The terms "pages," "documents," and "resources" are used, but the most common are "pages" and "documents."
- "Hypermedia" refers to documents with non-textual information.

4. Web vs. Internet:

- The Internet is the network connecting computers and devices.
- The Web is software installed on computers on the Internet, allowing the sharing of documents.
- Web servers provide documents, and Web browsers request and display them.
- The Internet was useful before the Web, and it still is, but most people now use the Internet through the Web.

S.No.	INTERNET	WWW
1	Internet is a global network of networks.	WWW stands for World wide Web.
2	Internet is a means of connecting a computer to any other computer anywhere in the world.	World Wide Web which is a collection of information which is accessed via the Internet.
3	Internet is infrastructure.	WWW is service on top of that infrastructure.
4	Internet can be viewed as a big book-store.	Web can be viewed as collection of books on that store.
5	At some advanced level, to understand we can think of the Internet as hardware.	At some advanced level, to understand we can think of the WWW as software.
6	Internet is primarily hardware-based.	WWW is more software-oriented as compared to the Internet.
7	It is originated sometimes in late 1960s.	English scientist Tim Berners-Lee invented the World Wide Web in 1989.
8	Internet is superset of WWW.	WWW is a subset of the Internet.
9	The first version of the Internet was known as ARPANET.	In the beginning WWW was known as NSFNET.
10	Internet uses IP address.	WWW uses HTTP.

Parameters of Comparison	Internet	World Wide Web
Founded	It was established in the later 1960s.	It was established in 1989.
Meaning	It is a massive global network made up of millions of small subnetworks.	It is a system of information where data is kept for public access.
Nature	There is a whole infrastructure there.	It is a specific service contained within an infrastructure.
Type	It emphasizes hardware.	The emphasis is on software.
Dependency	It is independent of the world wide web.	Here, the internet is a need.
Use	It may be used for a variety of things, including banking, entertainment, research, education, and navigation.	It is employed to gain access to resources all across the world.

In simpler terms, the World Wide Web started as a way for scientists to share documents globally. It uses hypertext for easy browsing and includes various types of media. Remember, the Web is a part of the Internet, and while the Internet is still useful without the Web, most people use the Internet through the Web nowadays.

WEB BROWSERS

1. Client-Server Communication:

- When computers communicate on the Web, one acts as a client (initiator) and the other as a server (respondent).
- Browsers are programs on client machines that request and display documents from servers.

2. Evolution of Browsers:

- Early browsers were text-based, limiting Web growth.
- In 1993, Mosaic, the first graphical browser, changed the game.
- Mosaic's user-friendly interface fueled explosive growth in Web usage.

3. Browsing Basics:

- Browsers initiate communication by requesting documents from servers.
- Servers send documents back to be displayed by the browser.
- The Web supports various protocols, with HTTP being the most common.

4. Types of Browsers:

- Common browsers include Microsoft Internet Explorer (IE), Firefox, and Chrome.
- IE is for PCs using Microsoft Windows, while Firefox and Chrome are cross-platform.
- Other browsers like Opera and Safari exist, but we'll focus on Chrome, IE, and Firefox.

In simpler terms, web browsers are like messengers that users employ to request and view information from servers on the Internet. They've come a long way from basic text to user-friendly graphical interfaces. The most commonly used browsers are Chrome, IE, and Firefox, each serving as a tool to explore and interact with the vast world of the Internet.

WEB SERVERS

1. Role of Web Servers:

Web servers are crucial programs responsible for delivering documents to browsers upon request. They act as responsive entities, only engaging when browsers on other computers make requests.

2. Dominant Web Servers:

The primary players in the web server domain are Apache (for various platforms) and Microsoft's Internet Information Server (IIS, specifically for Windows). As of October 2013, Apache held about 65%, IIS around 16%, and nginx (engine-x) from Russia at 14% among active web hosts.

3. Web Server Operation:

Web servers play a vital role in optimizing the distribution of information. While serving information is efficient, displaying it on client screens consumes time. The client-server architecture enables a small number of servers to efficiently provide documents to a large number of clients.

4. Communication Process:

Communication between web clients and servers relies on the HTTP protocol. A web server, upon execution, notifies the operating system of its readiness to accept incoming connections through a specific port. Web clients (browsers) establish connections, send requests and data, receive information, and then close the connection.

5. Server Characteristics:

Web servers share common characteristics, regardless of origin or platform. The file structure typically includes two main directories - the document root (housing accessible web documents) and the server root (storing the server and support software).

6. Document Retrieval Process:

Clients access documents through top-level URLs, with servers mapping requested URLs to the document root. Virtual document trees allow servers to store documents outside the document root, accommodating growing collections.

7. Additional Server Services:

Early servers primarily returned files or program outputs. Modern servers are complex systems offering various client services. They can support multiple sites on one computer (virtual hosts) and may serve as proxy servers, serving documents from other machines.

8**. Protocol and Database Interaction:**

Although originally designed for HTTP, many web servers now support additional protocols like ftp, gopher, news, and mailto. Furthermore, nearly all web servers can interact with database systems through server-side scripts.

9. Apache:

Apache, originating from the NCSA server httpd, is popular for its speed, reliability, and open-source nature. Managed by a large team of volunteers, Apache is especially favored for Linux systems, reading configuration information from files.

10. IIS:

Microsoft's IIS, bundled with Windows, dominates on Windows platforms. Unlike Apache's configuration file control, IIS is managed through a window-based program, the IIS snap-in, enabling site managers to modify server parameters.

In essence, web servers play a pivotal role in responding to client requests, optimizing information distribution, and supporting a variety of services. Apache and IIS, with their distinctive features, cater to different preferences and system environments.

URLs, MIME, HTTP, Security

Uniform Resource Locators

Definition of URIs:

- URIs are used to identify resources on the Internet, known as Uniform Resource Identifiers (URIs).
- URIs can be Uniform Resource Names (URNs) or Uniform Resource Locators (URLs).

URL Formats:

- General format: scheme:object-address.
- Scheme represents the communication protocol, like http, ftp, file, etc.
- HTTP protocol structure: //fully-qualified-domain-name/path-to-document.
- File protocol structure: file://path-to-document.

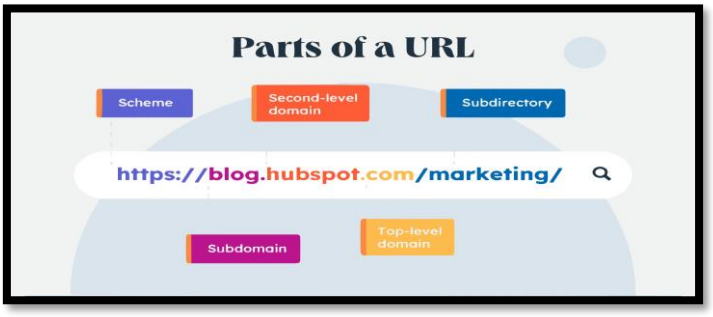
URL Components:

- Host Name: Server computer storing/accessing the document.
- Port Numbers: Specific processes on the host machine. Default HTTP port: 80.
- Encoding and Special Characters: Spaces and disallowed characters are replaced with percent-encoded ASCII codes in URLs.

URL Paths

Path Structure:

- HTTP protocol paths resemble file system paths.
- Specified by directory names and file names separated by slashes.
- Relative and complete paths, and their usage in URLs.



Multipurpose Internet Mail Extensions (MIME)

Type Specifications:

- MIME specifies document formats transmitted over the Web.
- Categories: type/subtype (e.g., text/plain, image/jpeg).
- Servers determine document type by file name extension.

Experimental Document Types:

- "x-" prefixed experimental subtypes used for custom data types.
- Introduced via MIME specifications in the Web provider's server.

The Hypertext Transfer Protocol (HTTP)

Request Phase:

- HTTP request structure: method, domain part of URL, HTTP version.
- Common methods: GET, HEAD, POST, PUT, DELETE.
- Request header fields and their purposes.

Response Phase:

- HTTP response structure: status line, response header fields, response body.
- Status codes and their categories (e.g., 200 OK, 404 Not Found).

Security

Encryption and Public-Key Encryption:

- Encryption for privacy and integrity; public-key encryption for secure transmission.
- RSA algorithm and its use in e-commerce.

Security Challenges:

- DoS attacks, viruses, worms causing damage by overwhelming servers or infecting systems.
- Protection with antivirus software and continuous updates to counter new threats.

INTRODUCTION TO HTML

Origins and Evolution of HTML and XHTML

Versions of HTML and XHTML

1. **HTML as a Markup Language:** HTML is a markup language designed to mark parts of documents for display. It uses the meta-markup language SGML to specify document structure rather than detailed presentation.
2. **Competition and Incompatibility:** In the mid-1990s, a competition between Netscape and Microsoft led to incompatible versions of HTML. This posed challenges for web content providers to design documents viewable on different browsers.
3. **Formation of W3C:** Tim Berners-Lee initiated the World Wide Web Consortium (W3C) in 1994 to develop web standards. The first HTML standard, HTML 2.0, was released in 1995. HTML 4.01 became the latest standard in 1999, developed primarily by W3C.
4. **Introduction of Style Sheets:** Style sheets in the late 1990s enhanced HTML's capabilities by allowing the specification of presentation details. This led to HTML being closer to other information-formatting languages.
5. **XHTML as an Alternative:** XHTML, based on XML, was introduced to address issues in HTML 4.01. It provided stricter syntax rules and required documents to be well-formed. XHTML 1.0 was approved in 2000.

HTML versus XHTML

1. **Evolution to XHTML:** XHTML 1.1, recommended in 2001, dropped some features of its predecessor and insisted on strict syntax rules. However, due to concerns about error messages, XHTML 1.1 documents continued to be served with the text/html MIME type.
2. **XHTML 2.0 Challenges:** XHTML 2.0, not backward compatible with HTML 4.01 or XHTML 1.1, faced criticism. In response, the WHAT Working Group started working on a new version of HTML, which eventually became HTML5.
3. **HTML5 Emergence:** In 2006, W3C joined forces with the WHAT Working Group, leading to the abandonment of XHTML 2.0 in favor of HTML5. HTML5 aimed for backward compatibility, clearly defined error handling, and user-friendly syntax.

Choosing Between HTML and XHTML

1. **Developer Choices:** Until 2010, some developers used XHTML for its stricter rules, while others stuck to HTML. HTML5's adoption by major browsers made W3C XHTML 1.0 Strict validation impractical.
2. **Standards and Consistency:** XHTML offers strict rules for consistent document structure, contrasting with the flexibility of HTML. The compromise approach in the book is to present HTML5 while promoting XHTML 1.0 Strict syntax.

Conclusion

The historical context of HTML and XHTML, detailing their evolution, challenges, and the emergence of HTML5. It also discusses the ongoing choice between HTML and XHTML, emphasizing the importance of standards and syntax rules.

S.No.	HTML	XHTML
1.	HTML stands for Hypertext Markup Language.	XHTML stands for Extensible Hypertext Markup Language.
2.	It was developed by Tim Berners-Lee.	It was developed by W3C i.e World Wide Web Consortium.
3.	It was developed in 1991.	It was released in 2000.
4.	It is extended from SGML.	It is extended from XML and HTML.
5.	The format is a document file format.	The format is a markup language.
6.	All tags and attributes are not necessarily to be in lower or upper case.	In this, every tag and attribute should be in lower case.
7.	Doctype is not necessary to write at the top.	Doctype is very necessary to write at the top of the file.
8.	It is not necessary to close the tags in the order they are opened.	It is necessary to close the tags in the order they are opened.
9.	While using the attributes it is not necessary to mention quotes. For e.g. <Geeks>.	While using the attributes it is mandatory to mention quotes. For e.g. <Geeks="GFG">.
10.	Filename extension used are .html, .htm.	Filename extension are .xhtml, .xht, .xml.

BASIC SYNTAX

Tags and Elements

- **Tags:** Tags are fundamental syntactic units in HTML used to specify content categories. They are enclosed in angle brackets (< and >) and written in lowercase letters. Most tags appear in pairs, with an opening tag and a closing tag.
- **Elements:** The combination of an opening tag, its corresponding closing tag, and the content between them is called an element. Elements form containers for the specified content.
- **Example:**
    - ** This is simple stuff. **

Attributes

- **Attributes:** Attributes are used to specify alternative meanings of a tag. They are written between the opening tag name and the right-angle bracket. Attributes are in keyword form, with the attribute's name followed by an equals sign and the attribute's value. Attribute values must be enclosed in double quotes.
- Example:
    - Visit Example.com

Comments

- **HTML Comments:** Comments in HTML are written between <!-- and -->. They are for human readability and are ignored by browsers. Comments can span multiple lines.
- Example:
        - <!-- This is a comment in HTML -->
    - **Informational Comments:** Comments play a crucial role in making HTML documents more understandable. They can provide information about the purpose of specific sequences of markup.

Ignored Text

- **Unrecognized Tags:** Browsers ignore unrecognized tags in HTML.
- **Line Breaks:** Browsers ignore line breaks unless specified with appropriate tags.
- **Multiple Spaces and Tabs:** Similar to line breaks, browsers ignore extra spaces and tabs unless specified using appropriate tags.

HTML as Suggestions

- **HTML vs. Programs:** HTML tags are treated more like suggestions to the browser rather than explicit instructions. Unlike programs, misspelled tags are often ignored by browsers without indicating an error to the user.
- **Browser Configuration:** Browsers have the flexibility to ignore recognized tags or react differently to specific tags based on user configurations.

Conclusion

This section introduces the foundational concepts of HTML syntax, emphasizing tags, elements, attributes, and comments. It also highlights the browser's flexibility in handling HTML content and the role of comments in improving document readability.

Standard HTML Document Structure

DOCTYPE Declaration

The first line of an HTML document is the DOCTYPE declaration, specifying the SGML Document-Type Definition (DTD) compliance. For HTML5, it's simply:

```
<!DOCTYPE html>
```

Root Element - `<html>`

- The `**<html>**` tag identifies the root element of the document.
- It includes an attribute `lang` specifying the language used in the document.

```
<html lang="en">
```

Document Sections - `<head>` and `<body>`

- `**<head>**` **Element:**Contains information about the document but not its content.
- Includes a `**<title>**` element displaying content at the top of the browser window's title bar.
- Utilizes the `**<meta>**` element to specify document information using attributes (e.g., character set).

```
<head>
  <title> A title for the document </title>
  <meta charset="utf-8" />
  <!-- ... Other meta information -->
</head>
```

- `<body>` Element: Provides the document's content.

```
<body>
  <!-- Content of the document -->
</body>
```

Basic Structure Example

A skeletal example demonstrating the basic structure of an HTML document:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title> A title for the document </title>
  <meta charset="utf-8" />
  <!-- ... Other meta information -->
</head>
<body>
  <!-- Content of the document -->
</body>
</html>
```

Formatting and Readability

Consistent formatting and indentation are recommended to enhance the readability of HTML documents. Indentation of nested elements improves the overall structure's clarity, similar to formatting conventions in programming.

Conclusion

This section presents the fundamental structure of an HTML document, emphasizing the required elements (`<html>`, `<head>`, `<title>`, `<meta>`, and `<body>`) and their respective roles in organizing and displaying content. It also highlights the importance of formatting for readability purposes.

Basic Text Markup

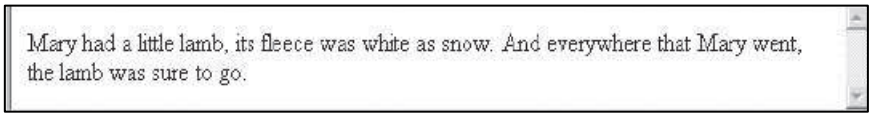
This section discusses how text content in the body of an HTML document can be formatted using HTML tags. The following elements are covered:

Paragraphs

Text is typically organized into paragraphs using the <p> (paragraph) element. The browser automatically provides line breaks, and multiple spaces are condensed to a single space.

Example:

```
<p>
Mary had
a
little lamb, its fleece was white as snow. And
everywhere that
Mary went, the lamb
was sure to go.
</p>
```



Line Breaks

The
 (break) tag is used to create explicit line breaks within text. It does not have a closing tag, and in XHTML, it can be written as
.

Example:

```
<p>
Mary had a little lamb, <br />
its fleece was white as snow.
</p>
```

Preserving White Space

The <pre> (preformatted text) element preserves white space, including multiple spaces and line breaks. The content of <pre> is displayed in a monospace font.

Example:

```
<pre>
Mary
 had  a
        little
                lamb
</pre>
```

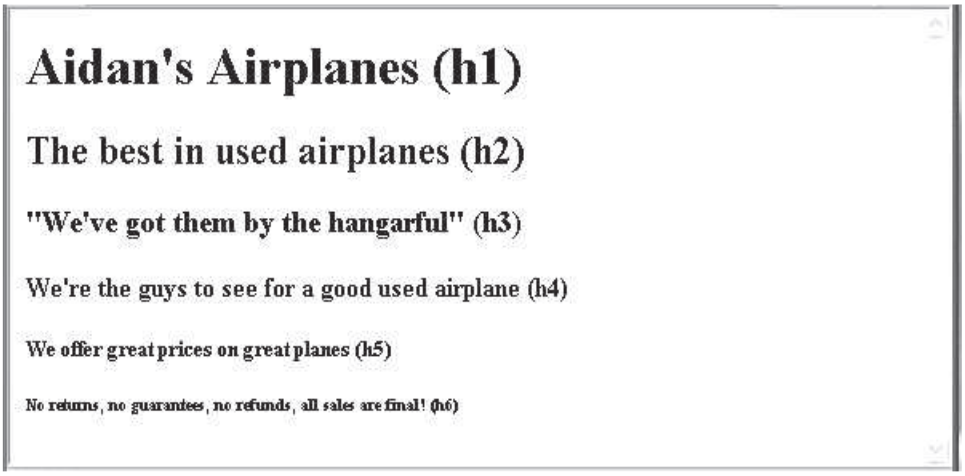
```
Mary
    had a
        little
            lamb
```

Headings

Headings are specified using <h1> to <h6>, where <h1> is the highest-level heading. Headings break the current line and usually appear in bold, with varying font sizes.

Example:

```
<h1> Aidan's Airplanes (h1) </h1>
<h2> The best in used airplanes (h2) </h2>
<!-- ... Other headings ... -->
```

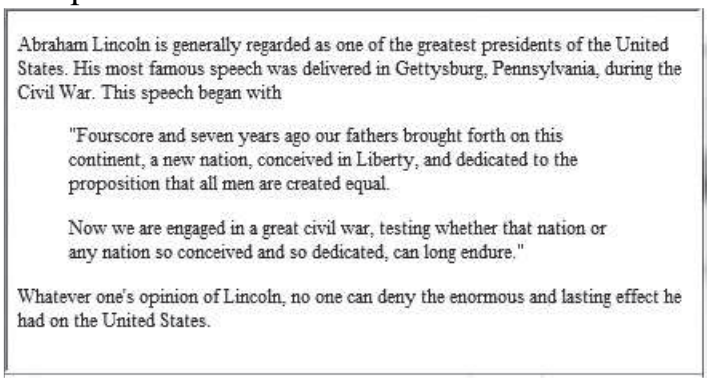


Block Quotations

Long quotations are set off using the <blockquote> element, which is often indented on both sides.

Example:

```
<blockquote>
  <p>
    "Fourscore and seven years ago our fathers brought forth on
    this continent, a new nation, conceived in Liberty, and
    dedicated to the proposition that all men are created equal."
  </p>
  <!-- ... Other paragraphs ... -->
</blockquote>
```



Font Styles and Sizes

Content-based style tags like (emphasis), (strong emphasis), and <code> (code) are used to indicate text styles. <sub> and <sup> create subscript and superscript characters, respectively.

```
cost = quantity * price
```

$x_2^3 + y_1^2$

Character Entities

HTML provides special character entities (e.g., < for < and € for €) to represent characters that cannot be typed directly.

Horizontal Rules

The <hr /> (horizontal rule) element is used to insert a horizontal line between document sections.

Other Uses of the meta-Element

The <meta> element is used to provide information about the document, such as keywords for search engines.

Example:

```
<meta name="keywords" content="binary trees, linked lists, stacks" />
```

In summary, this section covers various text formatting elements and techniques in HTML.

IMAGES

The inclusion of images in an HTML document can significantly improve its visual appeal, though it may impact the document's download speed. The two most common image formats are GIF and JPEG, with GIF supporting transparency and JPEG offering superior compression. PNG is a newer format with advantages of both GIF and JPEG, but it may result in larger file sizes.

Image Formats

- **GIF (Graphic Interchange Format)**: Supports 8-bit color, allowing 256 different colors. Files have a .gif extension.
- **JPEG (Joint Photographic Experts Group):** Supports 24-bit color, enabling more than 16 million colors. Files have .jpg, .jpeg, or .JPG extensions.
- **PNG (Portable Network Graphics):** Developed as a free replacement for GIF and JPEG, offering features of both formats. Files have a .png extension.

The Image Element

- The <img> element is an inline element used to display images. It includes the following attributes:
  - **src:** Specifies the file containing the image.
  - **alt:** Specifies text to be displayed when the image cannot be shown.

Optional attributes:

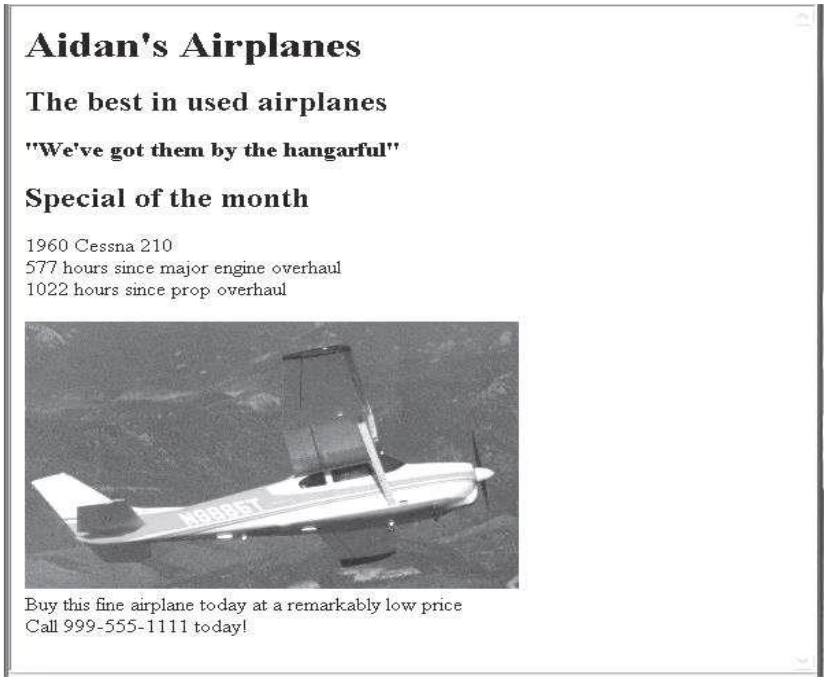
- width and height: Specify the size of the image in pixels.
- width="%": Specifies the width as a percentage of the display width.

Example:

```
<img src="images/stars.jpg" alt="Image description" />
<img src="c210.jpg" height="200" width="200" alt="Picture of a Cessna 210" />
```

Example Document with Image

```
<!DOCTYPE html>
<!-- image.html: An example to illustrate an image -->
<html lang="en">
<head>
  <title>Images</title>
  <meta charset="utf-8" />
</head>
<body>
  <h1>Aidan's Airplanes</h1>
  <h2>The best in used airplanes</h2>
  <h3>"We've got them by the hangarful"</h3>
  <h2>Special of the month</h2>
  <p>
    1960 Cessna 210 <br />
    577 hours since major engine overhaul<br />
    1022 hours since prop overhaul <br /><br />
    <img src="c210new.jpg" alt="Picture of a Cessna 210" /> <br />
    Buy this fine airplane today at a remarkably low price <br />
    Call 999-555-1111 today!
  </p>
</body>
</html>
```



In this example, an image of a Cessna 210 is included in an advertisement for an airplane sale. The image is displayed using the <img> element along with relevant details and a call to action.

HYPERTEXT LINKS

Hypertext links, specified using the <a> (anchor) element, enable navigation within and across HTML documents. The <a> element is an inline element and is used to create clickable links. The essential attribute for creating links is href (hypertext reference), specifying the target of the link.

Linking to Documents

The href attribute points to the target document. For documents in the same directory, only the document's name is needed. For documents in other directories, Unix path conventions are used.

Example:

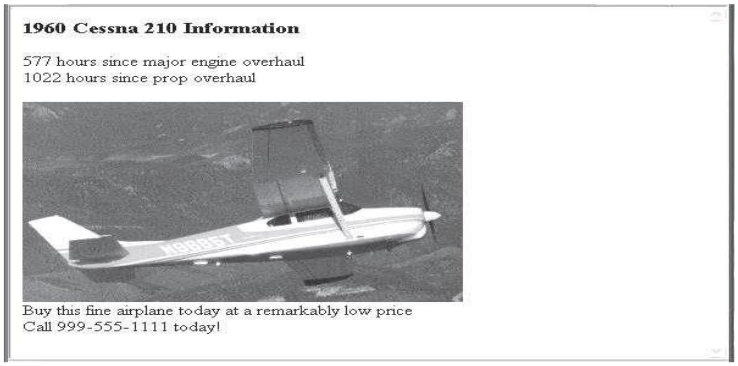
```
<a href="C210data.html">Information on the Cessna 210</a>
```

Linking with Images

Links can include images in their content. The image is displayed along with the textual link.

Example:

```
<a href="c210data.html">
  <img src="small-airplane.jpg" alt="An image of a small airplane" />
Information on the Cessna 210</a>
```



An image itself can serve as a link, with the content of the anchor element being just the image element.

Targets within Documents

To link to a specific element within a document, use the id attribute. The target element can be specified using the href attribute with a pound sign (#) followed by the id.

Example:

```
<a href="#avionics">What about avionics?</a>
```

If the target is in another document, append the id value to the end of the URL with a pound sign (#).

Example:

```
<a href="aidan1.html#avionics">Avionics</a>
```

Using Links

- Links are commonly used to create a table of contents within a document, allowing users to navigate quickly to various parts.
- Design links to blend into the text, avoiding distractions. A well-designed link should provide additional information without disrupting the reading flow.

Lists in HTML

HTML facilitates the creation and presentation of lists, offering three primary types: unordered lists, ordered lists, and definition lists.

Unordered Lists

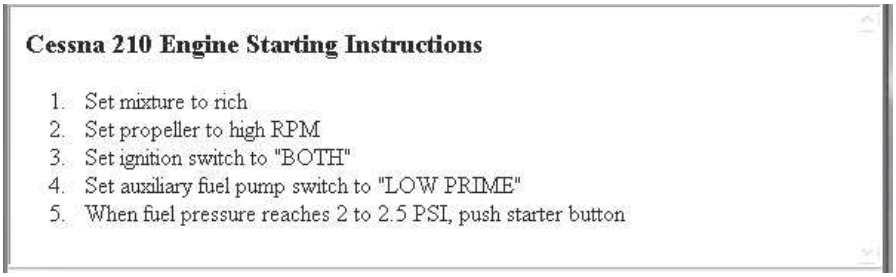
Unordered lists are crafted with the <ul> (unordered list) element, and each list item is defined using <li> (list item). This type of list is suitable for items where the order is not essential, such as a list of aircraft models:

```
<ul>
  <li>Cessna Skyhawk</li>
  <li>Beechcraft Bonanza</li>
  <li>Piper Cherokee</li>
</ul>
```

Ordered Lists

When the order of items is crucial, ordered lists come into play. Employing the <ol> (ordered list) element and <li> for list items, an ordered list provides a sequential structure. For instance, instructions for starting a Cessna 210 engine could be presented as:

```
<ol>
  <li>Set mixture to rich</li>
  <li>Set propeller to high RPM</li>
  <li>Set ignition switch to "BOTH"</li>
  <!-- ... other steps ... -->
</ol>
```

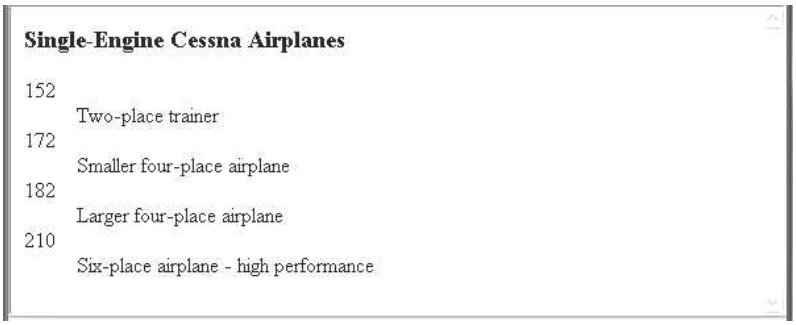


Nested ordered lists are achievable by placing a nested list within an <li> element.

Definition Lists

Definition lists, constructed with the <dl> (definition list) element, are ideal for presenting terms and their corresponding definitions. The terms are defined using <dt> (definition term), and the definitions with <dd> (definition description). An example related to single-engine Cessna airplanes could be:

```
<dl>
  <dt>152</dt>
  <dd>Two-place trainer</dd>
  <dt>172</dt>
  <dd>Smaller four-place airplane</dd>
  <!-- ... other terms and definitions ... -->
</dl>
```



In summary, HTML's list elements provide flexibility in structuring content, ensuring readability, and conveying relationships between different pieces of information. Whether representing a series of items, sequential steps, or term definitions, HTML lists are fundamental tools for effective content presentation.

TABLES IN HTML

Tables play a crucial role in presenting information effectively on the web. HTML provides features for creating tables, which are matrices of cells containing diverse content. This discussion covers the basic tags, attributes, and best practices associated with HTML tables.

Basic Table Tags:

In HTML, a table is created using the <table> element. The <caption> element, following the <table> tag, provides a title for the table. Rows are defined with <tr> (table row), and within each row, column labels are set using <th> (table heading), while data cells are specified with <td> (table data). The first row typically contains column labels.

```
<table>
 <caption> Fruit Juice Drinks </caption>
 <tr>
  <th> </th>
  <th> Apple </th>
  <th> Orange </th>
  <th> Screwdriver </th>
 </tr>
 <!-- ... data rows ... -->
</table>
```

Fruit Juice Drinks

	Apple	Orange	Screwdriver
Breakfast	0	1	0
Lunch	1	0	0
Dinner	0	0	1

The rowspan and colspan Attributes:

To handle complex table structures where labels span multiple rows or columns, the rowspan and colspan attributes come into play. They specify how many rows or columns a cell should span.

```
<tr>
 <th colspan="3"> Fruit Juice Drinks
</th>
</tr>
```

Fruit Juice Drinks		
Apple	**Orange**	**Screwdriver**

Table Sections:

Tables naturally consist of three parts: header (<thead>), body (<tbody>), and footer (<tfoot>). These parts aid in structuring and styling the table. The <thead> section typically contains column labels, <tbody> holds the main data, and <tfoot> may include additional information like column totals.

```
<table>
 <caption> Table with Sections </caption>
 <thead>
  <!-- ... column labels ... -->
 </thead>
 <tbody>
  <!-- ... data rows ... -->
 </tbody>
 <tfoot>
  <!-- ... additional information ... -->
 </tfoot>
</table>
```

Fruit Juice Drinks and Meals

	Fruit Juice Drinks		
	Apple	**Orange**	**Screwdriver**
Breakfast	0	1	0
Lunch	1	0	0
Dinner	0	0	1

Uses of Tables:

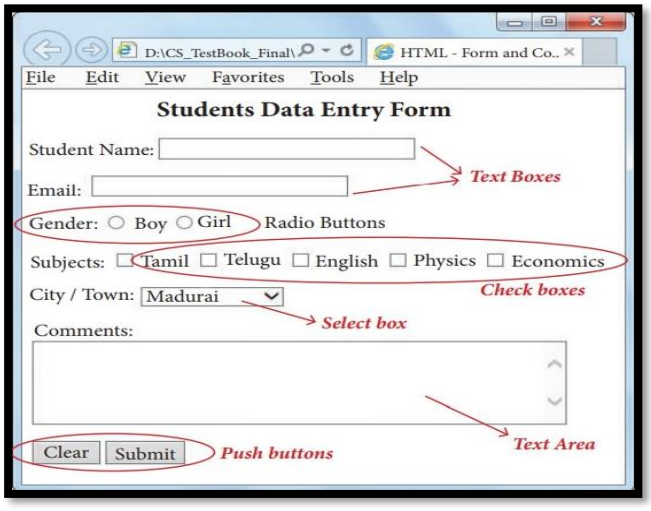
While tables are essential for presenting structured data, their misuse for whole document layout has been discouraged. During the late 1990s, tables were extensively used for layout due to limited CSS support and design tools. However, the industry trend shifted towards tableless layouts with the rise of CSS and increased awareness of its advantages, leading to smaller, more efficient documents.

Conclusion:

HTML tables are powerful tools for organizing and presenting data. Understanding their tags, attributes, and best practices ensures effective use without resorting to deprecated practices such as tables for layout, which can lead to unnecessary complexity and performance issues.

FORM IN HTML

- ✓ Form is used to collect user input. The user input is most often sent to a server for processing.
- ✓ HTML provides tags to generate the commonly used objects on a screen form. These objects are called controls or widgets or components.
    - Form Element
    - The input Element
    - The Select Element
    - The text area Element



Form Element (<form>):

- Acts as a container for form controls.
- Attributes include action (URL where form data is sent) and method (GET or POST).

Input Element (<input>):

- Used for various form controls like text, password, checkboxes, radio buttons, etc.
- Requires a type attribute to specify the control type.
- Additional attributes such as name, placeholder, and required can be used.

Select Element (<select>):

- Creates a dropdown list for users to choose from.
- Contains nested <option> tags that define individual options within the list.

Textarea Element (<textarea>):

- Allows users to input multiline text.
- Attributes include rows and cols to set the visible size of the textarea.

Form Submission:

- Form data is processed on the server and requires a "Submit" button.
- The action attribute in the <form> tag specifies the server URL.
- The method attribute determines how data is transmitted (GET or POST).

GET vs. POST:

- GET appends form data to the URL, visible in the address bar.
- POST sends form data in the request body, more secure for sensitive information.
- action and method attributes control form data transmission.

Input Types:

- Various input types include text, password, checkboxes, radio buttons, buttons, URLs, email addresses, and range (numeric input).

Validation:

- Textbox contents often need validation to ensure data integrity.
- Client-side validation is done to avoid sending invalid data to the server.
- Server-side validation is essential for security.

Checkbox and Radio Controls:

- Checkboxes collect on/off selections; radio buttons allow a single choice from a group.
- Each control needs a name attribute, and checkboxes have a value attribute.

Menus (<select>):

- Dropdown lists for multiple-choice options.
- Size and multiple attributes control the display and selection behavior.

Textarea (<textarea>):

- Allows multiline text input, useful for longer pieces of information.
- Attributes like rows and cols determine the visible size.

Action Buttons (<input type="submit/reset">):

- Submit button sends form data to the server.
- Reset button clears form controls to their initial states.

Image Button:

- Alternative to the Submit button, where an image is the clickable area.

Example of a Complete Form:

- Demonstrates a sales order form for popcorn with textboxes, a table for orders, radio buttons for payment methods, and action buttons.

Accessibility:

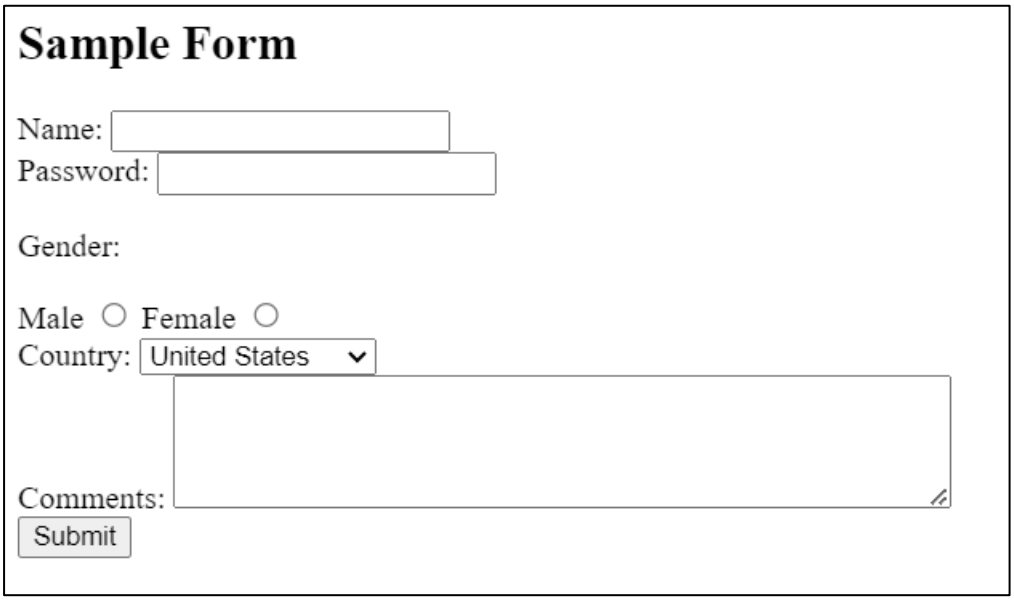
- Labels improve accessibility; use <label> to associate controls with descriptive text.
- Properly labeled controls assist users and support assistive technologies.

Example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Sample Form</title>
</head>
<body>
  <h2>Sample Form</h2>
```

```
  <form action="#" method="POST">
    <!-- Text Input -->
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required>
    <br>

    <!-- Password Input -->
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required>
    <br>

    <!-- Radio Buttons -->
    <p>Gender:</p>
    <label for="male">Male</label>
    <input type="radio" id="male" name="gender" value="male">
    <label for="female">Female</label>
    <input type="radio" id="female" name="gender" value="female">
    <br>

    <!-- Dropdown (Select) -->
    <label for="country">Country:</label>
    <select id="country" name="country">
      <option value="usa">United States</option>
      <option value="canada">Canada</option>
      <option value="uk">United Kingdom</option>
    </select>
    <br>

    <!-- Textarea -->
    <label for="comments">Comments:</label>
    <textarea id="comments" name="comments" rows="4"
cols="50"></textarea>
    <br>

    <!-- Submit Button -->
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

Output:



Overall, this comprehensive guide provides a solid understanding of HTML forms and their elements.

THE AUDIO ELEMENT:

The <audio> element in HTML5 is used to embed audio content, allowing you to include sound files directly in your web pages without the need for external plugins like Flash or media players. It provides native support for audio playback within modern web browsers.

Here's a breakdown of the key components and attributes associated with the <audio> element:

Syntax:

```
<audio controls>
  <source src="audio_file.mp3" type="audio/mp3">
  Your browser does not support the audio element.
</audio>
```

Attributes:

controls: This attribute adds audio controls like play, pause, and volume to the <audio> element. It is commonly used, as in the example, as controls="controls" or just controls.

Source Elements:

Inside the <audio> element, you use one or more <source> elements to specify the source of the audio files. Each <source> element should include the src attribute, pointing to the URL of the audio file, and the type attribute, indicating the MIME type of the file.

Fallback Content:

The text "Your browser does not support the audio element." is included between the opening and closing <audio> tags. This text is displayed if the browser does not support the <audio> element.

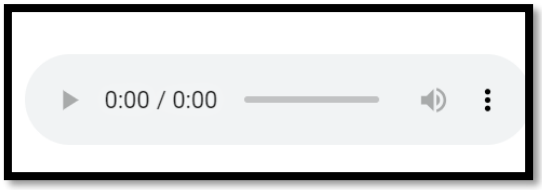
Browser Compatibility:

Different browsers support different audio container/codec combinations. Multiple <source> elements allow you to provide alternative audio formats, and the browser will choose the first one it supports.

Example:

```
<audio controls>
  <source src="audio_file.ogg" type="audio/ogg">
  <source src="audio_file.mp3" type="audio/mp3">
  Your browser does not support the audio element.
</audio>
```

In this example, the browser will attempt to play the Ogg file first and fall back to the MP3 file if Ogg is not supported.



Notes:

- Always include alternative audio formats to ensure compatibility across different browsers.
- Be mindful of the file formats and codecs supported by various browsers (as mentioned in your provided text).
- The <audio> element provides a simple and standardized way to integrate audio into web pages, enhancing the overall user experience.

THE VIDEO ELEMENT

The <video> element in HTML5 is used to embed video content directly into web pages, providing native support for video playback without the need for external plugins. It allows you to include video files in various formats, enhancing the multimedia capabilities of web documents. Here's an explanation of the key aspects associated with the <video> element:

Syntax:

```
<video width="600" height="500" autoplay controls preload>
  <source src="video_file.mp4" type="video/mp4">
  <source src="video_file.ogv" type="video/ogg">
  <source src="video_file.webm" type="video/webm">
  Your browser does not support the video element.
</video>
```

Attributes:

- **width and height:** These attributes set the dimensions of the video player on the page.
- **autoplay:** This attribute, when present, specifies that the video should start playing automatically as soon as it is ready.
- **controls:** When present, this attribute adds video controls (play, pause, volume) to the video player.
- **preload:** This attribute tells the browser to load the video file(s) as soon as the document is loaded. It may not be suitable if not all users will play the video.
- **loop:** If present, this attribute specifies that the video should play continuously in a loop.

Source Elements:

Inside the <video> element, you use one or more <source> elements to specify the source of the video files. Each <source> element should include the src attribute, pointing to the URL of the video file, and the type attribute, indicating the MIME type of the file.

Fallback Content:

The text "Your browser does not support the video element." is included between the opening and closing <video> tags. This text is displayed if the browser does not support the <video> element.

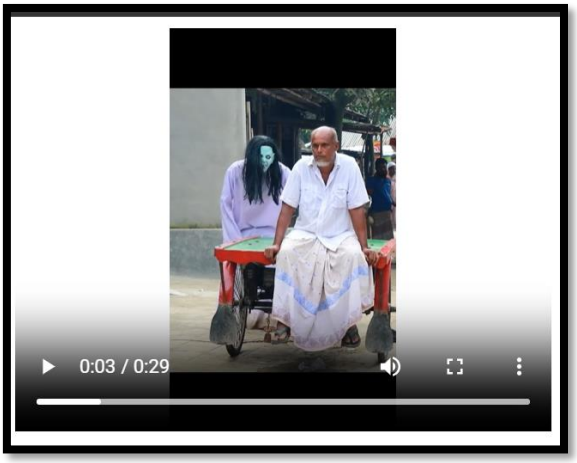
Browser Compatibility:

Different browsers support different video container/codec combinations. Multiple <source> elements allow you to provide alternative video formats, and the browser will choose the first one it supports.

Example:

```
<video width="600" height="500" autoplay controls preload>
  <source src="video_file.mp4" type="video/mp4">
  <source src="video_file.ogv" type="video/ogg">
  <source src="video_file.webm" type="video/webm">
  Your browser does not support the video element.
</video>
```

In this example, the browser will attempt to play the MP4 file first and fall back to Ogg and WebM if MP4 is not supported.

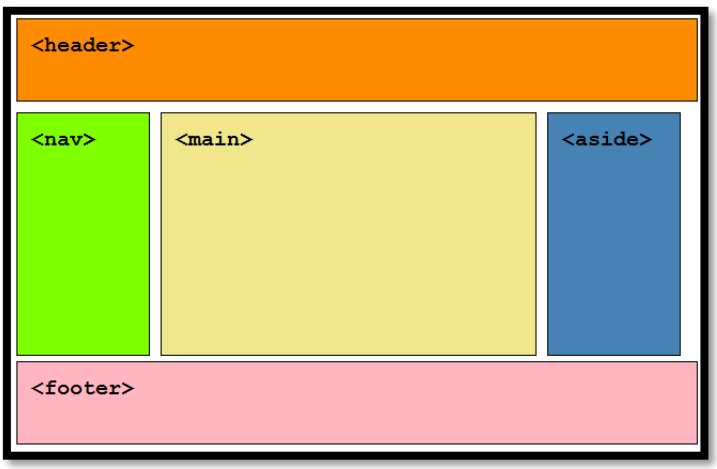


Notes:

- Provide alternative video formats for broader browser compatibility.
- Be aware of the file formats and codecs supported by different browsers (as mentioned in your provided text).
- The <video> element simplifies the process of integrating video content into web pages, making it more accessible and user-friendly.

ORGANIZATION ELEMENTS

In HTML5, several elements have been introduced to improve the organization of displayed information. Here's an explanation of some of these organization elements:



<header> Element:

The <header> element is used to represent introductory content at the beginning of a section or a page. It often contains headings, logos, and other elements related to the section or document. For example:

```
<header>
  <h1>The Podunk Press</h1>
  <h2>"All the news we can fit"</h2>
</header>
```

<hgroup> Element:

The <hgroup> element is used to group multiple heading elements (<h1>, <h2>, etc.) when they are part of a heading. It's useful for situations where additional information precedes the main heading. For example:

```
<hgroup>
  <header>
    <h1>The Podunk Press</h1>
    <h2>"All the news we can fit"</h2>
  </header>
  <!-- Additional content like a table of contents -->
</hgroup>
```

<footer> Element:

The <footer> element is used to represent footer content in a document. It's commonly used for information like copyright details and authorship. For example:

```
<footer>
  &copy; The Podunk Press, 2012
  <br />
  Editor in Chief: Squeak Martin
</footer>
```

<section> Element:

The <section> element is used to represent a generic section of a document. It's often used to encapsulate chapters of a book or separate parts of a paper. For example:

```
<section>
  <!-- Content of the section -->
</section>
```

<article> Element:

The <article> element is used to represent a self-contained part of a document, often coming from an external source. It can include a header, footer, and sections. For example:

```
<article>
  <header>
    <!-- Article header content -->
  </header>
  <!-- Main content of the article -->
  <footer>
    <!-- Article footer content -->
  </footer>
</article>
```

<aside> Element:

The <aside> element is used for content that is tangential or related to the main information in the document. It's often used for sidebars. For example:

```
<aside><!-- Content related to the main information --></aside>
```

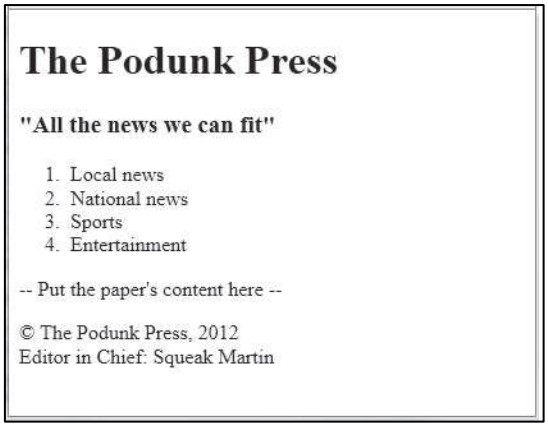
<nav> Element:

The <nav> element is used to represent a navigation section, typically containing lists of links to different parts of the document or other documents. It helps in marking navigation areas, especially useful for accessibility. For example:

```
<nav>
  <!-- Navigation links -->
</nav>
```

Example:

```
<!DOCTYPE html>
<!-- organized.html: An example to illustrate organization elements of HTML5
-->
<html lang="en">
<head>
  <title>Organization elements</title>
  <meta charset="utf-8" />
</head>
<body>
  <hgroup>
    <header>
      <h1>The Podunk Press</h1>
      <h3>"All the news we can fit"</h3>
    </header>
    <ol>
      <li>Local news</li>
      <li>National news</li>
      <li>Sports</li>
      <li>Entertainment</li>
    </ol>
  </hgroup>
  <p>
    <!-- Placeholder for the paper's content -->
    <!-- Put the paper's content here -->
  </p>
  <footer>
    &copy; The Podunk Press, 2012
    <br />
    Editor in Chief: Squeak Martin
  </footer>
</body>
</html>
```



Explanation:

- The document starts with the HTML5 doctype declaration.
- The <html> element has the lang attribute set to "en" for English.
- The <head> section contains the title of the document and the character set declaration.
- The <body> section includes several organization elements:
    - <hgroup> encapsulates the header information, which consists of an <header> containing <h1> and <h3> elements.
    - An ordered list (<ol>) with list items (<li>) represents different sections like local news, national news, sports, and entertainment.
    - A paragraph (<p>) provides a placeholder for the main content of the paper.
    - <footer> contains copyright and editor information.

The actual content of the paper should be placed inside the <p> element. The provided HTML serves as a template, and you can replace the placeholder text with the actual content for each section.

These elements provide a more semantic and structured way to organize content in HTML documents.

THE TIME ELEMENT

The time element in HTML5 is used to timestamp an article or a document. It consists of a textual part for human readability and a machine-readable part. The machine-readable part is represented by the datetime attribute, while the human-readable part is the content of the element.

Here's a breakdown of the provided information and an example:

```
<time datetime="2011-02-14T08:00" pubdate="pubdate">
  February 14, 2011 8:00am MDT
</time>
```

- **The datetime attribute:** This attribute holds the machine-readable date and time information. In the example, it is set to "2011-02-14T08:00", indicating February 14, 2011, at 8:00 AM.

- **The pubdate attribute:** This attribute is optional. If the time element is not nested within an article element, the pubdate attribute specifies that the timestamp is the publication date of the document. If nested inside an article element, it is the publication date of the article.

- **The content:** The content inside the time element is the human-readable representation of the date and time. In this example, it is "February 14, 2011 8:00am MDT". The information in the content is not necessarily related to the information in the datetime attribute.

- **Time zone offset:** The datetime attribute can include a time zone offset, represented as an offset in the range of –12:00 to +14:00 from Coordinated Universal Time (UTC). For example, "2011-02-14T08:00-06:00" indicates a time zone offset of UTC-6:00.

It's important to note the limitations of the time element: it cannot represent years before the beginning of the Christian era (negative years are not acceptable), and it does not allow for approximations (e.g., "circa 1900").

UNIT – 2

LEVELS OF STYLE SHEETS:

There are three levels of style sheets used in web design, each with a specific scope and precedence:

- Inline Style Sheets
- Document-level Style Sheets
- External Style Sheets

***1. Inline Style Sheets*:** These are style definitions that are applied directly within the HTML elements using the style attribute. They have the lowest level of precedence and only affect the specific element they are applied to. While convenient for individual element styling, they can lead toscattered and unmanageable style definitions within the document.

***example*:** An inline style sheet is defined directly within an HTML element using the style attribute. It affects only that specific element.

```
<p style="color: blue; font-size: 16px;">This is a blue and larger font-size paragraph. </p>
```

2. Document-level Style Sheets: These style sheets are defined within the <style> tags in the <head> section of an HTML document. They apply to the entire content of the document, providing a consistent style throughout. Document-level styles have a higher precedence than inline styles, allowing them to override properties defined inline.

example: A document-level style sheet is placed within the <style> tags in the <head> section of an HTML document. It applies to the entire content of the document.

```
<!DOCTYPE html>
<html>
<head>
   <style>
      body {
         font-family: Arial, sans-serif;
         background-color: #f0f0f0;
      }
      h1 {
         color: red;
      }
   </style></head>
```

```
<body>
    <h1>This is a red heading</h1>
    <p>This is a paragraph with the default font family.</p>
</body>
</html>
```

3. External Style Sheets: External style sheets are separate files with a .css extension that contain style rules. They are linked to HTML documents using the <link> element within the <head> section. External style sheets provide the highest level of separation between content and style, making it easier to maintain and apply consistent styles across multiple documents. They have the lowest precedence, allowing them to be overridden by both inline and document-level styles.

example: An external style sheet is a separate .css file that is linked to HTML documents using the <link> element in the <head> section.

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
    <h1>This is a green heading</h1>
    <p>This is a paragraph with the default font family.</p>
</body>
</html>

/* styles.css */
body {
    font-family: Verdana, sans-serif;
    background-color: #ffffff;
}
h1 {
color: green;
}
```

Precedence and Conflict Resolution:

When conflicting style rules are encountered for a specific element,the following precedence order is followed:

- Inline styles take precedence over document-level styles.
- Document-level styles take precedence over external style sheets.
- Lower-level style sheets (inline and document-level) have precedenceover higher-level style sheets (external).

In the case of conflicting property values for an element, the value specifiedin the lower-level style sheet will be applied.

Advantages of External Style Sheets:

- They promote separation of concerns by keeping CSS separate from HTML markup.
- They allow for consistent styling across multiple documents.
- They enhance maintainability and ease of updates.

It's important to note that inline style sheets, while useful for specific element customization, can lead to code redundancy and are generally discouraged for large-scale styling. Document-level and external style sheets are preferred for more organized and maintainable styles.

When using external style sheets, they can be validated using tools like theW3C CSS Validator.

Remember that browser defaults are used when no specific style is definedin any of the style sheets.

STYLE SPECIFICATION FORMATS

The format of a style specification depends on the level of the style sheet being used. Inline style specifications appear as values of the "style" attribute of a tag. The general form of this attribute is as follows:

style = "property_1: value_1; property_2: value_2; ...; property_n: value_n;"

Although it is not mandatory, it is recommended that the last property-value pair be followed by a semicolon.

1. An alternative to using the <link> tag for importing stylesheets is the @import rule. However, it's worth noting that the @import rule tends to be slower in loading styles, so there is usually no good reason to use it.
2. The "style" attribute is deprecated in the XHTML 1.1 recommendation. However, it is still a valid part of HTML5.

Document style specifications appear as the content of a <style> element within the <head> section of an HTML document. The format of the specification is quite different from that of inline style sheets. The general form of the content of a <style> element is as follows:

```
<style type="text/css">
rule_list
</style>
```

The "type" attribute of the <style> tag informs the browser about the type of style specification being used, which is typically set to text/css for CSS.

Each style rule within a rule list consists of two parts: a selector, which specifies the element or elements affected by the rule, and a list of property-value pairs. The list has a similar structure to that of inline style sheets, but it is enclosed within curly braces. The form of a style rule is as follows:

```
selector {
    property_1: value_1;
    property_2: value_2;
    ...
   property_n: value_n;
}
```

If a property is assigned multiple values, these values are generally separated by spaces. However, some properties require multiple values to be separated by commas.

Just like any other form of coding, complex CSS rule lists should be documented with comments. Since HTML comments cannot be used withinCSS, a different form of comment is required. CSS comments are introduced using /* and terminated with */, as shown in the following example:

```
<style type="text/css">
/* Styles for the initial paragraph */
...

/* Styles for other paragraphs */
...
</style>
```

An external style sheet consists of a list of style rules similar to those foundin document style sheets. The <style> tag is not used in external style sheets.

SELECTOR FORMS

A selector in Cascading Style Sheets (CSS) specifies the elements to which a particular style should be applied. Selectors come in various forms,each serving different purposes.

1. Simple Selector Forms

The simplest form of a selector is a single element name, such as h1, which targets all occurrences of that element. You can also create a list of element names separated by commas to target multiple elements, like h2, h3, which applies the style to both h2 and h3 elements.

2. Class Selectors

Class selectors allow you to style different occurrences of the same element uniquely. You define a style class by assigning a name with a period.

For instance, p.normal and p.warning define distinct classes. To apply these styles, use the class attribute in the HTML markup, such as <p class="normal"> and <p class="warning">.

3. Generic Selectors

Generic selectors are used when you want to apply a class of styles to various element types. A generic class is defined without an element name and starts with a period, like .sale. You can then use this class in your HTML, like <h3 class="sale">.

4. id Selectors

An id selector targets a specific element with a given id. Its form is #specific-id, for instance #section14. The corresponding style is applied to the element with that id, like <h2 id="section14">.

5. Contextual Selectors

Contextual selectors specify styles for elements in certain positions within the document. A descendant selector targets elements within the content of another element, like ul ol targeting ordered lists within unordered lists.

Child selectors (parent > child) apply to direct child elements. The first-child, last-child, and only-child selectors target specific positions. The empty selector is used for elements without child elements.

6. Pseudo Classes

Pseudo classes define styles based on interactions or conditions. The :link and :visited pseudo classes style unvisited and visited links. The :hoverpseudo class is for when the mouse cursor hovers over an element, and :focus is applied when an element gains focus.

7. The Universal Selector

The universal selector * applies a style to all elements in the document.

Here are examples to illustrate the different selector forms mentioned in your provided text within the context of a hypothetical HTML and CSS scenario:

Example: **Html Copy code**

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
    <h1>Welcome to CSS Selector Examples</h1>
    <p class="normal">This is a normal paragraph.</p>
    <p class="warning">This is a warning paragraph.</p>
    <h3 class="sale">Sale Items</h3>
    <ul>
        <li>Item 1</li>
        <ol>
            <li>Subitem A</li>
            <li>Subitem B</li>
```

```
        </ol>
        <li>Item 2</li>
    </ul>
    <a href="#" class="link">Unvisited Link</a>
    <a href="#" class="visited">Visited Link</a>
</body>
</html>
```

CSS Stylesheet (styles.css):

```
/* Simple Selector Forms */
h2, h3 {
    font-style: italic;
}
/* Class Selectors */
p.normal {
    font-weight: normal;
}
/* Generic Selectors */
.sale {
    background-color: yellow;}
/* id Selectors */
#section14 {
    border: 1px solid black;
}
/* Contextual Selectors */ul
ol {
    list-style-type: upper-roman;
}
ul > li {
    color: green;}
p:first-child {
    font-size: 18px;
}
/* Pseudo Classes */
a:link {
    text-decoration: none;
    color: blue;
}
/* Universal Selector */
* {
    margin: 0;
    padding: 0;}
```

Output:

Welcome to CSS Selector Examples
This is a normal paragraph.
This is a warning paragraph.
Sale Items
- Item 1
  1. Subitem A
  2. Subitem B
- Item 2

Unvisited Link
Visited Link

PROPERTY VALUE FORMS

CSS encompasses various properties grouped into categories such as fonts, lists, text alignment, margins, colors, backgrounds, and borders. A few common property values are discussed below.

1. Number Values:

Number values represent integers or decimal numbers with an optional sign (+ or -). For instance, font-size: 14px; sets the font size to 14 pixels.

2. Length Values:

Length values, like width, are numbers followed by a unit abbreviation (e.g., px, in, cm, mm, pt, pc). For instance, margin-left: 10mm;sets a left margin of 10 millimeters.

3. Percentage Values:

Percentage values are relative to a previous measure, often used for scaling. For example, width: 50%; sets the element's width to 50% of its parent's width.

4. URL Values:

URL property values are enclosed in parentheses after the url keyword, e.g., background-image: url(image.jpg);.

5. Color Values:

Color property values can be color names, hexadecimal numbers, or RGB format. For instance, color: red;, background-color: #336699;, and border-color: rgb(255, 0, 0); set the text color, background color, and bordercolor, respectively.

6. Inheritance:

Some property values are inherited by descendant elements. Forexample, setting font-size on the <body> tag affects the font size of all elements within the body. However, not all properties are inherited. For instance, background-color and margin properties are not inherited.

Example HTML and CSS:

Html code:

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
    <h1>Welcome to Property-Value Examples</h1>
    <p class="normal">This is a normal paragraph.</p>
    <p class="warning">This is a warning paragraph.</p>
</body>
</html>
```

CSS Stylesheet (styles.css):

```
/* Number Values */
h1 {
    font-size: 24px;
}
/* Length Values */
p.normal {
    margin-left: 20px;
}
/* Percentage Values */
p.warning {
    width: 75%;
}
/* URL Values */
body {
    background-image: url(background.jpg);
}
/* Color Values */p
{
    color: blue;
    background-color: #FFD700; /* Gold */ border:
    2px solid rgb(0, 128, 0); /* Green */}
```

Outuput:

Welcome to Property-Value Examples
This is a normal paragraph.
This is a warning paragraph.

FONT PROPERTIES

CSS provides various font properties to control the appearance of texton a web page. These properties include font-family, font-size, font-variant,font-style, font-weight, text-decoration, letter-spacing, word-spacing, and line-height.

1. Font-Family:

The font-family property specifies the font to be used for text. Multiple font names can be provided as alternatives in case the browser doesn't support the first choice. For example:

```
p {  font-family: Arial, Helvetica, sans-serif;}
```

2. Font-Size:

The font-size property determines the size of the font. It can be specified in absolute units like pixels (px), points (pt), or relative units like percentages (%) or em. For example:

```
h1 {
   font-size: 24px;

}
```

3. Font-Variant:

The font-variant property controls the display of capital letters. It canbe set to small-caps to display small capital letters:

```
p.capital {
   font-variant: small-caps;}
```

4. Font-Style:

The font-style property sets the style of the font, such as italic or oblique:

```
em {
   font-style: italic;}
```

5. Font-Weight:

The font-weight property determines the thickness of the font. Valueslike bold, normal, and numeric values from 100 to 900 are available:

```
strong {
    font-weight: bold;

}
```

6. Text-Decoration:

The text-decoration property adds special features to text, likeunderlining or strike-through:

```
a:hover {
    text-decoration: underline;

}
```

7. Letter-Spacing:

The letter-spacing property controls the space between letters in a word: p.tracked

```
{
   letter-spacing: 1px;

}
```

8. Word-Spacing:

The word-spacing property adjusts the space between words:

```
p.spaced {
    word-spacing: 5px;}
```

9. Line-Height:

The line-height property specifies the space between lines of text:

```
p.poetry { line-height: 1.5;}
```

Table 3.1 Generic fonts

Generic Name	Examples
Serif	Times New Roman, Garamond
Sans-serif	Arial, Helvetica
cursive	Caflisch Script, Zapf-Chancery
fantasy	Critter, Cottonwood
monospace	Courier, Prestige

If a font name has more than one word, the whole name should be delimited by single quotes,[12] as in the following example:

```
font-family: 'Times New Roman'
```

Example HTML and CSS:

Html code

```
<!DOCTYPE html>
<html>
<head>
   <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
   <p class="capital">This text has small capital letters.</p>
   <em>This text is italicized.</em>
   <strong>This text is bold.</strong>
   <a href="#">Hover over me</a>
   <p class="tracked">This text is spaced out.</p>
   <p class="spaced">These words are spaced apart.</p>
   <p class="poetry">Lines of poetry<br>with adjusted spacing.</p>
</body>
</html>
```

CSS Stylesheet (styles.css):

```
p {
   font-family: Arial, Helvetica, sans-serif;
   font-size: 16px;
}
/* Additional styles for different properties */
```

Output:

This text has small capital letters.This
text is italicized.
This text is bold.
Hover over me
This text is spaced out.
These words are spaced apart.
Lines of poetry
with adjusted spacing.

LIST PROPERTIES:

Two presentation details of lists can be specified in HTML documents: the shape of the bullets that precede the items in an unordered list and the sequencing values that precede the items in an ordered list. The list-style-type property is used to specify both of these. If list-style-type is set for a <ul> or an <ol> tag, it applies to all the list items in the list. If a list-style-type is set for an <li> tag, it only applies to that list item.

Unordered List Bullets:

The list-style-type property of an unordered list can be set to disc (the default), circle, square, or none. A disc is a small filled circle, a circle is an unfilled circle, and a square is a filled square.

Example Markup (styles.css):

```
<style type="text/css">
ul { list-style-type: square; }
</style>
...
<ul>
  <li>Item 1</li>
  <li>Item 2</li>
  <li>Item 3</li>
</ul>
```

Ordered List Sequencing

For ordered lists, the list-style-type property can be used to specify the types of sequencing values, such as decimal numerals, lowercase and uppercase letters, and Roman numerals.

Example Markup (styles.css):

```
<style type="text/css">
ol { list-style-type: upper-roman; } ol
ol { list-style-type: upper-alpha; }ol ol
ol { list-style-type: decimal; }
</style>
...
<ol>
  <li>First level
    <ol>
      <li>Second level
        <ol>
```

```
            <li>Third level</li>
          </ol>
        </li>
      </ol>
    </li>
  </ol>
```

Output:

- Item 1
- Item 2
- Item 3

I. First level
  A. Second level
    1. Third level

Explanation:

In the output, the unordered list items have square bullets, and the ordered list items are sequenced using upper-roman numerals for the first level, upper-alpha letters for the second level, and decimal numerals for thethird level.

Table 3.2 Possible sequencing value types for ordered lists in CSS2.1

Property Value	Sequence Type
`decimal`	Arabic numerals starting with `1`
`decimal-leading-zero`	Arabic numerals starting with `0`
`lower-alpha`	Lowercase letters
`upper-alpha`	Uppercase letters
`lower-roman`	Lowercase Roman numerals
`upper-roman`	Uppercase Roman numerals
`lower-greek`	Lowercase Greek letters
`lower-latin`	Same as `lower-alpha`
`upper-latin`	Same as `upper-alpha`
`armenian`	Traditional Armenian numbering
`georgian`	Traditional Georgian numbering
`None`	No bullet

The following example illustrates the use of different sequence value types in nested lists:

COLOR PROPERTIES IN WEB DOCUMENTS

Over the last decade, the issue of color in Web documents has become much more settled. In the past, one had to worry about the range of colors client machine monitors could display, as well as the range of colors browsers could handle. Now, however, there are few color limitations withthe great majority of client machine monitors and browsers.

Color Groups: There are three groups of predefined colors that were designed for Web documents:

1. The original group of seventeen named colors, which included far toofew colors to be useful.
2. A group of 147 named colors that are widely supported by browsers.
3. The so-called Web palette, which includes 216 named colors that wereonce the only predefined colors widely supported by browsers. Contemporary professional Web designers are more likely to define their own colors.

Color Properties Usage

The color property is used to specify the foreground color of HTML elements. For example, consider the following description of a small table:

Example Markup (styles.css):

```
<style type="text/css">
th.red { color: red; }
th.orange { color: orange; }
</style>
...
<table>
  <tr>
    <th class="red">Apple</th>
    <th class="orange">Orange</th>
    <th class="orange">Screwdriver</th>
  </tr></table>
```

The background-color property is used to set the background color of an element, where the element could be the whole body of the document. For example, consider the following paragraph element:

Example Markup (styles.css):

```
<style type="text/css">
p.standout {
  font-size: 2em;
  color: blue;
  background-color: magenta;
}
</style>
...
<p class="standout">
  To really make it stand out, use a magenta background!
</p>
```

Output:

When displayed by a browser, the text "To really make it stand out, use a magenta background!" will appear with blue font color and a magenta background color, as shown in the example.

Explanation:

The color property sets the color of the text, while the background- color property sets the background color of the element. In the provided example, the paragraph with class "standout" has blue text color and a magenta background color.

ALIGNMENT OF TEXT IN WEB DOCUMENTS

The presentation of text in web documents can be further enhanced by adjusting its alignment and indentation. CSS provides properties to controlthese aspects and improve the visual layout of content.

Text Indentation

The text-indent property is used to indent the first line of a paragraph. This property accepts length or percentage values. For instance:

Example Markup (styles.css):

```
<style type="text/css">
p.indent { text-indent: 2em; }
</style>
...
<p class="indent">
Now is the time for all good Web developers to begin using cascading style sheets for all presentation details in their documents. No more deprecated tags and attributes, just nice, precise style sheets.
</p>
```

Output:

The text within the paragraph with the class "indent" will be indentedby 2em (equivalent to two times the size of the current font), creating a visually appealing layout.

Text Alignment

The text-align property is used to horizontally arrange text within an element. Common keyword values are left, center, right, and justify.

Example Markup (styles.css):

```
<style type="text/css">
p.center-align { text-align: center; }
</style>
...
<p class="center-align">
This text is centered horizontally.
</p>
```

Output:

The text within the paragraph with the class "center-align" will be centered horizontally within its container.

Float Property

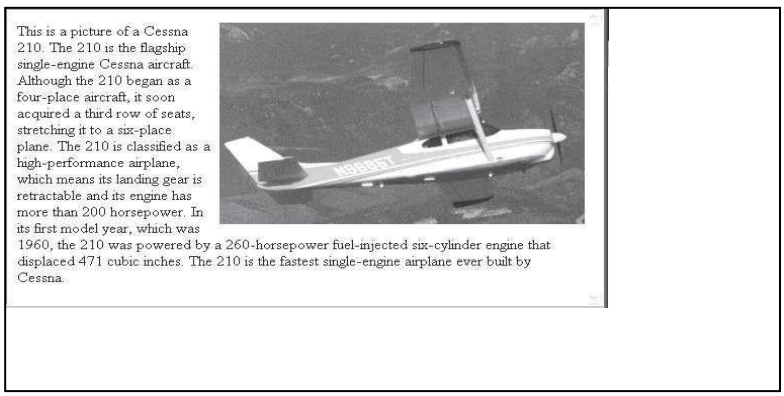
The float property is used to make text flow around elements likeimages. It accepts values left, right, and none.

Example Markup (float.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>The float property</title>
<meta charset="utf-8" />
<style type="text/css">
img { float: right; }
</style>
</head>
<body>
<p>
<img src="c210new.jpg" alt="Picture of a Cessna 210" />
</p>
<p>
<!-- Text describing the Cessna 210 -->
</p>
</body>
</html>
```

Output:

In the example, the image with the class "float" is floated to the right,and the text flows around it accordingly.



THE BOX MODEL IN CSS

The Box Model is a fundamental concept in CSS that defines how elements are structured and spaced within a web page. It encompasses the element's content, padding, border, and margin. These components work together to determine the overall appearance and spacing of elements on a webpage.

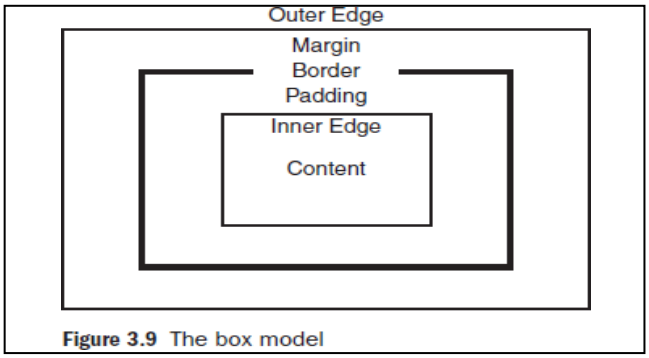


Figure 3.9 The box model

1. Borders and Border Styles

The border-style property controls whether an element's content has a border and specifies the style of the border. Various border styles includedotted, dashed, solid, and double. The default value is none.

Example Markup (styles.css):

```
<style type="text/css">
td, th { border: thin solid black; }
</style>
```

2. Border Width and Color

The border-width property specifies the thickness of a border, which can be set to thin, medium, thick, or a specific length value (in pixels). The border-color property sets the color of the border.

Example Markup (styles.css):

```
<style type="text/css">
td, th {
   border-top-width: medium;
   border-bottom-width: thick;
   border-top-color: red; border-
   bottom-color: blue; border-
   top-style: dotted;
   border-bottom-style: dashed;
}
</style>
```

3. Shorthand for Border Styles

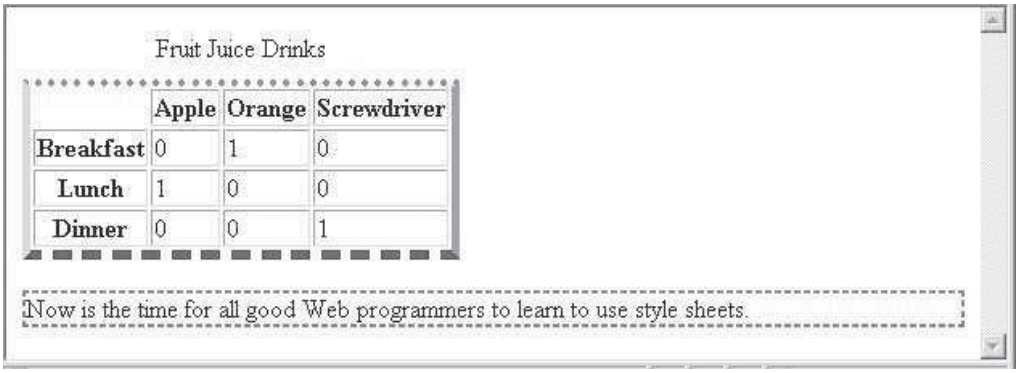
You can use shorthand to set multiple border properties at once.

Example Markup (styles.css):

```
.my-box {
   border: 2px dashed #3498db;
}
```

Overall Example Markup (styles.css):

```
<style type="text/css">
p { border: 5px solid blue; }
</style>
```



4.Margins and Padding

Margins create space between an element's border and its neighbors, while padding defines space between an element's content and its border.

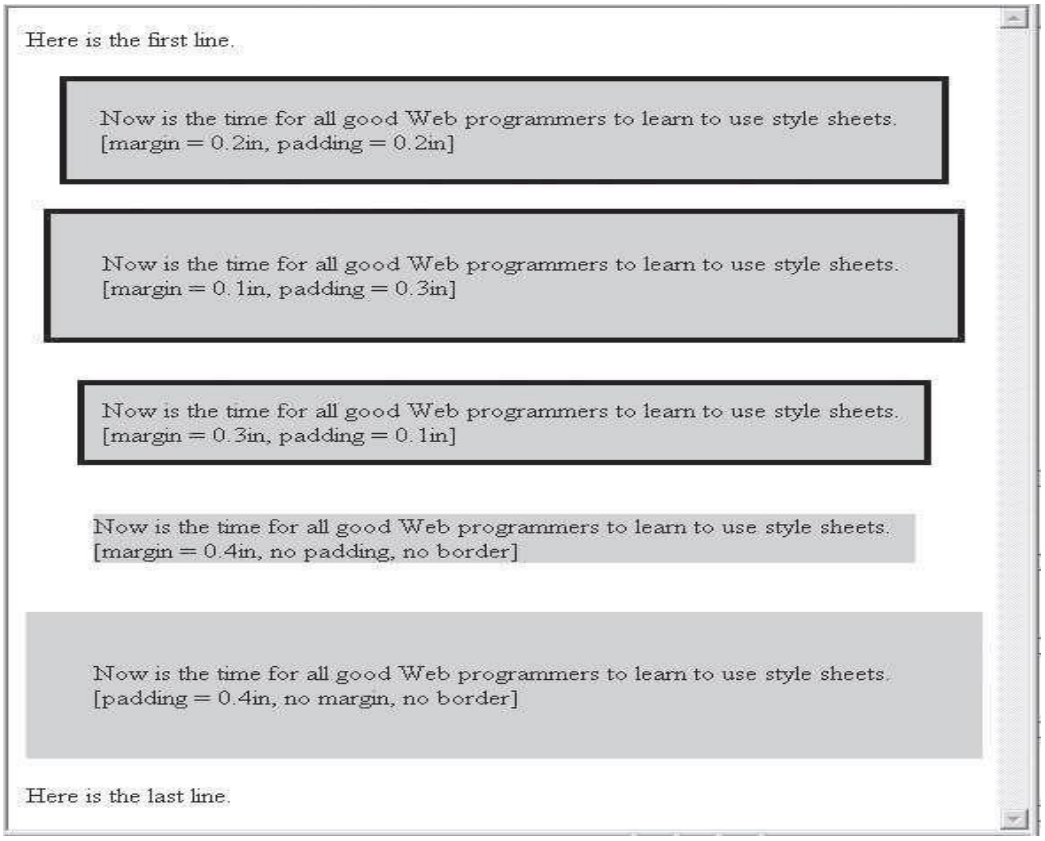
Example Markup (marpads.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Margins and Padding</title>
   <meta charset="utf-8" />
   <style type="text/css">
      p.one { margin: 15px; padding: 15px; background-color:
#C0C0C0; border-style: solid; }
      p.two { margin: 5px; padding: 25px; background-color:
#C0C0C0; border-style: solid; }
      p.three { margin: 25px; padding: 5px; background-color:
#C0C0C0; border-style: solid; }
      /* ... (other styles) ... */
   </style>
</head>
```

```
<body>
<p class="one">Now is the time for all good Web programmers tolearn
to use style sheets. [margin = 15px, padding = 15px]</p>
<p class="two">Now is the time for all good Web programmers tolearn to
use style sheets. [margin = 5px, padding = 25px]</p>
<p class="three">Now is the time for all good Web programmers
to
learn to use style sheets. [margin = 25px, padding = 5px]</p>
<!-- ... (other paragraphs) ... -->
</body>
</html>
```

Output:

The examples illustrate how margins and padding create spacearound elements and paragraphs, affecting their positioning and appearance within the webpage.



Explanation:

The Box Model consists of content, padding, border, and margin. Borders have styles, widths, and colors. Shorthand properties simplify border styling. Margins and padding control spacing around elements. Marpads.html demonstrates different margin and padding combinations.

BACKGROUND IMAGES

The background-image property is used to place an image in the background of an element. For example, an image of an airplane might be an effective background for text about the airplane. The following example, back_image.html.

The provided HTML code demonstrates the use of background images in combination with text. Here's the output of the given HTML code along with an explanation:

Html code:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Background images</title>
   <meta charset="utf-8" />
   <style type="text/css">
      body {
         background-image: url(../images/plane1.jpg);
         background-size: 375px 300px;
      }
      p {
         margin-left: 30px;
         margin-right: 30px;
         margin-top: 50px;
         font-size: 1.1em;
      }
   </style>
</head>
<body>
   <p>
      The Cessna 172 is the most common general aviation
airplane
      in the world. It is an all-metal, single-engine piston,
      high-wing, four-place monoplane. It has fixed gear and is
      categorized as a non-high-performance aircraft. The
current
      model is the 172R.
      <!-- ... (more text) ... -->
   </p>
</body>
</html>
```

Explanation:

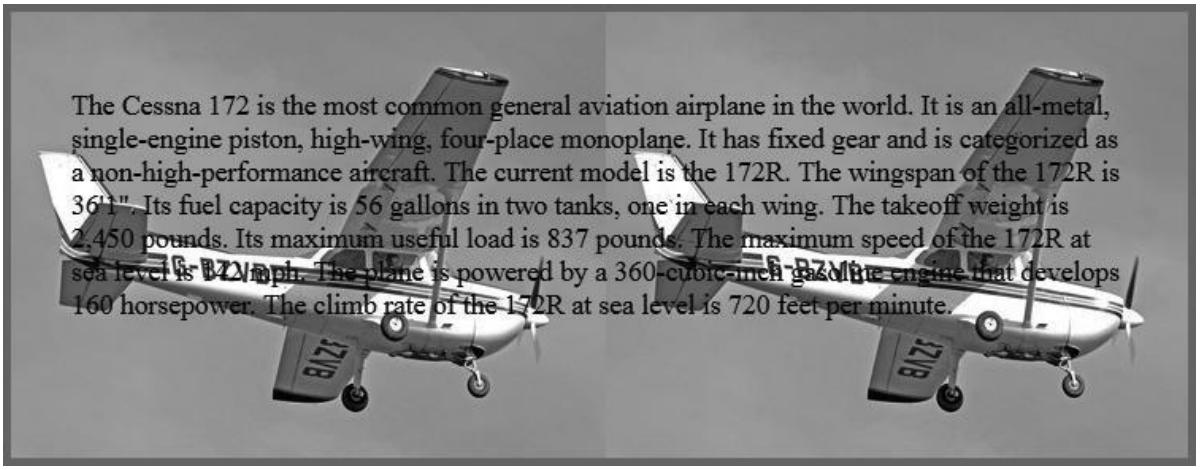
The given HTML code sets a background image for the entire body ofthe document and places text over the background image.

The <style> block within the <head> section sets the background image using the background-image property. It also specifies the dimensions of the background image using the background-size property.

The <body> section contains a paragraph (<p>) element that containstext about the Cessna 172 airplane. The text is indented from the left and right margins and is also pushed down from the top margin.

The background image plane1.jpg is tiled (repeated) to fill the background of the entire body.

Output:



THE<SPAN> AND <DIV>TAGS:

The provided text explains the use of the and elements for applying special font properties and styling sections of a document. Here's a breakdown of the explanation and an example for each element:

1. Using the Element:

The element is used to apply special font properties to a portion of text within a larger block, such as a paragraph. It doesn't have any default layout or styling. It's often used to change font size, color, or other text properties for a specific word or phrase.

Example: html code

```
<p>
   It sure is fun to be in <span style="font-size: 2em; font-family:Arial;
color: red;">total</span> control of text.
</p>
```

In this example, the word "total" is enclosed within a element, and a custom style is applied to it using the style attribute. The word "total" willappear larger, in the Arial font, and in red color.

2. Using the Element:

The element is used to create a block-level container that can be styled as a whole. It's often used to group and apply styles to sections of a document, allowing you to define presentation details for a group of elements.

Example: html code

```
<div class="primary">
   <p>...</p>
   <p>...</p>
   <p>...</p>
</div>
```

In this example, a element with the class name "primary" is used togroup and style a section of the document. You can then define CSS styles for the. primary class to apply specific presentation details to all theparagraphs within that .

Both the and elements are versatile tools for applying targeted styling to specific content or sections within an HTML document.

The display of this paragraph:

CONFLICT RESOLUTION:

When different values for the same property are specified on the same element in an HTML document, conflicts arise, and the browser needs to resolve them. The process of conflict resolution involves considering various factors such as the origin, specificity, and importance of the style declarations. Here's an explanation with an example:

Consider the following HTML document and external style sheet:

cstyle.css - an external style sheet*p*

{font-size: 0.8em;} cascade.html -

HTML document

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Style sheet levels</title>
   <meta charset="utf-8" />
   <link rel="stylesheet" type="text/css" href="cstyle.css" />
   <style type="text/css"> p.docstyle
      {font-size: 1.2em;}
   </style>
</head>
<body>
   <p>Now is the time</p>
   <p class="docstyle">for all good men</p>
   <p class="docstyle" style="font-size: 1.6em">to come to the
aid</p>
</body>
</html>
```

In this example, conflicts arise due to different font-size values specified forthe <p> elements. Let's break down the resolution process:

1. Precedence of Style Sheets:

Inline styles have higher precedence than document-level and externalstyle sheets. Thus, the style attribute takes precedence over the class-level and external styles.

2. Conflict Resolution Steps:

- The first paragraph uses the external style sheet (cstyle.css) to set font-size to 0.8em.
- The second paragraph has a class (docstyle) that specifies font-size: 1.2em in the document-level style sheet.
- The third paragraph has both the class and an inline style with font-size: 1.6em.

Here's how the conflicts are resolved:

- The first paragraph uses the external style sheet: font-size: 0.8em;
- The second paragraph uses the document-level style sheet: font-size: 1.2em;
- The third paragraph has an inline style with higher precedence: font-size: 1.6em;

Output:

- "Now is the time" - Font size: 0.8em
- "for all good men" - Font size: 1.2em
- "to come to the aid" - Font size: 1.6

UNIT - 3

THE BASICS OF JAVASCRIPT

OVERVIEW OF JAVASCRIPT

1. Origins of JavaScript

- **Developer:** Brendan Eich at Netscape.
- **Early Names:** Mocha, then LiveScript.
- **Joint Venture:** Netscape and Sun Microsystems.
- **Final Name:** JavaScript.
- **Standardization:** ECMA-262 by the European Computer Manufacturers Association (ECMA).
- **Current Standard:** ECMAScript (ECMA-262), version 5.

2. JavaScript Components

- **Three Parts:** Core, Client Side, Server Side.
- **Core:** Language basics - operators, expressions, statements.
- **Client-side:** Controls browsers and user interactions.
- **Server-side:** Supports server operations.

3. JavaScript vs. Java

- **Difference:** Despite the name, JavaScript and Java are different.
- **Object-Oriented:** JavaScript's object model is different from Java.
- **Typing:** JavaScript is dynamically typed, unlike Java's static typing.
- **Object Nature:** Java objects are static; JavaScript objects are dynamic.
- **Similarity:** Similar syntax in expressions, assignment, and control statements.

4. Uses of JavaScript

- **Original Goal:** Server and client programming.
- **Now:** Full-fledged language for various applications.
- **Focus in:** Client-side JavaScript.
- **Capabilities:** Interactions with users, dynamic content, event-driven programming.

5. Browsers and HTML-JavaScript Documents

- **Execution Flow:** Browser reads HTML, renders, executes JavaScript when encountered.
- **Embedding Methods:** Implicit (separate JavaScript file) and Explicit (in HTML document).
- **Placement:** Head for reactive scripts, body for scripts to be interpreted once.

6. Summary

JavaScript started at Netscape, evolved into a versatile language. It's different from Java, excelling in client-side interactions. It's widely used for creating dynamic content, handling user events, and enhancing web experiences. In this book, the focus is on client-side JavaScript, exploring its core features and applications.

OBJECT ORIENTATION AND JAVASCRIPT

1. Object-Based, Not Object-Oriented

- **JavaScript Nature:** Object-based, not object-oriented.
- **No Classes:** Lacks classes, unlike Java or C++.
- **Inheritance:** Prototype-based inheritance (not discussed here).
- **Polymorphism:** Limited without class-based inheritance.

2. JavaScript Objects

- **Properties:** Collections of properties (like members in Java or C++ classes).
- **Data vs. Functions:** Properties can be data or functions/methods.
- **Primitive Values:** Simple types (primitives) and references to objects.
- **Root Object:** Object is the root, ancestor of all objects.
- **Dynamic Properties:** Can add or delete properties during execution.

3. Accessing Objects

- **Variables as References:** Objects accessed indirectly through variables.
- **Primitive Values:** Accessed directly, similar to scalar types in other languages.
- **Root Object (Object):** Generic with methods, no data properties.

4. JavaScript Object Representation

- **Property–Value Pairs:** Objects represented as property–value pairs.
- **Dynamic Nature**: Properties can be added or deleted anytime during execution.
- **No Formal Types:** Objects have properties but no formal types.

5. Summary

JavaScript, although not strictly object-oriented, revolves around objects. Objects consist of properties, which can be data or functions. JavaScript uses prototype-based inheritance and lacks class-based inheritance, limiting polymorphism. The root object is Object, and objects are accessed through variables. The dynamic nature of JavaScript objects allows flexibility in property manipulation. More on objects is discussed in later sections.

GENERAL SYNTACTIC CHARACTERISTICS

1. **Direct Embedding:** JavaScript as content within <script> tags.
2. **Indirect Embedding:** Using the src attribute in <script> tags to reference an external file.
3. **Identifiers**: Start with a letter, underscore, or dollar sign, followed by letters, underscores, dollar signs, or digits.
4. **Case Sensitivity:** JavaScript identifiers are case sensitive.
5. **Reserved Words:** JavaScript has reserved words like if, function, return, etc.
6. **Comments:** Two types - single-line // and block comments /* ... */.
7. **Script Tag Handling:** Older browsers might not interpret <script>, causing issues.

Syntactic Details

- **Semicolon Usage:** JavaScript interpreter tries to insert semicolons automatically but might lead to unexpected behavior. It's safer to end statements with semicolons.
- **Code Organization:** Organize code with each statement on its own line for clarity and safety.
- **Best Practice:** Consider placing significant JavaScript code in separate files for better management and avoid embedding complexities.
- Example HTML Document with JavaScript
    - **Purpose:** Simple "Hello World" HTML document with a single line of JavaScript.

- **Structure:** DOCTYPE declaration, HTML tags, head, body, and a <script> tag containing JavaScript code using document.write.

This type of setup allows JavaScript to be used within HTML documents, making web pages interactive and dynamic.

The key takeaway is that while JavaScript can be embedded directly or indirectly in HTML documents, proper syntax, commenting, and code organization are crucial for compatibility and avoiding unexpected behavior, especially in older browsers.

Certainly! Here's the example HTML document with a simple "Hello World" message using JavaScript:

```
<!DOCTYPE html>
<!-- hello.html
A trivial hello world example of HTML/JavaScript
-->
<html lang="en">
<head>
   <title>Hello world</title>
   <meta charset="utf-8" />
</head>
<body>
   <script type="text/javascript">
      <!--
      document.write("Hello, fellow Web programmers!");
      // -->
   </script>
</body>
</html>
```

In this example:

- The <!DOCTYPE html> declaration defines the document type and version.
- HTML comments (<!-- ... -->) are used for documentation within the HTML.
- The <html> tag represents the root of an HTML document.
- The <head> section contains metadata like the document title and character set.
- The <body> section contains the content of the document.
- The <script> tag embeds JavaScript code. The type attribute specifies the script type as "text/javascript."
- The JavaScript code inside the <script> tag uses the document.write method to output the message "Hello, fellow Web programmers!" to the document.

This is a minimal example to demonstrate the integration of JavaScript within an HTML document. It showcases the basic structure and syntax for creating a simple dynamic web page.

PRIMITIVES, OPERATIONS, AND EXPRESSIONS

JavaScript has several primitive data types that represent simple values. These data types are immutable (cannot be changed) and are used to store individual, atomic values. Here are the main primitive data types in JavaScript:

1. **Number: Represents numeric values, including integers and floating-point numbers.**
    a. let integerNumber = 42;
    b. let floatingPointNumber = 3.14;
2. **String: Represents sequences of characters, enclosed in single (') or double (") quotes.**
    a. let greeting = 'Hello, World!';
    b. let name = "Alice";
3. **Boolean: Represents a logical value, either true or false.**
    a. let isTrue = true;
    b. let isFalse = false;
4. **Undefined: Represents an uninitialized variable or a function without a return value.**
    a. let undefinedVariable;

5. **Null: Represents the intentional absence of any object value.**
    a. let nullValue = null;
6. **Symbol: Introduced in ECMAScript 6 (ES6), symbols are unique and immutable primitives, often used as unique identifiers.**
    a. let uniqueKey = Symbol('description');

These primitive data types are the building blocks of JavaScript programs. They are passed by value (copied) when assigned to variables, making each variable independent of the others. Additionally, JavaScript has special objects that are closely related to some primitive types, such as the Number, String, and Boolean objects. These objects provide additional methods and properties for working with primitive values.

```
let stringObject = new String('Hello');

let length = stringObject.length; // length is 5
```

However, in most cases, developers use the primitive forms (like 'Hello') rather than the object forms (like new String('Hello')) for simplicity and efficiency.

Understanding primitive data types is essential for effective JavaScript programming, as it influences variable behavior, memory usage, and the way values are compared and manipulated in the language.

TYPE OF JAVASCRIPT OPERATOR WITH EXPLANATIONS:

Arithmetic Operators

1. Addition +: Adds two operands.
    a. let result = 5 + 3; // result is 8
2. Subtraction -: Subtracts the right operand from the left operand.
    a. let result = 7 - 4; // result is 3
3. Multiplication *: Multiplies two operands.
    a. let result = 6 * 2; // result is 12
4. Division /: Divides the left operand by the right operand.
    a. let result = 10 / 2; // result is 5
5. Modulus %: Returns the remainder when the left operand is divided by the right operand.
    a. let result = 15 % 4; // result is 3

6. Increment ++: Increases the value of a variable by 1.
    a. let count = 5;
    b. count++; // count is now 6
7. Decrement --: Decreases the value of a variable by 1.
    a. let count = 8;
    b. count--; // count is now 7

Assignment Operators

1. Assignment =: Assigns the value of the right operand to the left operand.
    a. let x = 10; // x is assigned the value 10
2. Compound Assignments (+=, -=, *=, /=): Perform the operation and then assign the result to the left operand.
    a. let num = 5;
    b. num += 3; // num is now 8 (equivalent to num = num + 3)

Comparison Operators

1. Equality ==: Returns true if the operands are equal.
    a. let isEqual = 5 == '5'; // isEqual is true (loose equality)
2. Strict Equality ===: Returns true if the operands are equal without type conversion.
    a. let isEqual = 5 === '5'; // isEqual is false (strict equality)
3. Inequality !=: Returns true if the operands are not equal.
    a. let notEqual = 7 != 5; // notEqual is true
4. Strict Inequality !==: Returns true if the operands are not equal without type conversion.
    a. let notEqual = 7 !== '7'; // notEqual is true
5. Greater Than >: Returns true if the left operand is greater than the right operand.
    a. let isGreater = 10 > 5; // isGreater is true
6. Less Than <: Returns true if the left operand is less than the right operand.
    a. let isLess = 3 < 8; // isLess is true
7. Greater Than or Equal To >=: Returns true if the left operand is greater than or equal to the right operand.
    a. let isGreaterOrEqual = 5 >= 5; // isGreaterOrEqual is true
8. Less Than or Equal To <=: Returns true if the left operand is less than or equal to the right operand.
    a. let isLessOrEqual = 4 <= 6; // isLessOrEqual is true

Logical Operators

1. Logical AND &&: Returns true if both operands are true.
    a. let result = (5 > 3) && (7 < 10); // result is true
2. Logical OR ||: Returns true if at least one operand is true.
    a. let result = (8 === 8) || (5 > 10); // result is true
3. Logical NOT !: Returns true if the operand is false and vice versa.
    a. let isFalse = !(3 > 5); // isFalse is true

Conditional (Ternary) Operator

1. Conditional (Ternary) Operator condition ? expr1 : expr2: If the condition is true, evaluates expr1; otherwise, evaluates expr2.
    a. let result = (8 > 5) ? 'Yes' : 'No'; // result is 'Yes'

Type Operators

1. 1.typeof: Returns a string indicating the type of the operand.
    a. let type = typeof 'Hello'; // type is 'string'
2. 2.instanceof: Returns true if an object is an instance of a specified object type.
    a. let isInstance = [1, 2, 3] instanceof Array; // isInstance is true

These operators are fundamental to JavaScript for performing various operations, comparisons, and logical evaluations, enabling dynamic and expressive programming.

EXPRESSIONS

In JavaScript, an expression is a combination of values, variables, operators, and functions that can be evaluated to produce a result. Expressions are the building blocks of JavaScript code, and they can be as simple as a single constant value or as complex as a combination of multiple operations. Here are some common types of expressions in JavaScript:

Arithmetic Expressions:

- let sum = 5 + 3; // Addition
- let difference = 8 - 2; // Subtraction
- let product = 4 * 6; // Multiplication
- let quotient = 9 / 3; // Division

String Concatenation:

- let greeting = 'Hello, ' + 'World!'; // Concatenation

Comparison Expressions:

- let isEqual = 10 === 10;   // Equality
- let isNotEqual = 5 !== '5'; // Inequality (strict)

Logical Expressions:

- let isTrue = true && false; // Logical AND
- let isFalse = true || false; // Logical OR
- let notTrue = !true;   // Logical NOT

Conditional (Ternary) Expression:

- let age = 20;
- let message = (age >= 18) ? 'Adult' : 'Minor';

Function Call Expressions:

- function square(x) {
  return x * x;
  }
- let result = square(4); // Calling the function

Array and Object Expressions:

- let numbers = [1, 2, 3, 4];
- let person = { name: 'Alice', age: 30 };

Template Literal Expression:

- let name = 'Bob';
- let greeting = `Hello, ${name}!`; // Template literal

Assignment Expressions:

- let x = 10;
- x += 5; // Equivalent to x = x + 5

SCREEN OUTPUT AND KEYBOARD INPUT

The process of handling screen output and keyboard input in JavaScript involves interacting with the Document Object Model (DOM) and using various methods provided by the Document and Window objects. Let's delve into these aspects:

Output with document.write()

A JavaScript script is typically interpreted by the browser when it encounters the script in the body of an HTML document. The normal output screen for JavaScript is the same as the screen in which the content of the HTML document is displayed. The Document object models the HTML document, and the Window object models the browser window.

The Document object has a method called write() that is used to dynamically create HTML document content. For example:

```
let result = 42;
document.write("The result is: ", result, "<br />");
```

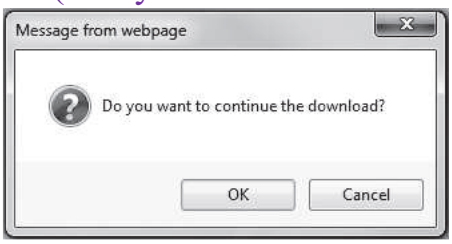
In this example, the write() method is used to create content in the HTML document dynamically.

Dialog Boxes with alert(), confirm(), and prompt()

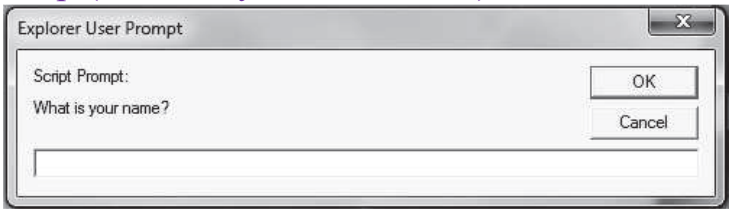
The Window object in JavaScript includes three methods that create dialog boxes for user interactions:

1. **alert():** Opens a dialog window displaying a message and an OK button.
   a. `let sum = 42;`
   b. `alert("The sum is: " + sum);`

2. **confirm():** Opens a dialog window with OK and Cancel buttons, returning true for OK and false for Cancel.
    a. let question = confirm("Do you want to continue this download?");



3. **prompt():** Opens a dialog window with a text box for user input, returning the entered value.
    a. let name = prompt("What is your name?", "");



The second parameter of prompt() is a default value.

These methods can be used for debugging or for interactive user experiences. They cause the browser to wait for user input.

Example: Quadratic Equation Roots

The example JavaScript script (roots.js) gets coefficients of a quadratic equation from the user using prompt() and computes and displays the real roots of the equation. If the roots are imaginary, it displays NaN (Not a Number). The associated HTML file (roots.html) includes a script tag to link to the JavaScript file.

```
var a = 1;
var b = -3;
var c = 2;

var root_part = Math.sqrt(b * b - 4.0 * a * c); // sqrt(1)
var denom = 2.0 * a; // 2

var root1 = (-b + root_part) / denom; // (3 + 1) / 2 = 2
var root2 = (-b - root_part) / denom; // (3 - 1) / 2 = 1

document.write("The first root is: ", root1, "<br />");
document.write("The second root is: ", root2, "<br />");
```

Output:

The first root is: 2
The second root is: 1

This script calculates and displays the roots of a quadratic equation based on user input.

These examples illustrate how JavaScript can interact with users, receive input, and dynamically create content within an HTML document.

CONTROL STATEMENTS

Control statements in JavaScript are essential for managing the flow of a program based on specific conditions. These statements allow developers to make decisions and execute different blocks of code depending on whether a given condition is true or false. There are primarily two types of control statements: Conditional Statements and Iterative Statements.

There are several methods that can be used to perform control statements in JavaScript.

- If Statement
- Using If-Else Statement
- Using Switch Statement
- Using the ternary operator (conditional operator)
- Using For loop

1. Conditional Statements:

Conditional statements are crucial for decision-making in JavaScript. These statements execute a block of code if a specified condition evaluates to true; otherwise, an alternative block is executed.

a. If Statement:

The if statement is the fundamental conditional statement. It executes a block of code if a given condition is true.

```
if ( condition_is_given_here ) {
    // If the condition is met,
    //the code  will get executed.
}
```

```
const num = 5;
if (num > 0) {
   console.log("The number is positive.");
}
```

Output:

```
The number is positive.
```

b. If-Else Statement:

The if-else statement extends the if statement by providing an alternative block of code to execute if the initial condition is false.

```
if (condition1) {
   // Executes when condition1 is true
   if (condition2) {
      // Executes when condition2 is true
   }
}
let num = -10;

if (num > 0) {
   console.log("The number is positive.");
} else {
   console.log("The number is negative.");
}
```

Output:

```
The number is negative.
```

2. Iterative Statements:

Iterative statements, also known as loops, allow a set of instructions to be executed repeatedly until a specified condition is met.

a***. For Loop:***

The for loop is a common iterative statement. It repeats a block of code for a specified number of iterations.

```
for (statement 1; statement 2; statement 3) {
    // Code here . . .
}
```

```
for (let i = 0; i <= 10; i++) {
    if (i % 2 === 0) {
        console.log(i);
    }
}
```

Output:

```
0
2
4
6
8
10
```

3. Switch Statement:

The switch statement provides an alternative way of handling multiple conditions. It is particularly useful when there are multiple possible execution paths based on the value of an expression.

```
switch (expression) {
    case value1:
        statement1;
        break;
    case value2:
        statement2;
        break;
    .
    .
    case valueN:
        statementN;
        break;
```

```
        default:
            statementDefault;
    }
```

```
let num = 5;

switch (num) {
    case 0:
        console.log("Number is zero.");
        break;
    case 1:
        console.log("Number is one.");
        break;
    case 2:
        console.log("Number is two.");
        break;
    default:
        console.log("Number is greater than 2.");
}
```

Output:

Number is greater than 2.

4. Ternary Operator:

The ternary operator (? :) is a concise way of expressing conditional statements. It is often used for simple, one-line conditions.

```
condition ? value if true : value if false
```

```
let num = 10;
let result = num >= 0 ? "Positive" : "Negative";
console.log(`The number is ${result}.`);
```

Output:

The number is Positive.

In conclusion, mastering control statements is fundamental for effective JavaScript programming. These statements empower developers to create dynamic and responsive applications by controlling the execution flow based on specific conditions or iterating through a set of instructions.

OBJECT CREATION AND MODIFICATION

In JavaScript, objects are a fundamental data structure that allows developers to organize and structure their code. Objects can be created and modified dynamically during runtime, providing flexibility and adaptability to the program. Let's delve into the basics of object creation and modification in JavaScript.

Object Creation with Constructors

Objects in JavaScript can be created using the new keyword along with a constructor function. Unlike some other programming languages, JavaScript objects are not bound to specific types. The constructor both creates and initializes the properties of the object.

```
// Creating an object using the Object constructor
var my_object = new Object();
```

In this example, my_object is initialized as a new, empty object using the Object constructor. The properties of the object can be added dynamically.

Adding Properties to Objects

Properties are the characteristics or attributes of an object. They can be added to an object using dot notation or by using the object literal notation.

```
// Adding properties to an object
my_object.make = "Ford";
my_object.model = "Fusion";
```

Alternatively, an object and its properties can be created in a single statement using object literal notation:

```
// Creating an object with properties
var my_car = { make: "Ford", model: "Fusion" };
```

In this example, my_car is created as an object with properties make and model using object literal notation.

Nested Objects

JavaScript allows objects to be nested within other objects, creating a hierarchical structure. This enables developers to organize data more effectively.

```
// Creating a nested object
my_car.engine = new Object();
my_car.engine.config = "V6";
my_car.engine.hp = 263;
```

In this example, the engine property of my_car is itself an object with properties config and hp.

Accessing and Deleting Properties

Properties of an object can be accessed using dot notation or square bracket notation:

```
// Accessing properties
var prop1 = my_car.make;          // Using dot notation
var prop2 = my_car["make"];       // Using square bracket notation
```

To delete a property from an object, the delete keyword can be used:

```
// Deleting a property
delete my_car.model;
```

Iterating Over Object Properties

JavaScript provides a for-in loop, which is useful for iterating over the properties of an object:

```
// Iterating over object properties
for (var prop in my_car) {
   document.write("Name: ", prop, "; Value: ", my_car[prop], "<br />");
}
```

This loop iterates over all properties of my_car, and for each iteration, prop takes on the name of the property, allowing easy access to both the property name and its value.

In conclusion, object creation and modification in JavaScript provide a dynamic and versatile way to structure data. Objects can be easily created, modified, and nested, making them a powerful tool for organizing and managing complex information in JavaScript programs.

ARRAYS

In JavaScript, arrays are versatile objects that can store a collection of elements. They can contain primitive values or references to other objects, including other arrays. Let's explore the basics of array creation, characteristics, methods, and examples.

1. Array Object Creation

Arrays in JavaScript can be created using the Array constructor or the literal array notation.

Using Array Constructor:

```
// Creating an array using the Array constructor
var my_list = new Array(1, 2, "three", "four");
var your_list = new Array(100);
```

Using Literal Array Notation:

```
// Creating an array using literal notation
var my_list_2 = [1, 2, "three", "four"];
```

Arrays created using the constructor can have a specified length, while arrays created using literal notation automatically adjust their length based on the number of elements.

2. Characteristics of Array Objects

- The lowest index of every JavaScript array is zero.
- Access to array elements is done using numeric subscript expressions.
- The length of an array is the highest subscript to which a value has been assigned, plus 1.
- The length property allows both reading and writing to dynamically change the array's length.

3. Array Methods

JavaScript provides several useful methods for working with arrays:

1. **join:** Converts all elements of an array to strings and concatenates them into a single string.
    a. `var names = ["Mary", "Murray", "Murphy", "Max"];`
    b. `var name_string = names.join(" : ");`

2. **reverse:** Reverses the order of elements in the array.
    a. names.reverse();
3. **sort:** Sorts the elements of the array either alphabetically or based on a provided comparison function.
    a. names.sort();
4. **concat:** Concatenates one or more arrays to the end of the calling array.
    a. var new_names = names.concat("Moo", "Meow");
5. **slice:** Returns a shallow copy of a portion of an array into a new array.
    a. var list = [2, 4, 6, 8, 10];
    b. var list2 = list.slice(1, 3);
6. **toString:** Converts each array element to a string and concatenates them, separated by commas.
    a. var names_string = names.toString();
7. **push and pop:** Add and remove elements from the end of an array, respectively.
    a. names.push("Dasher");
    b. var deer = names.pop();
8. **unshift and shift:** Add and remove elements from the beginning of an array, respectively.
    a. names.unshift("Dasher");
    b. var deer = names.shift();

4. Two-Dimensional Arrays

JavaScript implements two-dimensional arrays as arrays of arrays. They can be created using the Array constructor or nested array literals.

```
var nested_array = [[2, 4, 6], [1, 3, 5], [10, 20, 30]];
```

Accessing elements in a two-dimensional array is done using nested subscript expressions.

```
var element = nested_array[1][2]; // Accessing element at row 1, column 2
```

Arrays offer a flexible and powerful way to organize and manipulate data in JavaScript. They play a crucial role in various programming scenarios, including data storage, manipulation, and iteration.

FUNCTION

JavaScript functions play a crucial role in structuring code and performing specific tasks. Here are the fundamental concepts related to JavaScript functions, followed by an example illustrating their usage.

1. Function Fundamentals

A function definition consists of a header and a body. The header includes the reserved word function, the function name, and a list of parameters in parentheses.

```
function add(a, b) {
  // body of the function
  return a + b;
}
```

- The body of a function is a compound statement that describes the actions of the function.
- A return statement is used to return control from the function to the caller. It may include an expression whose value is returned to the caller.

JavaScript functions are objects and can be treated as such. They can be assigned to variables passed as parameters, and used as method references.

2. Local Variables

- Variables declared using var within a function have local scope, meaning they are only visible within the body of the function.
- Undeclared variables are implicitly global, even if used within a function. It is recommended to explicitly declare all variables using var to avoid unexpected behavior.

3. Parameters

- Parameters in a function call are actual parameters, while those in the function definition are formal parameters.
- JavaScript uses a pass-by-value parameter-passing method. Objects are passed by reference, but changes to the reference itself (not the object it points to) don't affect the caller.
- The arguments array provides access to all actual parameters, and its length property indicates the number of parameters passed.
- Variable numbers of parameters can be handled using the arguments array.

4. Example: Computing Median

Let's consider an example of a JavaScript function that computes the median of an array of numbers.

```
// Function to compute the median of an array
function median(list) {
  list.sort(function (a, b) { return a - b; });

  var list_len = list.length;

  if (list_len % 2 === 1)
    return list[Math.floor(list_len / 2)];
  else
    return Math.round((list[list_len / 2 - 1] + list[list_len / 2]) / 2);
}

// Test driver
var my_list_1 = [8, 3, 9, 1, 4, 7];
var my_list_2 = [10, -2, 0, 5, 3, 1, 7];

var med_1 = median(my_list_1);
var med_2 = median(my_list_2);

console.log("Median of", my_list_1, "is:", med_1);
console.log("Median of", my_list_2, "is:", med_2);
```

This function sorts the array in ascending order and then calculates the median, leaving the original array unchanged.

Median of [8, 3, 9, 1, 4, 7] is: 5.5

Median of [10, -2, 0, 5, 3, 1, 7] is: 3

Understanding these fundamentals is crucial for effective JavaScript programming. Functions enhance code modularity, reusability, and readability, making them a cornerstone of JavaScript development.

CONSTRUCTORS

Constructors in JavaScript are special functions that are used to create and initialize objects. When a constructor is invoked using the new keyword, it creates a new object and this variable inside the constructor refers to that newly created object. Constructors are crucial in establishing a blueprint for objects, providing a way to set initial properties and methods.

Here's a brief summary of the provided information:

```
// Example constructor for a car object
function Car(new_make, new_model, new_year) {
    // Initializing properties using 'this'
    this.make = new_make;
    this.model = new_model;
    this.year = new_year;
}

// Example method for the car object
function displayCar() {
    document.write("Car make: ", this.make, "<br/>");
    document.write("Car model: ", this.model, "<br/>");
    document.write("Car year: ", this.year, "<br/>");
}

// Adding the method to the Car constructor
Car.prototype.display = displayCar;

// Creating a new car object using the constructor
var my_car = new Car("Ford", "Fusion", "2012");

// Calling the display method on the car object
my_car.display();
```

In this example, the Car constructor creates a new car object with properties make, model, and year. Additionally, a method displayCar is defined, and then added to the Car constructor's prototype. This allows all instances created by the Car constructor to share the same method. The display method, when called on the my_car object, outputs information about the car. This approach is reminiscent of classes and objects in classical object-oriented programming, though JavaScript is a prototype-based language.

PATTERN MATCHING USING REGULAR EXPRESSIONS

1. Pattern Matching Methods

JavaScript provides two approaches for pattern matching:

- **RegExp Object:** Methods of the RegExp object.
- **String Object:** Methods of the String object. This explanation focuses on the String methods.

2. Delimiting Patterns

Patterns are based on regular expressions and are specified in a form delimited by slashes. For example:

```
var str = "Rabbits are furry";
var position = str.search(/bits/);
```

This searches for the pattern 'bits' in the string and returns the position.

3. Character and Character-Class Patterns

- **Normal Characters:** Match themselves.
- **Metacharacters:** Have special meanings (e.g., \ | ( ) [ ] { } ^ $ * + ? .).
- **Period (.):** Matches any character except newline.
- **Character Classes:** Defined by placing characters in brackets.

4. Predefined Character Classes

- \d: Digit
- \D: Not a digit
- \w: Word character (alphanumeric)
- \W: Not a word character
- \s: White-space character
- \S: Not a white-space character

5. Quantifiers

Numeric and symbolic quantifiers allow repetition in patterns (e.g., {4}, *, +, ?).

6. Anchors

Anchors (^ and $) specify that a pattern must match at the beginning or end of a string.

7. Pattern Modifiers

Modifiers like i (case-insensitive) and x (allows white space) can be used.

8. Other Pattern-Matching Methods

- **replace Method:** Replaces substrings that match a pattern.
- **match Method:** Returns an array of pattern-matching results.
- **split Method:** Splits the string into substrings based on a given pattern.

Example:

```
var str = "Having 4 apples is better than having 3 oranges";
var matches = str.match(/\d/g);
// matches is set to [4, 3]
```

This summary covers the key concepts of pattern matching using regular expressions in JavaScript. If you have specific questions or need further clarification, feel free to ask!

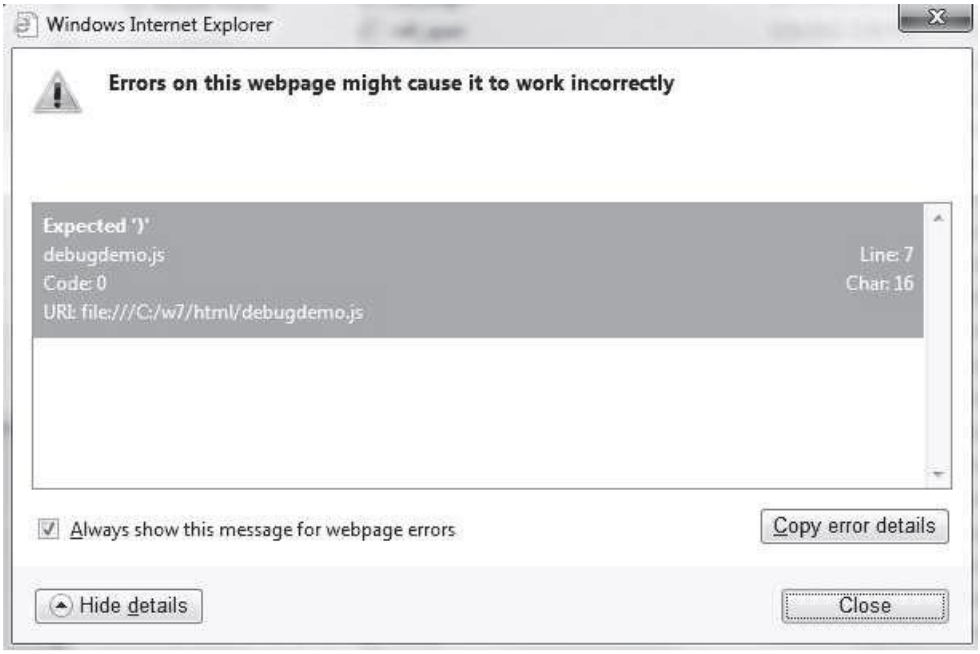
ERRORS IN SCRIPTS

1.Errors in Scripts:

- JavaScript interpreter detects syntax errors and undefined variable usage.
- Debugging in JavaScript can be different due to errors detected while browsers attempt to display documents.

2. Debugging Assistance:

- Default settings for IE8 provide JavaScript syntax error detection and debugging.
- IE9 and successors may have these features turned off by default. To enable them, go to Tools/Internet Options > Advanced tab.
- Internet Explorer settings: Remove the check on "Disable script debugging (Internet Explorer)" and set the check on "Display a notification about every script error."

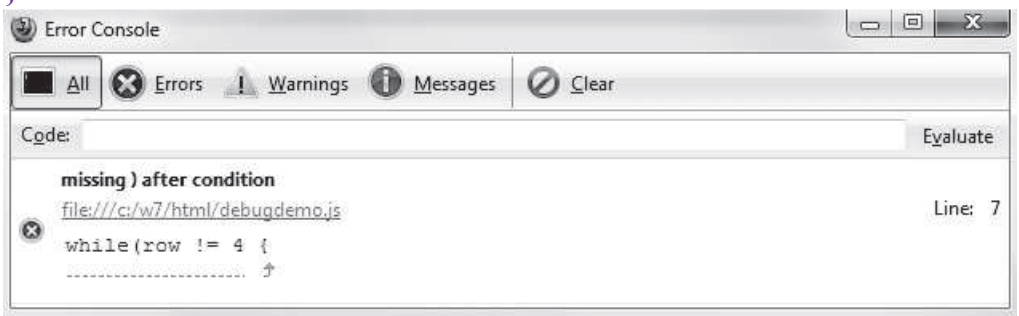


3. Debugging in Internet Explorer:

- IE8 shows syntax error detection and error messages with line and character positions.

Example:

```
var row;
row = 0;
while(row != 4 {
  document.write("row is ", row, "<br />");
  row++;
}
```



The error (missing right parenthesis) is displayed in a small window.

4. Firefox (FX) Browsers:

- FX3+ browsers have a special console window displaying script errors.
- Access the console through Tools/Web Developer/Error Console.
- Clear old error messages from the console to avoid confusion.

5. Chrome Browsers:

- In Chrome, access the JavaScript error console by selecting the upper-right icon (three horizontal bars), Tools, JavaScript console.

6. Debugging with Debuggers:

- Interesting and challenging programming problems may require debugging during execution or interpretation.
- IE and FX browsers have built-in JavaScript debuggers.
- IE10: Click on Tools/Developer Tools.
- FX browsers: Venkman debugger is available at mozilla.org, and Firebug debugger is available as an addon here.

This summary covers the key points related to debugging JavaScript scripts, including error detection and debugging tools in different browsers. If you have specific questions or need further clarification, feel free to ask!

JAVASCRIPT AND XHTML DOCUMENTS

THE JAVASCRIPT EXECUTION ENVIRONMENT

The JavaScript execution environment is the context in which JavaScript code is run, and it involves various objects and elements that facilitate the interaction between JavaScript and the browser. Here are key components of the JavaScript execution environment:

1. Window Object:

- The Window object represents the browser window or a tab.
- All global variables become properties of the Window object.
- It serves as the global object for client-side JavaScript.
- Properties include document, location, navigator, etc.

2. Document Object:

- The Document object represents the HTML document being displayed.
- Accessed through the document property of the Window object.
- Provides methods like getElementById, getElementsByTagName for DOM manipulation.
- Commonly used for writing content to the document using methods like write.

3. Global Variables:

- Variables declared outside functions become properties of the Window object.
- Accessible globally in the script.

4. Forms and Form Elements:

- The forms array of the Document object contains all form elements in the document.
- Each form has an elements array with references to individual form elements.
- Form elements can be accessed and manipulated through JavaScript.

5. Object Hierarchy:

- Beyond Window and Document, there's a hierarchy of objects (e.g., Element, Node, etc.).
- The Document Object Model (DOM) represents the structure of the HTML document as a tree of objects.

6. Execution Flow:

- JavaScript code is executed sequentially, from top to bottom.
- Events, such as user interactions, can trigger the execution of specific JavaScript code.

7. Error Handling:

- JavaScript interpreters detect syntax errors during script execution.
- Browsers often provide tools (e.g., Developer Tools) for debugging and error handling.

8. Debugging Tools:

- Browsers offer debugging tools (e.g., Chrome DevTools, Firefox Developer Tools) for inspecting, debugging, and profiling JavaScript code.
- Understanding the JavaScript execution environment is crucial for effective web development, as it enables the manipulation of HTML documents and the creation of dynamic and interactive web pages.

THE DOCUMENT OBJECT MODEL

Overview:

- The DOM has been developed by the W3C (World Wide Web Consortium) to provide a standardized interface for interacting with HTML and XML documents.
- DOM Level 3 (DOM 3) is the current approved version.
- DOM 0 refers to the version implemented in Netscape 3.0 and Internet Explorer 3.0 browsers.

DOM Versions:

1. ***DOM 0:***
    a. Used by early browsers supporting JavaScript (Netscape 3.0 and IE 3.0).
    b. Partially documented in the HTML 4 specification.
2. ***DOM 1:***
    a. First W3C DOM specification (October 1998).
    b. Focused on the HTML and XML document model.
3. ***DOM 2:***
    a. Issued in November 2000.
    b. Specified a style-sheet object model, document traversals, and a comprehensive event model.
4. ***DOM 3:***
    a. Issued in 2004, focused on XML content models, document validation, document views, and key events.
    b. Not covered in this book.

Abstract Model:

- DOM is an abstract model because it needs to apply to various programming languages.
- Consists of interfaces defining objects, methods, and properties for different node types.

JavaScript and DOM Binding:

- In JavaScript, elements in the DOM are represented as objects with properties and methods.
- Properties represent data (e.g., attributes), and methods perform operations on these objects.

Document Structure:

- Documents in the DOM have a tree-like structure.
- Each node type has an associated interface defining its properties and methods.
- Relationship representation in an implementation may vary.

DOM Inspection:

- Browsers provide tools for viewing the DOM structure of a displayed document.
- IE10: Tools/Developer Tools.
- Firefox (FX3+): DOM Inspector add-on.
- Chrome: Wrench icon, Tools, Developer Tools.

Purpose:

- The DOM allows the creation, manipulation, and dynamic modification of HTML and XML documents.
- Provides a standardized interface for cross-browser compatibility.

Resources:

- Detailed documentation of the DOM is available on the W3C website.

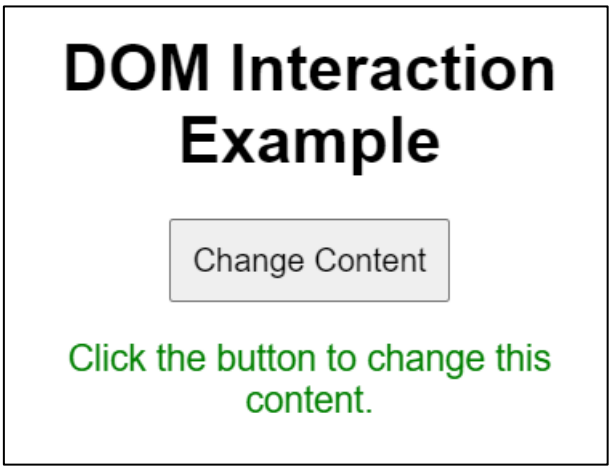
The DOM is essential for web developers working with JavaScript to create dynamic and interactive web pages. It provides a structured way to interact with and manipulate the content of HTML and XML documents.

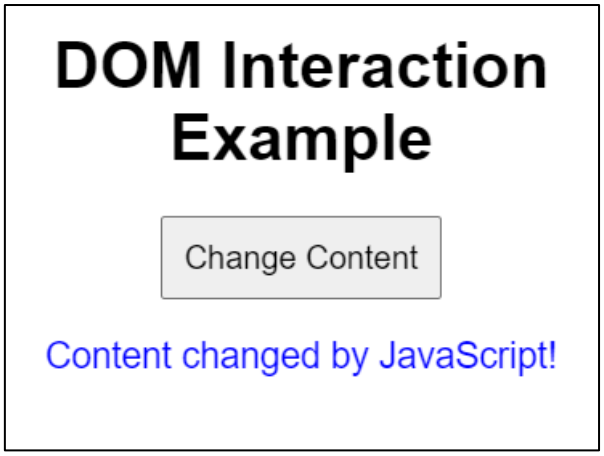
Certainly! Let's consider a simple example of how you can use JavaScript to interact with the **DOM**. In this example, we'll create an HTML page with a button, and when the button is clicked, it will change the content of a paragraph.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>DOM Interaction Example</title>
  <style>
    /* Some basic styling for demonstration purposes */
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      margin: 50px;
    }
    button {
      padding: 10px;
```

```
            font-size: 16px;
        }
        p {
            font-size: 18px;
            color: green;
        }
    </style>
</head>
<body>

    <h1>DOM Interaction Example</h1>

    <button id="changeContentBtn">Change Content</button>
    <p id="contentParagraph">Click the button to change this content.</p>

    <script>
        // JavaScript code to interact with the DOM

        // Get the button and paragraph elements by their IDs
        var changeContentBtn = document.getElementById("changeContentBtn");
        var contentParagraph = document.getElementById("contentParagraph");

        // Add a click event listener to the button
        changeContentBtn.addEventListener("click", function () {
            // Change the content of the paragraph when the button is clicked
            contentParagraph.textContent = "Content changed by JavaScript!";
            contentParagraph.style.color = "blue";  // Change text color for demonstration
        });
    </script>

</body>
</html>
```

This example demonstrates the following:





ELEMENTS ACCESS IN JAVA SCRIPT

In JavaScript, there are multiple ways to access elements in the HTML document. Here are some common methods:

1. **Using document.getElementById:**
    a. var element = document.getElementById("elementId");
    b. This method retrieves an element by its unique ID.
2. **Using document.getElementsByClassName:**
    a. var elements = document.getElementsByClassName("className");
    b. This method returns a collection of elements with the specified class name.
3. **Using document.getElementsByTagName:**
    a. var elements = document.getElementsByTagName("tagName");
    b. This method returns a collection of elements with the specified tag name.

4. **Using document.querySelector:**
    a. var element = document.querySelector("CSS selector");
    b. This method returns the first element that matches the specified CSS selector.
5. **Using document.querySelectorAll:**
    a. var elements = document.querySelectorAll("CSS selector");
    b. This method returns a NodeList containing all elements that match the specified CSS selector.
6. **Using document.getElementsByName:**
    a. var elements = document.getElementsByName("name");
    b. This method returns a collection of elements with the specified name attribute.

Traversing the DOM:

You can navigate the DOM hierarchy using properties like parentNode, childNodes, nextSibling, and previousSibling. For example:

```
var parentElement = childElement.parentNode;
```

Remember to replace "elementId," "className," "tagName," "CSS selector," and "name" with the actual identifiers or names you want to use for element access.

EVENTS AND EVENT HANDLING

Events and event handling are fundamental concepts in web development, particularly when working with JavaScript to create interactive and dynamic user interfaces. Let's break down these concepts:

Events:

An event is an occurrence or happening, often initiated by the user or the browser, that can be detected and responded to by JavaScript. Common events include:

1. ***User-Initiated Events:***
    a. **Click:** The user clicks the mouse button.
    b. **Mouseover/Mouseout:** The mouse pointer enters/leaves an element.
    c. **Keydown/Keypress/Keyup:** A keyboard key is pressed/pressed and released/released.

2. ***Document-Related Events:***
    a. **Load:** The HTML document or an external resource has finished loading.
    b. **Unload:** The user leaves the current page, and the document is about to be unloaded.
    c. **Resize:** The browser window is resized.
3. ***Form Events:***
    a. **Submit:** A form is submitted.
    b. **Change:** The value of a form element changes.
    c. **Focus/Blur:** An element gains/losses focus.

Events	Tag Attribute
blur	onblur
change	onchange
click	onclick
dblclick	ondblclick
focus	onfocus
keydown	onkeydown
keypress	onkeypress
keyup	onkeyup
load	onload
mousedown	onmousedown
mousemove	onmousemove
mouseout	onmouseout
mouseover	onmouseover
mouseup	onmouseup
reset	onreset
select	onselect
submit	onsubmit
unload	onunload

Event Handling:

Event handling is the process of defining and executing actions (JavaScript code) in response to specific events. In web development, you use event handlers to specify what should happen when an event occurs.

There are different approaches to event handling:

1. **Inline Event Handling:**
    a. Events can be directly specified as attributes in HTML tags. For example:
        i. <button onclick="handleClick()">Click me</button>

2. **DOM Event Handling:**
    a. You can use JavaScript to register event handlers programmatically. This approach provides more flexibility and separation of concerns.

```
<button id="myButton">Click me</button>
    <script>
    document.getElementById("myButton").onclick =
    function() {
    alert("Button clicked!");
    };
    </script>
```

3. **addEventListener:**
    a. A modern approach is to use the addEventListener method to attach event listeners to elements. This method allows attaching multiple listeners to the same event.

```
<button id="myButton">Click me</button>
    <script>
    document.getElementById("myButton").addEventList
    ener("click", function() {
    alert("Button clicked!");
    });
    </script>
```

HANDLING EVENTS FROM BODY ELEMENTS

Handling events from <body> elements involves executing JavaScript code in response to specific events occurring in the body of an HTML document. The <body> element can trigger various events, and JavaScript can be used to handle these events. One common event is the load event, which occurs when the HTML document has been completely loaded.

Let's consider an example of handling the load event in the <body> element:

HTML Document (index.html):

```
<!DOCTYPE html>
<!-- load.html
A document for load.js
-->
<html lang = "en">
<head>
```

```
<title> load.html </title>
<meta charset = "utf-8" />
<script type = "text/javascript" src = "load.js" >
</script>
</head>
<body onload="load_greeting();">
<p />
</body>
</html>
```

JavaScript File(load.js)

```
// An example to illustrate the load event
// The onload event handler
    function load_greeting () {
    alert("You are visiting the home page of \n" +
    "Pete's Pickled Peppers \n" + "WELCOME!!!");
    }
```

In this example:

1. The <body> element has an onload attribute set to "handleLoadEvent()". This attribute specifies that the handleLoadEvent function should be executed when the body is loaded.
2. The handleLoadEvent function, defined in the script.js file, contains JavaScript code that will be executed when the load event occurs. In this case, it displays an alert indicating that the body has been loaded.
3. The JavaScript file (script.js) is included in the <head> section of the HTML document using the <script> tag.
4. When the HTML document is loaded in a web browser, the handleLoadEvent function is triggered by the load event, and an alert is shown to the user.

This is a simple example, and in practice, you might use the load event to perform more complex tasks, such as initializing variables, making API requests, or setting up event listeners for other elements on the page. The load event is just one of many events that can be handled in the <body> element. Other events include click, mouseover, keydown, etc., depending on the desired interaction or behavior.

HANDLING EVENTS FROM TEXT BOX AND PASSWORD ELEMENTS

Handling events from text boxes and password elements is crucial for validating user input and ensuring data integrity. In the provided examples, JavaScript is used to check and validate the input in real-time, providing immediate feedback to users.

Example 1: Preventing Changes in Total Cost (Blur Event)

In the first example, the focus event is used to prevent changes to the total cost in a coffee order form. This is done to ensure the integrity of the order before submission.

HTML Document (nochange.html):

```
<!-- nochange.html -->
<html lang="en">
<head>
 <title>nochange.html</title>
 <meta charset="utf-8" />
 <script type="text/javascript" src="nochange.js"></script>
 <style type="text/css">
  td, th, table {border: thin solid black}
 </style>
</head>
<body>
 <form action="">
  <h3>Coffee Order Form</h3>
  <table>
   <!-- Table entries for coffee order -->
  </table>
  <p>
   <input type="button" value="Total Cost" onclick="computeCost();" />
   <input type="text" size="5" id="cost" onfocus="this.blur();" />
  </p>
  <p>
   <input type="submit" value="Submit Order" />
   <input type="reset" value="Clear Order Form" />
  </p>
 </form></body></html>
```

JavaScript File (nochange.js):

```
// nochange.js
function computeCost() {
  // Compute total cost and display
}
```

Example 2: Password Checking (Focus and Blur Events)

In the second example, the focus and blur events are used to check and verify passwords entered by the user. The chkPasswords function is triggered on blur and form submission to ensure password consistency.

HTML Document (pswd_chk.html):

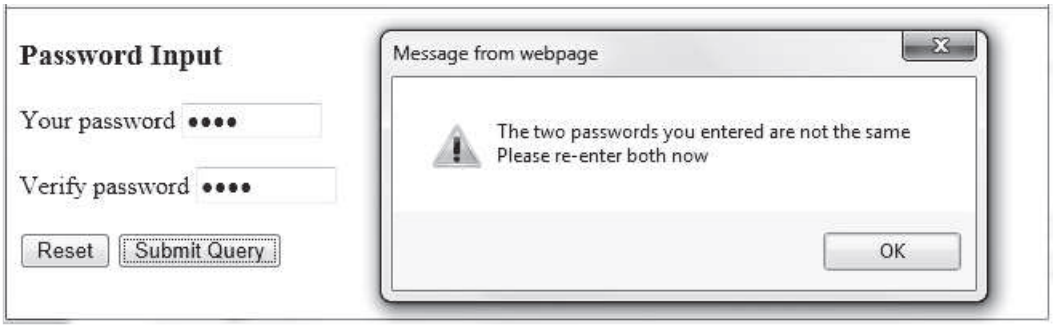
```
<!-- pswd_chk.html -->
<html lang="en">
<head>
  <title>Password Checking</title>
  <meta charset="utf-8" />
  <script type="text/javascript" src="pswd_chk.js"></script>
</head>
<body>
  <h3>Password Input</h3>
  <form id="myForm" action="">
    <p>
      <label>Your password
        <input type="password" id="initial" size="10" />
      </label>
      <br /><br />
      <label>Verify password
        <input type="password" id="second" size="10" />
      </label>
      <br /><br />
      <input type="reset" name="reset" />
      <input type="submit" name="submit" />
    </p>
  </form>
  <script type="text/javascript" src="pswd_chkr.js"></script>
</body>
</html>
```

JavaScript File (pswd_chk.js):

```
// pswd_chk.js
function chkPasswords() {
  // Check and verify passwords
}
```

JavaScript File (pswd_chkr.js):

```
// pswd_chkr.js
// Register the event handlers for pswd_chk.html
document.getElementById("second").onblur = chkPasswords;
document.getElementById("myForm").onsubmit = chkPasswords;
```



These examples illustrate how event handling in JavaScript can be utilized to enhance user interaction and validate input in web forms. The focus, blur, and submit events play crucial roles in ensuring a smooth and secure user experience.

DOM 2 EVENT MODEL

The DOM 2 event model is a more sophisticated and powerful approach compared to the DOM 0 event model. It is modularized, with one of its modules being the "Events" module, which includes submodules like HTMLEvents and MouseEvents. The table below outlines the interfaces and event types defined by these modules:

Module	Event Interface	Event Types
HTMLEvents	Event	abort, blur, change, error, focus, load, reset, resize, scroll, select, submit, unload
MouseEvents	MouseEvent	click, mousedown, mousemove, mouseout, mouseover, mouseup

Event Propagation:

In the DOM 2 event model, when an event occurs, an event object implementing the associated interface is implicitly passed to the event handler. The event object contains properties with information related to the event.

Event propagation in the DOM 2 model involves three phases:

1. **Capturing Phase:** The event propagates from the document root node down to the target node, triggering handlers registered on nodes encountered during this phase (excluding the target node).
2. **Target Node Phase:** Handlers registered specifically for the target node are executed.
3. **Bubbling Phase:** The event bubbles back up the document tree to the document node, triggering handlers registered on nodes encountered during this phase.

Not all events bubble; for example, load and unload events do not. Handlers can stop further propagation using the stopPropagation method of the event object.

Preventing Default Actions:

To prevent default actions associated with an event, the preventDefault method of the event object can be used. It is equivalent to returning false in the DOM 0 event model.

Event Handler Registration:

Event handler registration in the DOM 2 event model is done using the addEventListener method, which is defined in the EventTarget interface. The method takes three parameters: the event name, the handler function, and a boolean indicating whether the handler is enabled for the capturing phase.

Example:

```
document.custName.addEventListener("change", chkName, false);
```

Temporary event handlers can be created and removed using the removeEventListener method.

Example using DOM 2 Event Model:

The provided example is a revision of a form input validation script using the DOM 2 event model. It includes event handlers for the "change" event on text input elements for name and phone number validation. The registration script is as follows:

```
var customerNode = document.getElementById("custName");
var phoneNode = document.getElementById("phone");
customerNode.addEventListener("change", chkName, false);
phoneNode.addEventListener("change", chkPhone, false);
```

Note: The DOM 2 event model can be mixed with the DOM 0 event model in a document. The choice depends on convenience and compatibility considerations.

UNIT-4

DYNAMIC DOCUMENTS WITH JAVASCRIPT AND XML

INTRODUCTION

The passage you provided outlines the concept of Dynamic HTML (DHTML) and its use in web development. Let me break down the key points:

Dynamic HTML Definition: Dynamic HTML is not a new markup language itself but rather a collection of technologies that enables dynamic changes to HTML documents. These changes can be made in response to user interactions or browser events after the document has been initially loaded and displayed.

Modification through Scripting: Dynamic changes in a DHTML document are achieved through embedded scripts. These scripts access elements of the document as objects in the Document Object Model (DOM) structure. The DOM represents the structure of a document as a tree of objects, each corresponding to an element in the document.

Uniformity Issues Across Browsers: Support for DHTML is not consistent across different web browsers. The discussion in the passage focuses on W3C-standard approaches rather than features specific to a particular browser vendor. It mentions that the examples provided work on Internet Explorer 8 (IE8) and Firefox 3 (FX3) using the DOM 0 event model.

DOM Event Models: The DOM 0 event model is predominantly used in the examples, with an exception in Section 6.11, which utilizes the DOM 2 event model. The choice depends on the specific requirements of the example. The passage notes that the example using DOM 2 event model does not work with IE8 but is compatible with IE9 and later versions, which support the DOM 2 event model.

User Interaction with JavaScript: The chapter primarily focuses on user interactions through HTML documents using client-side JavaScript. It implies that JavaScript is used to handle events and make dynamic changes to the document in response to user actions.

Server-Side Technologies: User interactions through HTML documents using server-side technologies. This indicates a broader exploration of handling user interactions on the server side of web development.

Overall, the passage provides an overview of the concepts and challenges associated with implementing Dynamic HTML, with a focus on standard approaches and cross-browser compatibility.

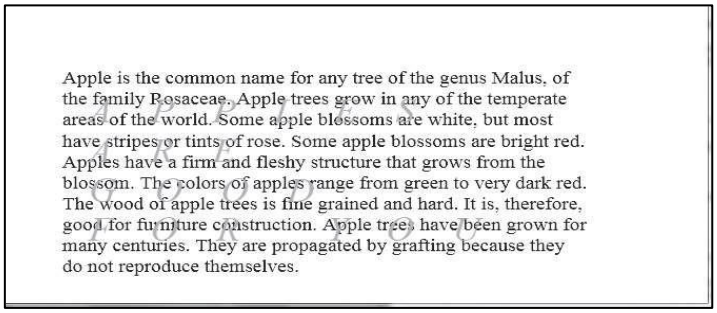
POSITIONING ELEMENTS

Introduction of CSS-P: The World Wide Web Consortium (W3C) released Cascading Style Sheets–Positioning (CSS-P) in 1997, offering a solution to the limitations of traditional HTML layout. CSS-P allows web developers not only to position elements anywhere in a document but also to dynamically move elements using JavaScript by changing the positioning style properties.

Positioning Properties in CSS: CSS-P introduces several positioning properties, including left, top, and position. The left and top properties dictate the distance from the left and top of a reference point to where an element should appear. The position property has three possible values: absolute, relative, and static.

1. **Absolute Positioning:** Specifies that an element should be placed at a specific location in the document display without regard to the positions of other elements. ***Example:***

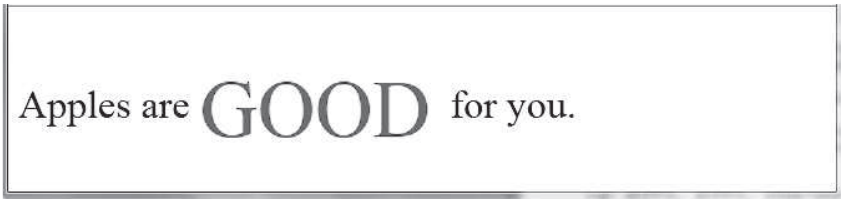
```
<p style="position: absolute; left: 100px; top: 200px;">-- text --</p>
```



2. **Relative Positioning:** Places an element in the document as if the position attribute were not set at all. However, the element can be moved later by specifying values for top and left.

Example:

```
<p style="position: relative; top: 100px; left: 100px;">-- text --</p>
```



3. **Static Positioning:** The default value for the position property. A statically positioned element is placed in the document as if it had position: relative with no values for top or left. A statically positioned element cannot be initially displaced or moved later.

Examples of Positioning: The passage provides examples of absolute positioning for creating effects like watermarks and relative positioning for special text effects, such as superscripts and subscripts.

Overall, the passage highlights how CSS-P revolutionized the control over the layout and positioning of HTML elements on web pages.

MOVING ELEMENT:

Using JavaScript to move an HTML element within a document by dynamically changing its top and left CSS properties. Here's a breakdown of the provided code:

HTML Structure (mover.html):

- The HTML file contains a form with two text input fields labeled 'x-coordinate' and 'y-coordinate'. These fields allow users to input new values for the left and top properties of an image.
- There's a button labeled 'Move it', which triggers the moveIt() function when clicked.
- An image of Saturn is initially positioned absolutely within a element.

JavaScript Logic (mover.js):

- The moveIt() function is defined in the JavaScript file. This function takes three parameters: the ID of the element to be moved (movee), the new top position (newTop), and the new left position (newLeft).
- Inside the moveIt() function, it retrieves the DOM element to be moved using document.getElementById(movee).
- It then updates the top and left CSS properties of the specified element with the new values obtained from the user input. The values are concatenated with "px" to ensure they are interpreted as pixel units in CSS.

HTML Structure (mover.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Moving elements</title>
   <meta charset="utf-8" />
   <!-- Include the JavaScript file -->
   <script type="text/javascript" src="mover.js"></script>
</head>
<body>
   <!-- Form for user input -->
   <form action="">
      <p>
         <!-- Text input for x-coordinate -->
         <label>
            x-coordinate:
            <input type="text" id="leftCoord" size="3" />
         </label>
         <br />
         <!-- Text input for y-coordinate -->
         <label>
            y-coordinate:
            <input type="text" id="topCoord" size="3" />
         </label>
         <br />
         <!-- Button to trigger the moveIt function -->
         <input type="button" value="Move it" onclick="moveIt('saturn',
document.getElementById('topCoord').value,
document.getElementById('leftCoord').value)" />
      </p>
   </form>
   <!-- Container div with initial absolute positioning -->
   <div id="saturn" style="position: absolute; top: 115px; left: 0;">
      <!-- Image of Saturn -->
      <img src="../images/saturn.png" alt="(Picture of Saturn)" />
   </div>
</body>
</html>
```

JavaScript Logic (mover.js):

```
// Illustrates moving an element within a document
// The event handler function to move an element
function moveIt(movee, newTop, newLeft) {
   // Get the style property of the element to be moved
   var dom = document.getElementById(movee).style;

   // Change the top and left properties to perform the move
   // Note the addition of units ("px") to the input values
   dom.top = newTop + "px";
   dom.left = newLeft + "px";
}
```

Display:

Figure likely represents the initial display of the mover.html page before any movement occurs.

After clicking the 'Move It' button with new coordinates entered, Figurewould show the updated position of the Saturn image based on the user-inputted values for top and left.

ELEMENT VISIBILITY

Document elements can be specified to be visible or hidden with the value of their visibility property. The two possible values for visibility are, quite naturally, visible and hidden. The appearance or disappearance of an element can be controlled by the user through a widget. The following example displays an image and allows the user to toggle a button, causing the image to appear and not appear in the document display (once again, the event handler is in a separate file):

HTML Structure (showHide.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8">
   <title>Visibility Control</title>
   <!-- Include the JavaScript file -->
   <script type="text/javascript" src="showHide.js"></script>
</head>
<body>
   <!-- Form with a div containing an image of Saturn -->
   <form action="">
      <div id="saturn" style="position: relative; visibility: visible;">
         <img src="../images/saturn.png" alt="(Picture of Saturn)" />
      </div>
      <!-- Button to toggle the visibility of the Saturn image -->
      <p>
         <br />
       <input type="button" value="Toggle Saturn" onclick="flipImag()" />
      </p>
   </form>
</body>
</html>
```

JavaScript Logic (showHide.js):

```
// showHide.js
// Illustrates visibility control of elements

// The event handler function to toggle the visibility of the image of Saturn
function flipImag() {
```

```
    var dom = document.getElementById("saturn").style;

    // Flip the visibility adjective to whatever it is not now
    if (dom.visibility == "visible") {
       dom.visibility = "hidden";
    } else {
       dom.visibility = "visible";
    }
}
```

Explanation:

HTML Structure:

- The HTML file includes a form containing a with an image of Saturn initially set to visible.
- A button with the text "Toggle Saturn" triggers the flipImag() function when clicked.

JavaScript Logic:

- The flipImag() function toggles the visibility of the Saturn image.
- It retrieves the style property of the element with the ID "saturn" using document.getElementById("saturn").style.
- The visibility is toggled by checking the current visibility value. If it's "visible," it changes it to "hidden," and vice versa.

This example demonstrates a basic way to use JavaScript to toggle the visibility of an element in response to user interaction.

CHANGING COLOR AND FONTS

When users hover over links, they often notice a change in color. This effect is achieved using the mouseover event in JavaScript. This event allows us to alter various properties of any element when the cursor hovers over it. This includes changing the font style, font size, color, and background color.

In JavaScript, if an attribute in CSS has a single word without hyphens, like color, the associated property name in JavaScript is the same. However, if an attribute includes a hyphen, like font-size, the associated property name in JavaScript must be different. The convention is to remove the hyphen and capitalize the following letter. For example, font-size becomes fontSize.

1. Changing Colors Dynamically:

HTML Structure (dynColors.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Dynamic Colors</title>
   <meta charset="utf-8" />
   <script type="text/javascript" src="dynColors.js"></script>
</head>
<body>
   <p style="font-family: Times; font-style: italic; font-size: 2em;">
      This small page illustrates dynamic setting of the foreground and
background colors for a document.
   </p>
   <form action="">
      <p>
         <label>
            Background color:
            <input type="text" name="background" size="10"
onchange="setColor('background', this.value)" />
         </label>
         <br />
         <label>
            Foreground color:
            <input type="text" name="foreground" size="10"
onchange="setColor('foreground', this.value)" />
         </label>
         <br />
      </p>
   </form>
</body>
</html>
```

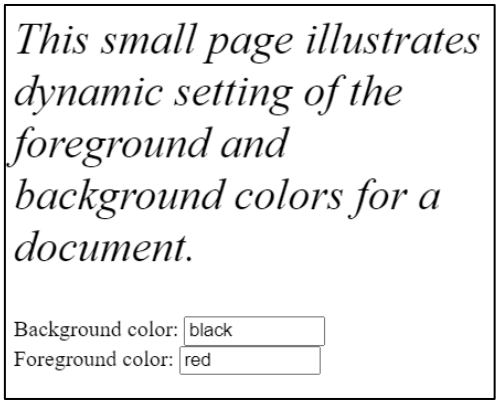
JavaScript (dynColors.js):

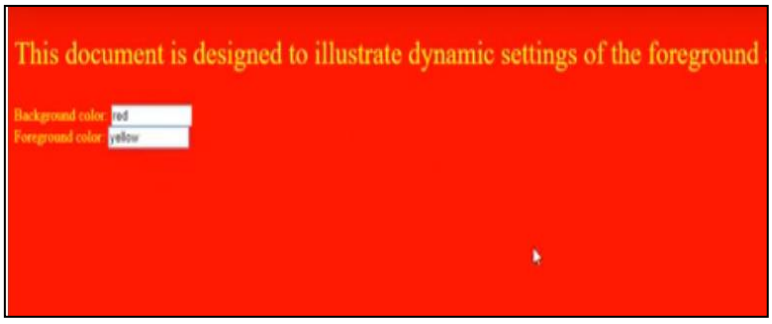
```
// dynColors.js
// Illustrates dynamic foreground and background colors
```

```
// The event handler function to dynamically set the color of background
or foreground
function setColor(where, newColor) {
    if (where === "background") {
        document.body.style.backgroundColor = newColor;
    } else {
        document.body.style.color = newColor;
    }
}
```

In this example:

- Users are presented with text boxes to input color values for the document's background and foreground.
- JavaScript (dynColors.js) provides an event handler (setColor) that changes either the background or foreground color based on user input.





2. Changing Fonts Dynamically:

HTML Structure (dynFont.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Dynamic Fonts</title>
    <meta charset="utf-8" />
```

```
    <style type="text/css">
      .regText {font: 1.1em 'Times New Roman';}
      .wordText {color: blue;}
    </style>
  </head>
  <body>
    <p class="regText">
      The state of
      <span class="wordText"
          onmouseover="changeFont(this, 'red', 'italic', '2em')"
          onmouseout="changeFont(this, 'blue', 'normal', '1.1em')">
        Washington
      </span>
      produces many of our nation's apples.
    </p>
  </body>
  </html>
```

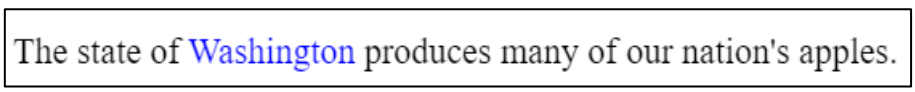
JavaScript (dynFont.js):

```
// dynFont.js
// Illustrates dynamic font styles and colors
// The function to dynamically change font styles
function changeFont(element, color, style, size) {
  element.style.color = color;
  element.style.fontStyle = style;
  element.style.fontSize = size;
}
```

In this example:

- The word "Washington" is styled with a specific font, color, and size initially.
- JavaScript (dynFont.js) provides a function (changeFont) to dynamically change the font color, style, and size when the mouse cursor hovers over or leaves the word.

These examples illustrate how you can use JavaScript to dynamically manipulate the appearance of elements in response to user interactions, providing a more interactive and engaging user experience.



DYNAMIC CONTENT

We've explored dynamic changes like moving elements, changing visibility, colors, background colors, and text font styles. Now, let's dive into changing the content of HTML elements dynamically. The content of an element is accessed through the value property of its associated JavaScript object. So, altering the content is similar to changing style properties.

Consider a scenario where a user needs assistance filling out a form. An associated text area, often referred to as a help box, can dynamically change its content based on the element the cursor hovers over. For instance, when the cursor is over a specific input field, the help box displays advice on how to fill in that field. When the cursor moves away, the help box content changes to indicate general assistance.

In the example below, an array of messages is defined in JavaScript to be displayed in the help box. The mouseover event triggers a handler function that changes the help box content to the appropriate message associated with the input field. The mouseout event resets the content back to a standard message.

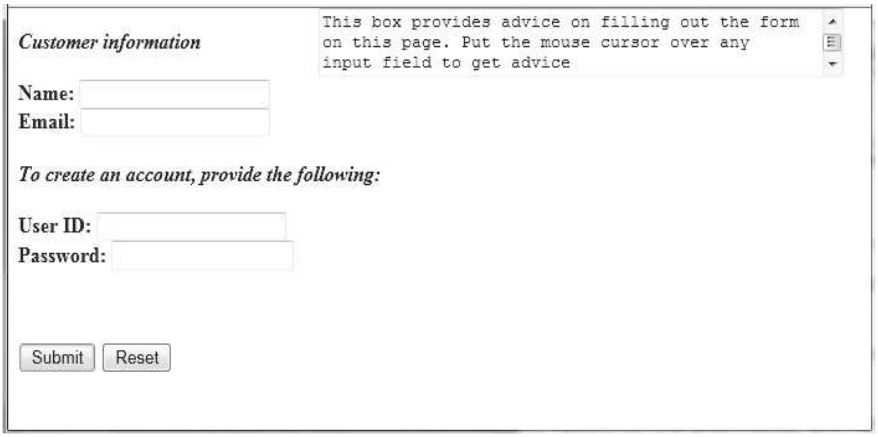
HTML Structure (dynValue.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Dynamic values</title>
   <meta charset="utf-8" />
   <script type="text/javascript" src="dynValue.js"></script>
   <style type="text/css">
      textarea {position: absolute; left: 250px; top: 0px;}
      span {font-style: italic;}
      p {font-weight: bold;}
   </style>
</head>
<body>
   <form action="">
      <p>
         <span>Customer information</span><br /><br />
         <label>Name: <input type="text" onmouseover="messages(0)"
onmouseout="messages(4)" /></label><br />
```

```
            <label>Email: <input type="text" onmouseover="messages(1)"
onmouseout="messages(4)" /></label><br /><br />
            <span>To create an account, provide the following:</span><br
/><br />
            <label>User ID: <input type="text" onmouseover="messages(2)"
onmouseout="messages(4)" /></label><br />
            <label>Password: <input type="password"
onmouseover="messages(3)" onmouseout="messages(4)" /></label><br
/>
        </p>
        <textarea id="adviceBox" rows="3" cols="50">This box provides
advice on filling out the form on this page. Put the mouse cursor over any
input field to get advice.</textarea>
        <br /><br />
        <input type="submit" value="Submit" />
        <input type="reset" value="Reset" />
    </form>
</body>
</html>
```

JavaScript (dynValue.js):

```
// dynValue.js
// Illustrates dynamic values
var helpers = [
    "Your name must be in the form: \nfirst name, middle initial., last
name",
    "Your email address must have the form: user@domain",
    "Your user ID must have at least six characters",
    "Your password must have at least six characters and it must include
one digit",
    "This box provides advice on filling out the form on this page. Put the
mouse cursor over any input field to get advice"
];

// The event handler function to change the value of the textarea
function messages(adviceNumber) {
    document.getElementById("adviceBox").value =
helpers[adviceNumber];
}
```



This example showcases how dynamic content can be used to provide helpful tips to users as they interact with a form. The help box dynamically changes its content based on user actions.

STACKING ELEMENT

This web page utilizes HTML, CSS, and JavaScript to demonstrate dynamic stacking of images. The stacking effect is achieved by manipulating the z-index attribute, allowing elements to overlap in a three-dimensional space. The z-index determines the stacking order, with higher values bringing elements to the forefront.

HTML Structure:

- Three images (plane1, plane2, and plane3) are positioned using absolute coordinates with initial z-index values set to 0.
- Each image has an onclick attribute triggering the toTop JavaScript function, facilitating dynamic stacking.

JavaScript (stacking.js):

- The toTop function is responsible for rearranging the stacking order.
- It receives the ID of the clicked image and updates the z-index values accordingly.
- The global variable topp keeps track of the currently topmost image.

CSS Styles:

- Each image is styled with a specific class (plane1, plane2, plane3) defining its initial position and z-index.

Example Execution:

- Initial Display (Figure 1): All images have a z-index of 0.
- After Clicking Second Image (Figure 2): The second image's z-index becomes 10, bringing it to the top.
- After Clicking Bottom Image (Figure 3): The bottom image's z-index is set to 10, placing it above the others.
- This dynamic stacking technique enhances user interaction, enabling the user to bring different images to the forefront by clicking on them.

LOCATING MOUSE CURSOR COORDINATES EXAMPLE

This example demonstrates how to capture and display the coordinates of the mouse cursor when a click event occurs. The provided HTML document (where.html) includes two pairs of text boxes for displaying the clientX, clientY, screenX, and screenY properties. The associated JavaScript file (where.js) defines an event handler to extract and present these coordinates.

HTML Structure (where.html):

- Two pairs of text boxes are embedded in a form to display mouse cursor coordinates.
- The findIt function is triggered by the onclick attribute of the <body> element.
- An image is included to add visual interest to the webpage.

JavaScript (where.js):

- The findIt function takes the event object (evt) as a parameter, capturing mouse click details.
- Using getElementById, it updates the values of the text boxes with the corresponding cursor coordinates (clientX, clientY, screenX, screenY).

Implementation Details:

- clientX and clientY represent coordinates relative to the upper-left corner of the browser window.
- screenX and screenY represent coordinates relative to the client's computer screen.
- The event object (evt) is implicitly created and passed to the handler function by the onclick event.

HTML (where.html):

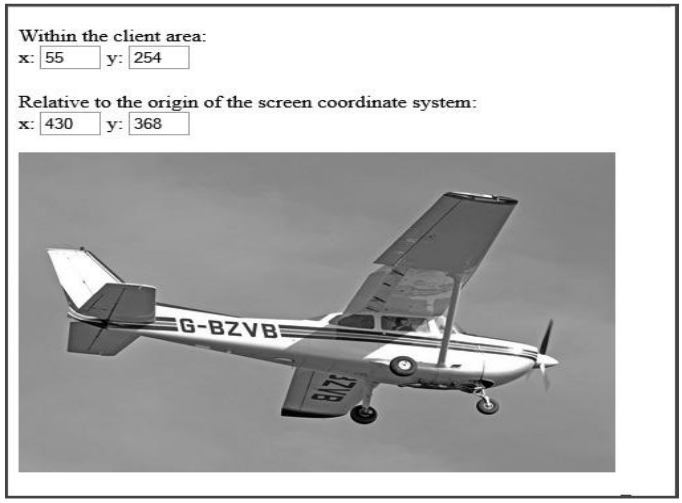
```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Where is the cursor?</title>
   <script type="text/javascript" src="where.js"></script>
   <style>
      body {
         display: flex;
```

```
            align-items: center;
            justify-content: center;
            text-align: center;
            height: 100vh;
            margin: 0;
        }
    </style>
</head>
<body onclick="findIt(event)">
    <form action="">
        <p>
            Within the client area: <br />
            x: <input type="text" id="xcoor1" size="4" />
            y: <input type="text" id="ycoor1" size="4" />
            <br /><br />
            Relative to the origin of the screen coordinate system: <br />
            x: <input type="text" id="xcoor2" size="4" />
            y: <input type="text" id="ycoor2" size="4" />
        </p>
    </form>
    <p>
        <img src="../images/plane1.jpg" height="300" alt="(Picture of an
airplane)" width="450" />
    </p>
</body>
</html>
```

JavaScript (where.js):

```
// where.js
// Show the coordinates of the mouse cursor position
// in an image and anywhere on the screen when the mouse
// is clicked
// The event handler function to get and display the
// coordinates of the cursor, both in an element and
// on the screen
function findIt(evt) {
    document.getElementById("xcoor1").value = evt.clientX;
    document.getElementById("ycoor1").value = evt.clientY;
    document.getElementById("xcoor2").value = evt.screenX;
    document.getElementById("ycoor2").value = evt.screenY;
}
```

Example Execution:



- Clicking anywhere on the page triggers the findIt function.
- The corresponding mouse cursor coordinates are displayed in the text boxes.

This example provides a simple yet effective way to interactively showcase mouse cursor coordinates on a webpage, demonstrating practical event handling in JavaScript.

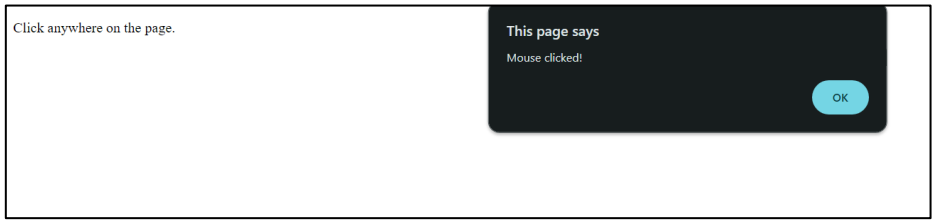
REACTING TO MOUSE CLICKS EXAMPLE

- ✓ The next example is another one related to reacting to mouse clicks. In this case, the mousedown and mouseup events are used, respectively, to show and hide the message "Please don't click here!" on the display under the mouse cursor when ever the mouse button is clicked, regardless of where the cursor is at the time.
- ✓ The offsets (-130 for left and -25 for top) modify the actual cursor position.
- ✓ A mouse click can be used to trigger an action.
- ✓ Use event handlers for onmousedown and onmouseup that change the visibility attributes of the message.

HTML (anywhere.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Sense events anywhere</title>
```

```
    <script type="text/javascript">
        function displayIt(evt) {
            alert("Mouse clicked!");
        }
    </script>
</head>
<body onmousedown="displayIt(event);">
    <p>Click anywhere on the page.</p>
</body>
</html>
```



SLOW MOVEMENT OF ELEMENTS

- ✓ Slow Movement of Element Using setTimeout and onload:
- ✓ To achieve a slow movement of an element in JavaScript, we can use the setTimeout method, which executes a function after a specified delay. This allows us to move the element gradually by making small adjustments over time.

HTML (moveText.html):

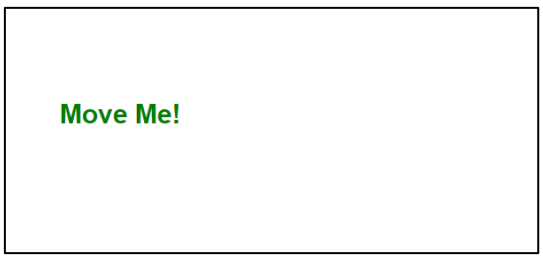
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Slow Element Movement</title>
    <script type="text/javascript" src="movement.js"></script>
    <style>
        #movingElement {
            position: absolute;
            left: 50px;
            top: 50px;
            font: bold 1.5em 'Arial';
            color: green;
        }
    </style>
```

```
</head>
<body onload="initMovement()">
   <p id="movingElement">Move Me!</p>
</body>
</html>
```

JavaScript (moveText.js):

```
var element, targetX = 200, targetY = 200;

function initMovement() {
   element = document.getElementById('movingElement').style;
   moveElement();
}

function moveElement() {
   var currentX = parseInt(element.left) || 50;
   var currentY = parseInt(element.top) || 50;

   if (currentX !== targetX) {
      currentX += (currentX < targetX) ? 1 : -1;
   }

   if (currentY !== targetY) {
      currentY += (currentY < targetY) ? 1 : -1;
   }

   if (currentX !== targetX || currentY !== targetY) {
      element.left = currentX + "px";
      element.top = currentY + "px";
      setTimeout(moveElement, 20);
   }
}
```

Output:



DRAGGING AND DROPPING ELEMENTS

Drag and Drop is an interactive feature that allows users to move elements by clicking, holding, and releasing the mouse button. In HTML5, Drag and Drop is simplified, and any element can be easily made draggable.

Events:

Several Drag and Drop events are crucial for understanding and implementing this feature:

1. **ondrag:** Element or text selection is being dragged.
2. **ondragstart:** Calls a function, usually drag(event), to specify what data to be dragged.
3. **ondragenter:** Determines whether the drop target accepts the drop. If accepted, this event needs to be canceled.
4. **ondragleave:** Occurs when the mouse leaves an element before a valid drop target while dragging.
5. **ondragover:** Specifies where the dragged data can be dropped.
6. **ondrop:** Specifies where the drop occurs at the end of the drag operation.
7. **ondragend:** Occurs when the user finishes dragging an element.

Procedure for Drag and Drop:

1. **Make an object draggable:**

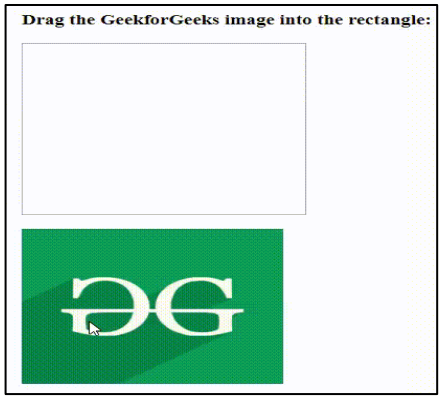
✓ Set the draggable attribute to true for the element: <img draggable="true">.
✓ Specify what happens when the element is dragged using ondragstart, which calls a function like drag(event).
✓ Use dataTransfer.setData() to set the data type and value of the dragged data.

2. **Dropping the Object:**

✓ Use ondragover to specify where the dragged data can be dropped. Prevent the default handling of the element.
✓ Implement the ondrop event, which allows the actual drop to occur.

Example 1: Image Drag & Drop:

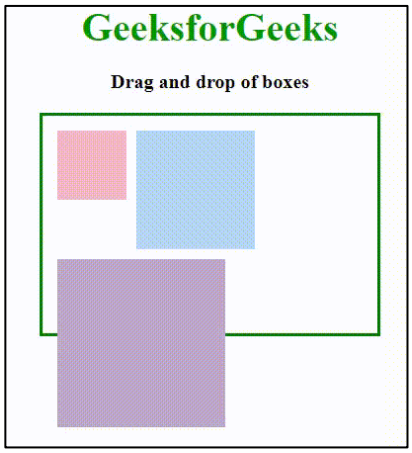
```
<!DOCTYPE HTML>
<html>
<head>
  <style>
    #getData {
      width: 250px;
      height: 200px;
```



```
            padding: 10px;
            border: 1px solid #4f4d4d;
        }
    </style>
    <script>
        function allowDrop(event) {
            event.preventDefault();
        }
        function drag(event) {
            event.dataTransfer.setData("text", event.target.id);
        }
        function drop(event) {
            event.preventDefault();
            var fetchData = event.dataTransfer.getData("text");
            event.target.appendChild(document.getElementById(fetchData));
        }
    </script>
</head>
<body>
    <h3>Drag the GeekforGeeks image into the rectangle:</h3>
    <div id="getData" ondrop="drop(event)"
ondragover="allowDrop(event)"></div>
    <br>
    <img id="dragData" src="gfg.png" draggable="true"
ondragstart="drag(event)" width="250" height="200">
</body>
</html>
```

Example 2: Box Drag & Drop:

```
<!DOCTYPE HTML>
<html>
<head>
    <title>Drag and Drop box</title>
    <script>
        function allowDrop(event) {
            event.preventDefault();
        }
        function dragStart(event) {
            event.dataTransfer.setData("text",
event.target.id);       }
```



```
        function dragDrop(event) {
            event.preventDefault();
            let data = event.dataTransfer.getData("text");
            event.target.appendChild(document.getElementById(data));
        }
    </script>
    <style>
        /* Styles for the boxes and text */
    </style>
</head>
<body>
    <div>
        <div id="box1" draggable="true"
ondragstart="dragStart(event)"></div>
        <div id="box2" draggable="true"
ondragstart="dragStart(event)"></div>
        <div id="box3" ondrop="dragDrop(event)"
ondragover="allowDrop(event)"></div>
    </div>
</body>
</html>
```

XML

INTRODUCTION TO XML

XML, or eXtensible Markup Language, is a powerful tool for organizing and sharing data. Let's break down its key points:

1**. What is XML?**

- ✓ XML is a meta-markup language, a tool for defining markup languages.
- ✓ Markup languages help structure and describe content in documents.

2. Origin and Standards:

- ✓ SGML (Standard Generalized Markup Language) was the precursor to XML.
- ✓ In 1986, SGML became an ISO standard.
- ✓ HTML (Hypertext Markup Language) was developed based on SGML for the web in 1990.
- ✓ In 1996, W3C started working on XML, a simplified version of SGML.

- ✓ The first XML standard (1.0) was published in 1998.

3. **XML vs. HTML:**

- ✓ HTML describes the layout of information on the web but lacks meaning.
- ✓ XML addresses this by allowing users to define their own markup languages.
- ✓ XML focuses on representing data meaningfully, making it versatile.

4**. XML Application:**

- ✓ XML is not a replacement for HTML; it's a meta-markup language.
- ✓ HTML can be defined as an XML markup language.
- ✓ XML is a universal data interchange language for storing textual data.

5. **XML Document:**

- ✓ Documents using XML-based markup language are called XML documents.
- ✓ XML documents can be written by hand or generated by programs.
- ✓ Universal readability makes XML ideal for data interchange.

6. **XML Processing:**

- ✓ XML processors analyze documents for applications before data access.
- ✓ Browsers need style sheets for displaying XML content.
- ✓ XML processors parse XML documents to extract elements, attributes, and data.

7. **Uses of XML:**

- ✓ CDF for scalar and multidimensional data.
- ✓ SVG for scalable vector graphics.
- ✓ MathML for mathematical notation.
- ✓ CML for chemical information.
- ✓ GPX for GPS data.
- ✓ MML for medical information.
- ✓ OOXML for Microsoft Office formats.
- ✓ W3C-developed tag sets for web applications.

In essence, XML provides a standardized, readable, and universal way to structure and exchange data. It's not just a markup language; it's a tool for creating tailored markup languages to meet diverse data needs.

SYNTAX OF XML

XML syntax has two levels: low-level (basic rules for all documents) and schemas (rules for specific documents). Let's focus on the basic syntax:

Low-Level Syntax:

- ✓ XML documents start with an XML declaration, specifying the version and encoding.
- ✓ Comments are like HTML and can't have consecutive dashes.
- ✓ XML names (for elements and attributes) start with a letter or underscore, case-sensitive, and can include digits, hyphens, and periods.

Document Structure:

- ✓ Every XML document has a single root element.
- ✓ Elements must be properly nested inside the root.
- ✓ XML tags, like HTML, use angle brackets.
- ✓ Elements with content must have closing tags; self-closing tags end with /.

Attributes:

- ✓ XML tags can have attributes specified with name-value assignments.
- ✓ Attribute values must be enclosed in single or double quotes.

Well-Formed XML:

- ✓ An XML document adhering to syntax rules is considered well-formed.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<ad>
  <year>1960</year>
  <make>Cessna</make>
  <model>Centurian</model>
  <color>Yellow with white trim</color>
  <location>
    <city>Gulfport</city>
    <state>Mississippi</state>
  </location>
</ad>
```

Design Choices:

- ✓ Choose between adding attributes or defining nested elements.
- ✓ Attributes are great for identifiers or data from a set of possibilities.
- ✓ Nested elements are preferable when dealing with substructure or growing complexity.
- ✓ Consider nested elements if data is subdata of the parent.

Example Choices:

Attribute Example:

```
<patient name="Maggie Dee Magpie">...</patient>
```

Nested Element Example:

```
<patient>
  <name>Maggie Dee Magpie</name>
  ...
</patient>
```

Nested Element with Substructure:

```
<patient>
  <name>
    <first>Maggie</first>
    <middle>Dee</middle>
    <last>Magpie</last>
  </name>
  ...
</patient>
```

Remember, simplicity and clarity guide your choice between attributes and nested elements in XML design.

XML DOCUMENT STRUCTURE

An XML document consists of entities, logically related information collections, ranging from characters to larger sections. Here's a simplified breakdown:

Document Entity:

- The main entity representing the entire document.
- Can include references to other named entities.

- Breaks down large documents for manageability.

Named Entities:

- Entities, except the document entity, must have a name.
- Entities can be used to avoid data repetition and manage binary data.
- Binary entities store non-textual data like images.

Entity Naming:

- Entity names can be any length.
- Must start with a letter, dash, or colon.
- Followed by letters, digits, periods, dashes, underscores, or colons.

Entity References:

- Represented by the entity name enclosed in ampersand and semicolon, like &entity_name;.
- Commonly used to handle characters that are XML markup delimiters.

Predefined Entities:

- XML includes predefined entities for common HTML characters.
- Example: < for < and > for >.

Character Data Section:

- Used to include special characters directly without entity references.
- Syntax: <![CDATA[content]]>.
- No parsing by the XML parser, making it useful for special characters.

Example Usage:

- <![CDATA[The last word of the line is >>> here <<<]]>

Instead of:

- The last word of the line is >>> here <<<.

Important Note:

- CDATA opening keyword is effectively [CDATA[.
- No spaces between [and C or A and the second [.

- Content inside CDATA is not parsed, and entity references are not expanded.

Entities help structure XML documents, and predefined entities handle common characters. CDATA sections allow including special characters directly without cluttering the content. Keep entity naming rules in mind for consistency.

DOCUMENT TYPE DEFINITION

A Document Type Definition (DTD) is a way to describe the structure and legal elements of an XML document. Let's break down DTD into simpler terms:

Definition:

- A set of rules that defines the structure of an XML document.
- Describes the elements and attributes allowed in the document.

Purpose:

- Ensures consistency and conformity in XML documents.
- Acts as a guide for creating valid and well-formed XML.

Key Components:

- Elements: Names of elements and their order in the document.
- Attributes: Define properties or additional information for elements.
- Entities: Used for defining reusable text or symbols.
- Notations: Specify formats of non-XML data (like images).

Declaration in XML Document:

- Placed at the beginning of an XML document.
- Starts with <!DOCTYPE root_element_name SYSTEM "document.dtd">.

Example DTD:

```
<!DOCTYPE bookstore [
  <!ELEMENT bookstore (book+)>
  <!ELEMENT book (title, author, price)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT author (#PCDATA)>
  <!ELEMENT price (#PCDATA)>]>
```

Explanation:

- bookstore must contain one or more book elements.
- Each book must have title, author, and price elements.
- title, author, and price elements contain parsed character data (#PCDATA).

Validation:

- DTD helps validate XML documents.
- Validates the document against the rules specified in the DTD.

Internal vs. External DTD:

- Internal DTD: DTD is included within the XML document.
- External DTD: DTD is stored in a separate file and referenced in the XML document.

Common Use Cases:

- Ensures consistent data structure in documents.
- Facilitates interoperability by defining a standard structure.

DTD Example (Internal):

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to, from, heading, body)>
  <!ELEMENT to (#PCDATA)>
  <!ELEMENT from (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
]>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

DTD is a set of rules defining the structure of an XML document. It specifies allowed elements, attributes, entities, and notations. DTD can be internal or external, and its purpose is to ensure consistency and facilitate validation of XML documents.

NAMESPACES

XML Namespaces are used to avoid naming conflicts in XML documents, especially when combining different markup vocabularies.

Definition:

- XML Namespaces prevent naming clashes in XML documents.
- They allow elements and attributes from different sources to coexist.

Declaration:

- Declare a namespace using xmlns attribute in the root element.
- Syntax: <element xmlns[:prefix]="namespaceURI">
- Example: <birds xmlns:bd="http://www.audubon.org/names/species">

Prefixing

- Use prefixes to distinguish elements from different namespaces.
- Example: <bd:lark> (bd is the prefix)

Default Namespace:

- If no prefix is used, it implies the default namespace for the document.
- Example: <birds xmlns="http://www.audubon.org/names/species">

Multiple Namespaces:

- Declare multiple namespaces in the root element.
- Example:
    - <birds xmlns:bd="http://www.audubon.org/names/species"
      xmlns:html="http://www.w3.org/1999/xhtml">

Namespace Usage:

- Helps when combining, for instance, bird species data (bd) and HTML elements.
- Example:
    - <bd:parrot>...</bd:parrot>
      <html:div>...</html:div>

Namespace Considerations:

- Namespaces use URIs (Uniform Resource Identifiers) to uniquely identify.
- Prefixes make long URIs more manageable.
- Attribute names are not included in namespaces.

Simplified Example:

```
<states xmlns="http://www.states-info.org/states"
        xmlns:cap="http://www.states-info.org/state-capitals">
  <state>
    <name>South Dakota</name>
    <population>754844</population>
    <capital>
      <cap:name>Pierre</cap:name>
      <cap:population>12429</cap:population>
    </capital>
  </state>
  <!-- More states -->
</states>
```

In essence, XML Namespaces are like last names for XML elements, preventing confusion when elements from different families (vocabularies) come together in one document.

XML SCHEMAS

XML Schemas are like blueprints for XML documents, providing rules and structures. Here's a simplified breakdown:

Purpose of Schemas:

- XML Schemas define elements, attributes, and structure rules for XML documents.
- They ensure consistency and standardization in XML data.

Comparison to Classes and Objects:

- Schemas are akin to class definitions in object-oriented programming.
- An XML document following a schema is like an instance of that schema.

Namespace Centric:

- XML Schemas, like XML, use namespaces represented by URIs.
- Unique URIs, often starting with the author's web address, prevent conflicts.

Schema Declaration:

- Schemas are written using elements from the http://www.w3.org/2001/XMLSchema namespace.
- The schema element is the root, specifying namespace and often a prefix.
- Example:
    - <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://cs.uccs.edu/planeSchema" xmlns="http://cs.uccs.edu/planeSchema" elementFormDefault="qualified">

Defining a Schema Instance:

- An instance must specify namespaces it uses in the root element.
- Example:
    - <planes xmlns="http://cs.uccs.edu/planeSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://cs.uccs.edu/planeSchema planes.xsd">

Validation:

- XML documents are validated against XML Schemas for correctness.
- Validation programs, like xsv, ensure conformance to schema rules.

Schema Fundamentals:

- Schemas serve three primary purposes: element/attribute specification, structure definition, and data type specification.
- XML Schemas are namespace-centric, using URIs for uniqueness.

Schema Instance Details:

- The schemaLocation attribute in an instance points to the schema file.
- Namespaces like xsi and xsd are used for schema-related declarations.

Example:

- A snippet of schema declaration:
    - <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://cs.uccs.edu/planeSchema" xmlns="http://cs.uccs.edu/planeSchema" elementFormDefault="qualified">
- A snippet of instance declaration:

- <planes xmlns="http://cs.uccs.edu/planeSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://cs.uccs.edu/planeSchema planes.xsd">

In essence, XML Schemas act as guardians, ensuring XML documents follow predefined rules and maintain order and consistency.

DISPLAYING RAW XML DOCUMENTS

When you open an XML document in a browser without a designated style sheet, what you see is the raw markup. Let's break down this process into comprehensible points:

1. Formatting Challenges:

- XML-enabled browsers don't come equipped with predefined styles for document elements.
- Default styles kick in when there's no specific style sheet present.

2**. Default Styles in Browsers:**

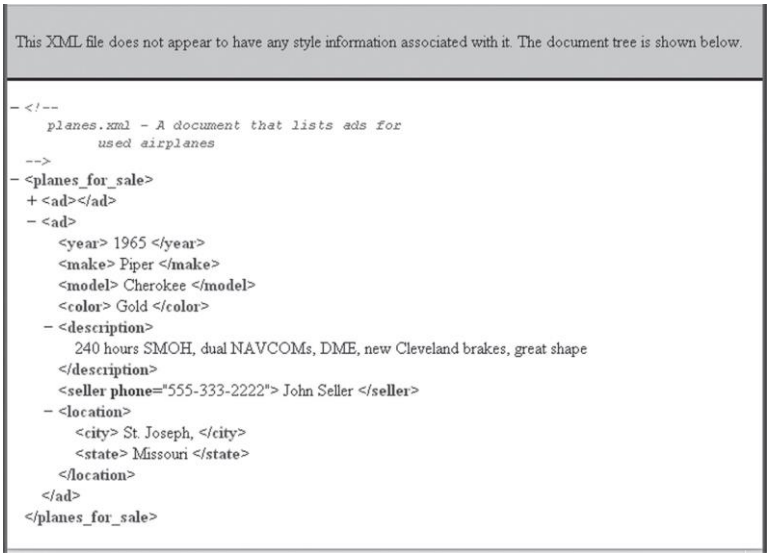
- Modern browsers have default styles in place for displaying XML content.
- The result is a somewhat stylized listing of XML markup.

3. Example Display (Firefox 3):

- Browser display in its raw form without a specified style sheet.
- An example would be the portrayal of Planes.xml shown in Figure

4. Handling Elided Elements:

- Some elements might be elided, meaning they are temporarily suppressed in the display.
- Typically, you can elide elements by clicking on a dash that precedes the respective element.



5. Usage and Restrictions:

- Viewing raw XML isn't a common practice and is mainly reserved for document development stages.
- Internet Explorer (prior to IE9) might have restrictions on eliding; Chrome, on the other hand, lacks a default style sheet.

6**. Purpose of Raw Display:**

- The raw XML display is a tool used by developers to review and check the structure of a document during the development phase.
- It's not intended for regular consumption by end-users.

7. Browser-Specific Behaviour:

- Internet Explorer may require allowing blocked content to elide elements.
- In Chrome, you can view the XML by selecting Tools/View Source.

In essence, the raw display of XML serves as a developer's tool, offering an in-depth look at the document's underlying structure and content during the developmental stages. It's a practical way to inspect the XML markup before finalizing the document for end-user consumption.

DISPLAYING XML DOCUMENTS WITH CSS

When presenting an XML document, style-sheet information can be added to enhance its appearance. Cascading Style Sheets (CSS) provide a way to define the style for XML elements, making the document more visually appealing. Below is a simplified explanation of how to achieve this:

1. Create a CSS File:

Develop a Cascading Style Sheet (CSS) file with style information for XML elements. For instance, in the example below (planes.css), styles are defined for elements like ad, year, make, model, etc.

```
ad { display: block; margin-top: 15px; color: blue;}
year, make, model { color: red; font-size: 16pt;}
color {display: block; margin-left: 20px; font-size: 12pt;}
description {display: block; margin-left: 20px; font-size: 12pt;}
seller { display: block; margin-left: 15px; font-size: 14pt;}
location {display: block; margin-left: 40px; }
city {font-size: 12pt;}
state {font-size: 12pt;}
```

2. Link XML to CSS:

Connect the XML document to the CSS style sheet using the xml-stylesheet processing instruction. Include this line at the beginning of your XML document.

```
<?xml-stylesheet type="text/css" href="planes.css"?>
```

This line specifies the type of style sheet (text/css) and the file name (planes.css).

3. Display the Formatted XML:

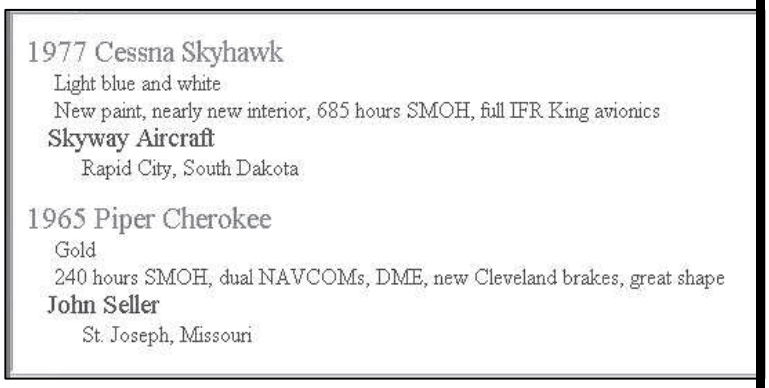
Once linked, the XML document, when opened in a browser, will be styled according to the rules defined in the CSS file.

Example:

Assuming the XML file is named planes.xml, and it contains information about planes. Link it with the planes.css style sheet.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="planes.css"?>
```

```
<planes>
  <!-- XML content... -->
</planes>
```



planes.css:

```
/* CSS styles for elements in
planes.xml */
ad { /* styles for ad element */ }
year, make, model { /* styles for year,
make, model elements */ }
color { /* styles for color element */ }
description { /* styles for description
element */ }
seller { /* styles for seller element */ }
location { /* styles for location element */ }
city { /* styles for city element */ }
state { /* styles for state element */ }
```

By following these steps, you can utilize CSS to format and enhance the display of XML documents, making them more visually appealing and easier to understand. ***Note: Refer Lab Program 5***

XSLT

XSLT (Extensible Stylesheet Language Transformations) is a powerful technology developed by the W3C for transforming and styling XML documents. It allows you to define rules for how XML data should be presented or transformed into different structures, such as HTML or other XML formats. Here's a simplified guide to understanding XSLT style sheets:

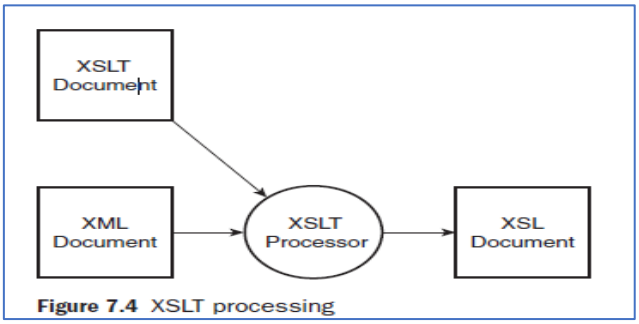


Figure 7.4 XSLT processing

1. Basics of XSLT:

XSLT uses XML syntax to define transformations. An XSLT style sheet consists of a set of rules that match elements in the source XML document and define how they should be transformed in the output.

2. Structure of an XSLT Style Sheet:

An XSLT style sheet typically has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <!-- XSLT rules go here -->
</xsl:stylesheet>
```

xmlns:xsl: Specifies the XSL namespace.
version: Specifies the XSLT version.

3. Matching and Transformation:

Rules in XSLT are defined using patterns to match elements in the source XML document. Once matched, you can specify how to transform or present those elements in the output.

Example:

```
<xsl:template match="book">
  <!-- Transformation rules for book element -->
</xsl:template>
```

4. Output Specification:

You can define the structure and format of the output document using elements like <xsl:element> and <xsl:attribute>.

Example:

```
<xsl:template match="book">
  <div>
    <h2><xsl:value-of select="title"/></h2>
    <!-- Other content -->
  </div>
</xsl:template>
```

5. Applying Templates:

Use <xsl:apply-templates> to apply a template to a specific element or set of elements.

Example:

```
<xsl:template match="library">
```

```
    <html>
      <body>
        <xsl:apply-templates select="book"/>
      </body>
    </html>
  </xsl:template>
```

6. Example:

Consider a simplified example where you want to transform a set of books into an HTML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="library">
    <html>
      <body>
        <h1>Book List</h1>
        <xsl:apply-templates select="book"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="book">
    <div>
      <h2><xsl:value-of select="title"/></h2>
      <p>Author: <xsl:value-of select="author"/></p>
    </div>
  </xsl:template>
</xsl:stylesheet>
```

7. Applying XSLT:

To apply an XSLT style sheet to an XML document, you can use various tools, APIs, or browsers that support XSLT transformations.

8. Output:

The result of applying the XSLT style sheet to your XML document would be an HTML document with the transformed content.

XSLT provides a flexible and powerful way to transform and stylize XML data, making it suitable for various scenarios such as converting XML to HTML for web display or generating different XML structures.

WEB SERVICES

Web services allow different software applications, written in various languages and residing on different platforms, to connect and work together. This concept started gaining traction with Bill Gates introducing BizTalk in 1999, later renamed .NET.

Basic Idea:

- Originally, the web was about sharing information. Web services take this further by enabling services rather than just sharing documents.
- Think of it as a way for software components to communicate and collaborate across the internet.

Evolution from Web-to-Web Services:

- The web started with providing information through documents using methods like GET and POST.
- Web services deploy services rather than documents. Software components are hosted on a web server and accessed remotely.

Comparison with RPC (Remote Procedure Call):

- RPC was an earlier concept for distributed components, but technologies like DCOM and CORBA were complex and not universally interoperable.
- Web services aim to provide a simple and convenient way for components of different systems to interact seamlessly.

Web Services Dream:

- The dream is to have protocols allowing all components to interoperate seamlessly without human intervention.
- Web services use non-proprietary protocols and languages for universal component interoperability.

Roles in Web Services:

- Service Providers: Develop and deploy software services. Described using Web Services Definition Language (WSDL).
- Service Requestors (Consumers): Users or applications seeking services.
- Service Registry: Created using Universal Description, Discovery, and Integration Service (UDDI). Helps discover available services.

Key Standards:

- WSDL (Web Services Definition Language): Describes operations, messages, and protocols for a web service.
- UDDI (Universal Description, Discovery, and Integration): Protocol to query a registry for available services.

Communication in Web Services:

- SOAP (Simple Object Access Protocol): Defines message and RPC formats for web services. Originally an acronym for Standard Object Access Protocol.

Web Services Tools:

- Developed using tools like Microsoft's Visual Studio (VS) and Oracle's NetBeans.

Web Services Consumers:

- Clients of the service, which could be web applications, non-web applications, or other web services.
- Clients interact with a local proxy, a substitute for the remote web service, making it seem like they are calling the remote service directly.

In essence, web services simplify collaboration between software components over the internet, enabling a new level of interoperability.

UNIT-5

PHP, ANGULAR JS AND JQUERY

INTRODUCTION TO PHP

PHP, initially created by Rasmus Lerdorf in 1994, started as a tool for tracking visitors to his personal website. In 1995, he released the first public version called Personal Home Page Tools. Originally standing for Personal Home Page, PHP's name evolved to PHP: Hypertext Preprocessor.

Within two years, PHP gained popularity on many websites. As its development outgrew a single person, a group of volunteers took over. PHP is now open-source, with a processor on most web servers.

PHP, a server-side scripting language, is commonly used for form handling and database access. It supports 15 database systems and email protocols like POP3 and IMAP. Additionally, it works with distributed object architectures like COM and CORBA.

OVERVIEW OF PHP

PHP is a server-side scripting language embedded in HTML, offering an alternative to Java Server Pages and Active Server Pages. When a browser encounters JavaScript in HTML, it uses a JavaScript interpreter. In contrast, if it finds PHP script, the web server's PHP processor is called, identified by file extensions like .php.

The PHP processor has two modes: copy mode and interpret mode. It reads a PHP document, copies HTML code, and interprets PHP script, producing an HTML output file. This file is sent to the browser, hiding the PHP script. PHP, like JavaScript, is usually interpreted but may have some precompilation for speed.

PHP shares syntax and semantics with JavaScript, making it easy to learn for JavaScript users. It uses dynamic typing, meaning variable types are set when assigned values. PHP is forgiving and supports both procedural and object-oriented programming.

With a wide range of functions and extensive library support, PHP is versatile for server-side development. Many functions provide interfaces to systems like mail and databases, making PHP a powerful tool.

GENERAL SYNTACTIC CHARACTERISTICS

PHP scripts are either embedded in markup documents or stored in separate files. In documents, PHP code is enclosed between <?php and ?> tags. To include a script from another file, the include construct is used, like include("table2.inc").

Variable names in PHP start with a dollar sign ($), followed by letters, digits, or underscores. They are case-sensitive. Reserved words in PHP, like if, while, and break, are not case-sensitive.

and	else	global	require	virtual
break	elseif	if	return	xor
case	extends	include	static	while
class	false	list	switch	
continue	for	new	this	
default	foreach	not	true	
do	function	or	var	

PHP supports three ways to write comments: #, // for single-line, and /* */ for multiple lines. Statements end with semicolons, and braces are used for compound statements in control structures.

PHP integrates with markup using tags, includes external scripts, uses $ for variables, and supports various comment styles. Statements end with semicolons, and braces organize compound statements in control structures.

PRIMITIVES, OPERATIONS, AND EXPRESSIONS

In PHP, there are four scalar types: Boolean, integer, double, and string. Additionally, there are compound types like arrays and objects, as well as special types like resource and NULL. For now, let's focus on the scalar types and NULL.

Scalar Types:

1. **Boolean:** Represents true or false values.
2. **Integer:** Represents whole numbers.
3. **Double:** Represents floating-point numbers (like decimals).
4. **String:** Represents sequences of characters.

NULL:

- The default value for an unassigned variable in PHP is NULL.

Variable Types in PHP:

- PHP is dynamically typed, meaning you don't need to declare the type of a variable explicitly.
- The type of a variable is determined when it's assigned a value.
- Variable Usage:
- Variables can be tested to see if they have a value using isset($variable).
- Unset a variable or reset it to an unassigned state using unset($variable).
- Type Coercion:
- PHP can coerce NULL into different values based on the context.

For example, if NULL is used in a numeric context, it becomes 0; in a string context, it becomes an empty string.

Data Type Operations:

- PHP has specific functions and operators for each data type.
- For example, there are arithmetic operators for numeric types (+, -, *, /, %), string concatenation (.), and more.
- String variables can be treated like arrays for accessing individual characters.

Type Conversions:

- PHP performs both implicit and explicit type conversions.
- Context often determines coercions between different types.
- Explicit type conversions can be done using casts (int)$variable, or functions like intval(), doubleval(), or strval().
- Functions like gettype() and type-testing functions (is_int(), is_float(), etc.) help determine variable types.

Assignment Operators:

- PHP has various assignment operators, like += or /=, for shorthand operations.

Let's go through some examples to illustrate the concepts:

Scalar Types and NULL:

```
<?php
// Boolean
$isTrue = true;
$isFalse = false;

// Integer
$number = 42;

// Double
$floatNumber = 3.14;

// String
$text = "Hello, PHP!";

// Unassigned variable is NULL
$unassignedVariable = NULL;

// Testing if a variable is set
if (isset($number)) {
    echo "$number is set.\n";
} else {
    echo "$number is not set.\n";
}

// Unsetting a variable
unset($number);
// Now $number is NULL
// Type Coercion
$numericContext = $unassignedVariable + 10; // NULL coerced to 0
$stringContext = "Value: " . $unassignedVariable; // NULL coerced to an
empty string
echo $numericContext . "\n"; // Outputs: 10
echo $stringContext . "\n";  // Outputs: Value:
?>
```

Variable Types and Type Conversions:

```
<?php
// Implicit Type Conversion
$integerValue = 10;
$doubleValue = $integerValue / 2; // Integer is coerced to double
echo $doubleValue . "\n"; // Outputs: 5

// Explicit Type Conversion
$doubleValueExplicit = (double)$integerValue;
echo $doubleValueExplicit . "\n"; // Outputs: 10.0

// Using functions for explicit conversion
$stringValue = "123";
$convertedInteger = intval($stringValue);
echo $convertedInteger . "\n"; // Outputs: 123

// Get Type of a Variable
$typeOfVariable = gettype($stringValue);
echo "Type of \$stringValue is: $typeOfVariable\n"; // Outputs: Type of $stringValue is: string

// Type Testing Functions
$isInteger = is_int($convertedInteger);
echo $isInteger ? "It's an integer.\n" : "It's not an integer.\n"; // Outputs: It's an integer.
?>
```

These examples cover the basics of scalar types, NULL, variable handling, type coercion, and type conversions in PHP. Feel free to run these examples in a PHP environment to see the output.

OUTPUT

- All output in PHP is part of HTML or XHTML, including embedded client-side scripts.
- The print function is used for simple, unformatted output.
- It can be called with or without parentheses around its parameter.

Example:

```
<?php
// Using print for simple output
print "Some apples are red <br /> No kumquats are <br />";

// Coercion example: print will convert non-string types to string
print(47); // Outputs: 47

// Interpolating variables in double-quoted strings
$result = 100;
print "The result is: $result <br />";

// Using printf for formatted output
$day = "Tuesday";
$high = 79;
printf("The high on %7s was %3d", $day, $high);
?>
```

Example Output:

```
Some apples are red <br /> No kumquats are <br />
47
The result is: 100 <br />
The high on Tuesday was  79
```

A Simple PHP Document:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>today.php</title>
   <meta charset="utf-8" />
</head>
<body>
   <p>
```

```
<?php
// Generating date information using the date function
print "<b>Welcome to my home page <br /> <br />";
print "Today is:</b> ";
print date("l, F jS");
print "<br />";
?>
</p>
</body>
</html>
```

This example displays a welcome message and the current day, month, and day of the month using the date function. The date format is specified in the first parameter of the date function.

CONTROL STATEMENTS

- PHP control statements (if, switch, while, for, do-while) are similar to those in C and its descendants.
- Control expressions can be of any type, but they are coerced to Boolean.
- PHP has relational operators (>, <, >=, <=, !=, ==, ===, !==) for comparisons.
- Boolean operators (and, or, xor, !, &&, ||) are available.
- Selection statements (if, switch) are used for conditional execution.

Examples:

Example 1 - If Statement:

```
<?php
$num = 5;
if ($num > 0)
   $pos_count++;
elseif ($num < 0)
   $neg_count++;
else {
```

```
    $zero_count++;
    print "Another zero! <b />";
}
?>
```

Example 2 - Switch Statement:

```
<?php
$bordersize = "4";
switch ($bordersize) {
    case "0": print "<table>"; break;
    case "1": print "<table border='1'>"; break;
    case "4": print "<table border='4'>"; break;
    case "8": print "<table border='8'>"; break;
    default: print "Error-invalid value: $bordersize <br />";
}
?>
```

Example 3 - Loop Statements:

```
<?php
// While loop
$count = 1;
$sum = 0;
while ($count <= 100) {
    $sum += $count;
    $count++;
}

// Do-while loop
$count = 1;
$sum = 0;
do {
    $sum += $count;
    $count++;
} while ($count <= 100);

// For loop
for ($count = 1, $fact = 1; $count < $n;) {
    $count++;
    $fact *= $count;
}
?>
```

Example Document - HTML-PHP Integration:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>powers.php</title>
   <meta charset="utf-8" />
   <style type="text/css">
      td, th, table {border: thin solid black;}
      tr {text-align: center;}
   </style>
</head>
<body>
   <table>
      <caption>Powers table</caption>
      <tr>
         <th>Number</th>
         <th>Square Root</th>
         <th>Square</th>
         <th>Cube</th>
         <th>Quad</th>
      </tr>
      <?php
      for ($number = 1; $number <= 10;
$number++) {
         $root = sqrt($number);
         $square = pow($number, 2);
         $cube = pow($number, 3);
         $quad = pow($number, 4);
         print("<tr> <td> $number </td>");
         print("<td> $root </td> <td> $square </td>");
         print("<td> $cube </td> <td> $quad </td> </tr>");
      }
      ?>
   </table>
</body>
</html>
```

Powers table

Number	Square Root	Square	Cube	Quad
1	1	1	1	1
2	1.4142135623731	4	8	16
3	1.7320508075689	9	27	81
4	2	16	64	256
5	2.2360679774998	25	125	625
6	2.4494897427832	36	216	1296
7	2.6457513110646	49	343	2401
8	2.8284271247462	64	512	4096
9	3	81	729	6561
10	3.1622776601684	100	1000	10000

This example generates an HTML table with mathematical calculations using loops and functions.

ARRAYS

- **Unique Structure**: PHP arrays combine features of typical arrays and associative arrays (hashes).
- **Elements:** Each array element has a key and a value.
- **Key Types:** Keys can be nonnegative integers or strings.
- **Creation:**
    - ***Assignment:*** Creating an array by assigning values to specific keys.

```
$list[0] = 17;
$list[1] = "Today is my birthday!";
```

    - ***Array Construct:*** Creating an array using the array() construct.

```
$list = array(17, 24, 45, 91);
$ages = array("Joe" => 42, "Mary" => 41, "Bif" => 17);
```

Accessing Array Elements:

- ***Subscripting:*** Accessing individual elements using keys (subscripting).

```
$ages['Mary'] = 29;
```

- ***List Construct:*** Assigning multiple elements to scalar variables.

```
$trees = array("oak", "pine", "binary");
list($hardwood, $softwood, $data_structure) = $trees;
```

Functions for Dealing with Arrays:

- ***unset:*** Deleting array elements.

```
unset($list[2]);
```

- ***array_keys, array_values:*** Extracting keys and values from an array.

```
$days = array_keys($highs);
$temps = array_values($highs);
```

- ***array_key_exists:*** Checking if a key exists in an array.

```
if (array_key_exists("Tue", $highs)) {
$tues_high = $highs["Tue"];
}
```

- ***is_array:*** Checking if a variable is an array.

```
if (is_array($variable)) {
// It's an array.
}
```

- ***sizeof:*** Getting the number of elements in an array.

```
$len = sizeof($list);
```

- ***implode, explode:*** Converting between strings and arrays.

```
$str = implode(" ", $words);
$words = explode(" ", $str);
```

Sequential Access and Functions:

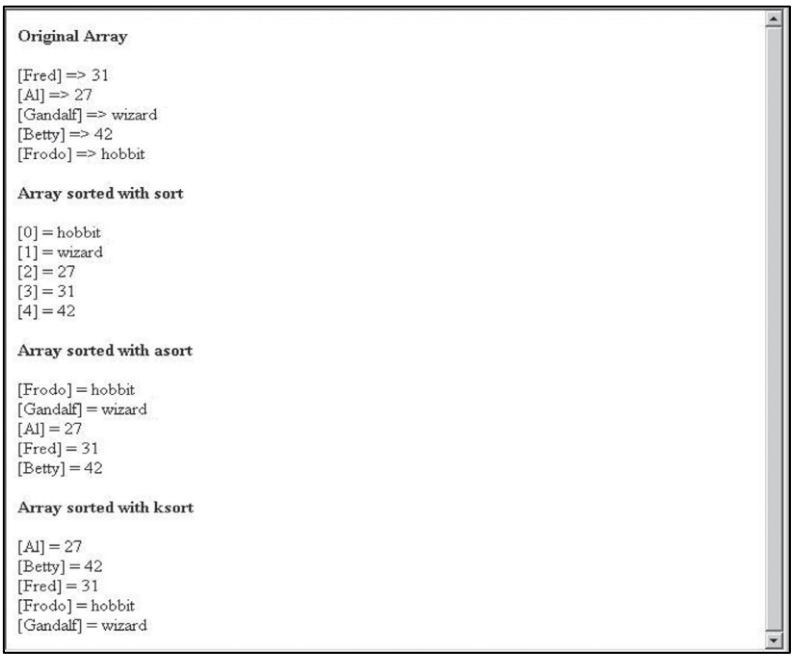
- Internal Pointer: Arrays have an internal pointer for sequential access.
- foreach: A loop to process all elements of an array.

```
foreach ($list as $temp)
   print("$temp <br />");
```

Sorting Arrays:

- sort, asort, ksort: Sorting arrays by values or keys.

```
sort($new);
asort($new);
ksort($new);
```



FUNCTIONS

PHP supports user-defined functions that are typical of C-based programming languages.

```
function name([parameters]) {
   // function body
}
```

Parameters are optional, and function definitions can appear anywhere in the document.

Return Statement:

- Used to specify the value to be returned to the caller.
- Function execution ends when a return statement is encountered or the last statement in the function has been executed.

Include Function:

- Used to include function definitions from a separate file into a document.
- Helpful for reusing functions across multiple documents.

Parameters:
Actual vs. Formal Parameters:

- Actual parameters are used in the function call.
- Formal parameters are listed in the function definition.
- Number mismatch is allowed, supporting functions with a variable number of parameters.

Passing by Value:

- Default parameter-passing mechanism.
- Values of actual parameters are copied into memory locations associated with formal parameters.
- No changes to actual parameters.

Passing by Reference:

- Achieved by using an ampersand (&) with formal parameters or actual parameters.
- Allows two-way communication, modifying actual parameters within the function.

Scope of Variables:
Local Variables:

- Defined in a function and have local scope.
- Hidden from non-local variables with the same name.
- Changes to local variables don't affect non-local variables.

Global Declaration:

- Used to access non-local variables within a function.
- Global variables have the same meaning inside and outside the function.

Lifetime of Variables:
Static Local Variables:

- Specified using the static keyword in a function.
- Retains information across multiple function activations.
- Lifetime begins at the first use in the first execution of the function and ends when the script execution ends.

PATTERN MATCHING

In PHP, there are two types of string pattern matching using regular expressions: POSIX regular expressions and Perl-Compatible Regular Expressions (PCRE). The preg_match function is used for pattern matching with PCRE. Below is a brief overview and an example:

Using preg_match for Pattern Matching

The preg_match function takes two parameters:

1. **Pattern:** The Perl-style regular expression as a string.
2. **Subject:** The string to be searched.

Example:

```
if (preg_match("/^PHP/", $str)) {
    print "\$str begins with PHP <br />";
} else {
    print "\$str does not begin with PHP <br />";
}
```

This example checks if the string $str begins with "PHP" using a regular expression.

Using preg_split for Splitting Strings

The preg_split function is used to split strings based on a pattern. It returns an array of substrings.

Example:

```
$fruit_string = "apple : orange : banana";
$fruits = preg_split("/ : /", $fruit_string);
```

Here, the string $fruit_string is split into an array named $fruits using the pattern " : ".

Example: Word Frequency Table

The following example demonstrates the use of preg_split to parse words and create a frequency-of-occurrence table:

```
<?php
// Function splitter
function splitter($str) {
    $freq = array(); // Empty word frequency array
    $words = preg_split("/[ \.,;:!\?]\s*/", $str); // Split the string into words
    foreach ($words as $word) {
```

```
            $keys = array_keys($freq);
            if (in_array($word, $keys)) {
                $freq[$word]++;
            } else {
                $freq[$word] = 1;
            }
        }
        return $freq;
    }

    // Main test driver
    $str = "apples are good for you, or don't you like apples? or maybe you like
    oranges better than apples";
    $tbl = splitter($str);

    // Display the words and their frequencies
    print "<br /> Word Frequency <br /><br />";
    $sorted_keys = array_keys($tbl);
    sort($sorted_keys);
    foreach ($sorted_keys as $word) {
        print "$word $tbl[$word] <br />";
    }
    ?>
```

Output:

```
Word Frequency

apples 3
are 1
better 1
don't 1
for 1
good 1
like 2
maybe 1
or 2
oranges 1
than 1
you 3
```

This script splits the input string into words, counts their frequencies, and displays a word frequency table. The output shows each word and its occurrence frequency.

FORM HANDLING

- To get form data, we need to use PHP superglobals $_GET and $_POST.
- The form request may be get or post. To retrieve data from get request, we need to use $_GET, for post request $_POST.

Certainly! Below is an extended explanation and code example for PHP form handling using both GET and POST methods. The example includes HTML form files (form1.html) and corresponding PHP handling files (welcome.php and login.php).

HTML Form for GET Request (form1.html):

```
<!-- File: form1.html -->
<form action="welcome.php" method="get">
    Name: <input type="text" name="name"/>
    <input type="submit" value="Visit"/>
</form>
PHP Script to Handle GET Request (welcome.php):
php
Copy code
<!-- File: welcome.php -->
<?php
    $name = $_GET["name"]; // Receive name field value in $name
variable
    echo "Welcome, $name";
?>
```

HTML Form for POST Request (form1.html):

```
<!-- File: form1.html -->
<form action="login.php" method="post">
    <table>
        <tr><td>Name:</td><td> <input type="text"
name="name"/></td></tr>
        <tr><td>Password:</td><td> <input type="password"
name="password"/></td></tr>
        <tr><td colspan="2"><input type="submit" value="Login"/>
</td></tr>
    </table>  </form>
```

PHP Script to Handle POST Request (login.php):

```
<!-- File: login.php -->
<?php
   $name = $_POST["name"]; // Receive name field value in $name variable
   $password = $_POST["password"]; // Receive password field value in $password variable

   echo "Welcome, $name, your password is: $password";
?>
```

Explanation:

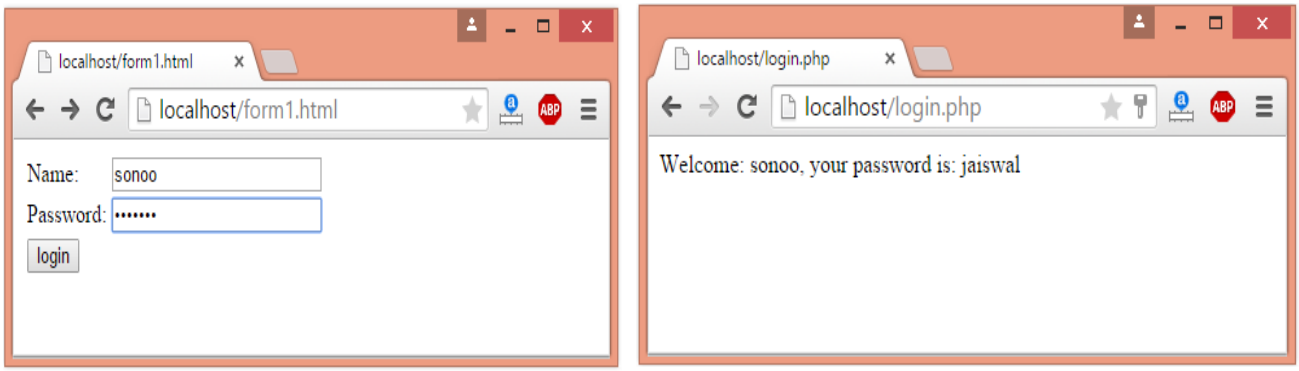
GET Request Example:

- A simple HTML form is created with a text input for the name.
- The form's action is set to "welcome.php" with the method "get."
- The PHP script (welcome.php) retrieves the name from the URL using $_GET and displays a welcome message.

POST Request Example:

- Another HTML form is created with text inputs for name and password.
- The form's action is set to "login.php" with the method "post."
- The PHP script (login.php) retrieves the name and password using $_POST and displays a welcome message.

Output:



COOKIES

A session in web development refers to the duration during which a user's browser interacts with a specific server. Sessions start when a browser connects to the server and end either when the browser is closed or due to server-defined inactivity. Cookies, small pieces of information with names and values, play a crucial role in managing sessions.

Common Uses of Cookies:

- **Shopping Carts:** Identify each user's shopping cart using session identifiers stored in cookies.
- **Personalization:** Create visitor profiles to personalize user experiences based on past interactions.
- **Targeted Advertising:** Store information about user preferences to deliver targeted advertisements.

Cookie Lifetime:

- Cookies are assigned a lifetime, and when this duration expires, the cookie is deleted from the browser.
- Every HTTP communication between a browser and a server includes a header containing cookies.
- Cookies are exchanged exclusively between a specific browser and a specific server.

Privacy Concerns:

- Cookies record browser activities, raising privacy concerns.
- Browsers allow users to reject cookies by changing settings, limiting the effectiveness of cookies.

Deletion of Cookies:

- Users can delete cookies, although the process varies by browser.

PHP Support for Cookies

setcookie Function:

- PHP provides the setcookie function to create and manage cookies.
- **Parameters:**
  - **Name (mandatory):** Cookie name as a string.
  - **Value (optional):** New value for the cookie as a string. If absent, the cookie is undefined.
  - **Expiration Time (optional):** Lifetime of the cookie in seconds. Default is zero, expiring at the end of the session.

- **Example:**

```
setcookie("voted", "true", time() + 86400);
```

This creates a cookie named "voted" with the value "true" and a lifetime of one day.

Cookie Retrieval in PHP:

- Cookies and their values arriving with a request are stored in the $_COOKIES array.
- Check if a cookie exists using the IsSet predicate function on the associated variable.

Important Consideration:

- Cookie creation or modification must occur before any HTML is generated in the PHP document, as cookies are stored in the HTTP header.

Cookies in PHP are essential for managing sessions and storing information between server and browser interactions. While widely used, it's crucial to consider user privacy and browser settings that may reject or limit the effectiveness of cookies.

SESSION TRACKING

Session tracking is a process used to manage information about a user's session, particularly during the session's duration. Unlike cookies, which are stored on the client-side, session tracking often involves the use of a session array stored on the server. A unique session identifier is commonly used for tasks such as shopping cart applications.

Key Points:

1. **Purpose of Session Tracking:**
   - To manage information about a client during a session.
   - Often involves storing a unique session ID in a session array.
2. **Session ID in PHP:**
   - In PHP, a session ID is an internal value that identifies a session.
   - PHP scripts become aware of session tracking by calling the session_start function, which creates or retrieves a session ID.
   - The session ID is created and recorded on the first call to session_start in a session.

3. **Session Arrays:**
    - Session arrays, stored on the server, hold information about previous client requests during a session.
    - Variables and their values registered in previously executed scripts are stored in the $_SESSION array.
    - Values can be created, changed, or destroyed using assignments and the unset operator.
4. **Example:**

```
session_start();

// Check if page_number is set, if not, initialize it to 1
if (!isset($_SESSION["page_number"]))
$_SESSION["page_number"] = 1;

$page_num = $_SESSION["page_number"];
echo "You have now visited $page_num page(s) <br />";

// Increment page_number
$_SESSION["page_number"]++;
```

    - This example demonstrates session tracking for the number of pages visited.
    - If page_number is not set, it initializes it to 1; otherwise, it displays the current count and increments it.
5. **Session ID Handling:**
    - PHP manages session IDs internally, and scripts need not handle them explicitly.
    - Subsequent calls to session_start in the same session retrieve the existing $_SESSION array.

Session tracking in PHP allows the management of user-specific information during a session. The use of a server-side session array, such as $_SESSION, provides a convenient way to store and retrieve data between requests within the same session.

INTRODUCTION TO JQuery

jQuery is a fast, small, and feature-rich JavaScript library designed to simplify the client-side scripting of HTML. It was created by John Resig and first released in 2006. jQuery is free, open-source software and is widely used in web development to enhance the functionality and interactivity of websites.

Here are some key aspects of jQuery:

1. **Cross-Browser Compatibility:** jQuery abstracts away many of the differences between browsers, providing a consistent and reliable platform for web development. It helps developers write code that works seamlessly across various browsers.
2. **DOM Manipulation:** One of jQuery's main strengths is its ability to simplify Document Object Model (DOM) manipulation. It provides a concise syntax for selecting and manipulating HTML elements on the page. This makes it easier to create dynamic and interactive user interfaces.

```
// Example: Change the text content of an element with the ID "myElement"
$("#myElement").text("Hello, jQuery!");
```

3. **Event Handling:** jQuery simplifies the process of handling events such as clicks, keypresses, and mouse movements. Event handling becomes more straightforward and consistent across different browsers.

```
// Example: Handle a button click event
$("#myButton").click(function() {
alert("Button clicked!");
});
```

4. **Ajax (Asynchronous JavaScript and XML):** jQuery simplifies the implementation of asynchronous requests to the server. This allows developers to load or send data without requiring a page refresh, creating a more responsive user experience.

```
// Example: Make an Ajax request to fetch data from a server
$.ajax({
url: "https://api.example.com/data",
method: "GET",
success: function(response) {
console.log("Data received:", response);
},
error: function(error) {
console.error("Error fetching data:", error);
}});
```

5. **Animations and Effects:** jQuery provides methods for creating animations and applying various effects to elements on the page. This includes features like fading in/out, sliding up/down, and custom animations.

```
// Example: Fade in an element with the class "myElement"
$(".myElement").fadeIn();
```

6. **Plugins:** jQuery has a rich ecosystem of plugins that extend its functionality. These plugins cover a wide range of features, from sliders and lightboxes to complex UI components, making it easy to enhance a website's capabilities.

To use jQuery, you need to include the jQuery library in your HTML document. You can either download the library and host it on your server or include it directly from a content delivery network (CDN). For example:

```
<!-- Include jQuery from a CDN -->
<script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>
```

Once included, you can start writing jQuery code in your JavaScript files or directly in your HTML documents using script tags.

jQuery syntax

The jQuery syntax is tailor-made for **selecting** HTML elements and performing some **action** on the element(s).

Basic syntax is: **$(*selector*).*action*()**

- A $ sign to define/access jQuery
- A (*selector*) to "query (or find)" HTML elements
- A jQuery *action*() to be performed on the element(s)

Examples:

- $(this).hide() - hides the current element.
- $("p").hide() - hides all <p> elements.
- $(".test").hide() - hides all elements with class="test".
- $("#test").hide() - hides the element with id="test".

1. **`$(this).hide()`:** This jQuery code hides the current HTML element that triggers an event. It is often used in event handlers to hide the element that triggered the event.

2. **`$("p").hide()`:** This jQuery code hides all `<p>` elements on the page. It uses the "element" selector (`$("p")`) to select all paragraphs and then applies the `hide()` action to hide them.

3. **`$("p.test").hide()`:** This jQuery code hides all `<p>` elements with the class "test". It uses the "class" selector (`$("p.test")`) to select paragraphs with the specified class and hides them.

4. **`$("#test").hide()`:** This jQuery code hides the HTML element with the id "test". It uses the "id" selector (`$("#test")`) to select the element with the specified id and hides it.

These examples illustrate the basic structure of jQuery syntax: `$(selector).action()`. The dollar sign `$` is used to define jQuery, the `(selector)` is used to select HTML elements, and the `.action()` is used to perform an action on the selected element(s).

jQuery Selectors

- jQuery Selectors are used to select and manipulate HTML elements. They are very important part of jQuery library.
- With jQuery selectors, you can find or select HTML elements based on their id, classes, attributes, types and much more from a DOM.
- In simple words, you can say that selectors are used to select one or more HTML elements using jQuery and once the element is selected then you can perform various operation on that.
- All jQuery selectors start with a dollor sign and parenthesis e.g. $(). It is known as the factory function.

Factory Function:

- jQuery selectors start with the factory function: `$()`. This function is used to select and manipulate HTML elements.

Basic Selectors:

1. Tag Name:
- Example: `$("p")`
- Selects all `<p>` elements in the document.

2. Tag ID:
- Example: `$("#real-id")`
- Selects a specific element with the ID "real-id".

3. Tag Class:
- Example: `$(".real-class")`
- Selects all elements with the class "real-class".

jQuery Selectors Examples:

Example - Setting Background Color:

```
<!DOCTYPE html>
<html>
<head>
  <title>First jQuery Example</title>
  <script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>
  <script>
    $(document).ready(function() {
      $("p").css("background-color", "pink");
    });
  </script>
</head>
<body>
  <p>This is the first paragraph.</p>
  <p>This is the second paragraph.</p>
  <p>This is the third paragraph.</p>
</body>
</html>
```

Commonly Used Selectors:

1. ***Universal Selector:***
- Example: `$("*")`
- Selects all elements in the document.

2. ID Selector:
- Example: `$("#firstname")`
- Selects the element with ID "firstname".

3. *Class Selector:*
- Example: `$(".primary")`
- Selects all elements with the class "primary".

4. ***Element Selector:***
   - Example: `$("p")`
   - Selects all `<p>` elements.
5. ***Multiple Elements Selector:***
   - Example: `$("h1, div, p")`
   - Selects all `<h1>`, `<div>`, and `<p>` elements.

Additional Selectors:

Filtering:
- `:first`, `:last`, `:even`, `:odd`

Child and Descendant:
- `:first-child`, `:first-of-type`, `:last-child`, `:last-of-type`
- `:nth-child(n)`, `:nth-last-child(n)`, `:nth-of-type(n)`, `:nth-last-of-type(n)`
- `:only-child`, `:only-of-type`

Attribute Selectors:
- `[attribute]`, `[attribute=value]`, `[attribute!=value]`, `[attribute$=value]`
- `[attribute|=value]`, `[attribute^=value]`, `[attribute~=value]`, `[attribute*=value]`

Form Element Selectors:
- `:input`, `:text`, `:password`, `:radio`, `:checkbox`
- `:submit`, `:reset`, `:button`, `:image`, `:file`

State Selectors:
- `:enabled`, `:disabled`, `:selected`, `:checked`

Other Selectors:
- `:header`, `:animated`, `:focus`, `:contains(text)`, `:has(selector)`
- `:empty`, `:parent`, `:hidden`, `:visible`, `:root`, `:lang(language)`

These selectors help in targeting specific elements or groups of elements for manipulation in jQuery.

JQuery Events

In jQuery, events are actions or occurrences that happen on a web page. jQuery provides a simple and efficient way to handle these events. Here's an overview of jQuery events and how they are used:

Basic Syntax:

```
$(selector).event(function() {
   // Code to be executed when the event occurs
});
```

Common jQuery Events:

Click Event:

Triggers when a specified element is clicked.

```
$("button").click(function() {
   // Code to be executed on button click
});
```

Double Click Event:

Triggers when a specified element is double-clicked.

```
$("p").dblclick(function() {
   // Code to be executed on paragraph double-click
});
```

Mouse Enter and Mouse Leave Events:

Triggers when the mouse enters or leaves a specified element.

```
$("div").mouseenter(function() {
   // Code to be executed when the mouse enters the div
});

$("div").mouseleave(function() {
   // Code to be executed when the mouse leaves the div
});
```

Change Event:

Triggers when the value of an input element changes (for form elements).

```
$("input").change(function() {
   // Code to be executed when the input value changes});
```

Keydown, Keypress, and Keyup Events:

Triggered when a key on the keyboard is pressed, while it's being pressed, or when it's released.

```
$("input").keydown(function() {
  // Code to be executed when a key is pressed
});

$("input").keypress(function() {
  // Code to be executed when a key is pressed and held down
});

$("input").keyup(function() {
  // Code to be executed when a key is released
});
```

Submit Event:

Triggers when a form is submitted.

```
$("form").submit(function() {
  // Code to be executed when the form is submitted
});
```

Focus and Blur Events:

Triggers when an element gains or loses focus.

```
$("input").focus(function() {
  // Code to be executed when the input gets focus
});

$("input").blur(function() {
  // Code to be executed when the input loses focus
});
```

Resize Event:

Triggers when the browser window is resized.

```
$(window).resize(function() {
  // Code to be executed on window resize
});
```

Custom Events:

Besides the standard events, you can also create and trigger custom events using jQuery.

Triggering a Custom Event:

```
$("#myButton").on("customEvent", function() {
   // Code to be executed when the custom event is triggered
});

// Triggering the custom event
$("#myButton").trigger("customEvent");
```

These are just a few examples of the many events that jQuery supports. Events provide a way to enhance the interactivity and responsiveness of a web page by responding to user actions and system events.

jQuery HTML

In jQuery, HTML manipulation involves dynamically changing or updating the content and structure of HTML elements. jQuery provides methods to easily select and modify HTML elements. Here are some common HTML manipulation methods in jQuery:

Setting Content:

text() Method:

```
Sets or returns the text content of selected elements.
// Set text content
$("p").text("New text content");

// Get text content
var content = $("p").text();
```

html() Method:

```
Sets or returns the HTML content of selected elements.
// Set HTML content
$("div").html("<p>New HTML content</p>");

// Get HTML content
var content = $("div").html();
```

Adding or Changing Attributes:

attr() Method:

Gets the value of an attribute for the first element in the set of matched elements or sets one or more attributes for every matched element.

```
// Set attribute
$("img").attr("src", "newimage.jpg");

// Get attribute value
var srcValue = $("img").attr("src");
```

addClass(), removeClass(), toggleClass() Methods:

Add, remove, or toggle a class on selected elements.

```
// Add a class
$("div").addClass("newClass");

// Remove a class
$("div").removeClass("oldClass");

// Toggle a class
$("p").toggleClass("highlight");
```

Inserting and Appending Elements:

append() Method:

Insert content at the end of each element in the set of matched elements.

```
// Append content
$("ul").append("<li>New item</li>");
```

prepend() Method:

Insert content at the beginning of each element in the set of matched elements.

```
// Prepend content
$("ul").prepend("<li>New item</li>");
```

Removing Elements:

remove() Method:

Remove the set of matched elements from the DOM.

```
// Remove element
$("div").remove();
```

CSS Manipulation:

css() Method:

Get the computed style properties or set one or more CSS properties for the set of matched elements.

```
// Set CSS property
$("div").css("color", "blue");


// Get CSS property value
var colorValue = $("div").css("color");
```

Traversal and Filtering:

find() Method:

Get the descendants of each element in the current set of matched elements, filtered by a selector, jQuery object, or element.

```
// Find descendants
$("div").find("p");
```

filter() Method:

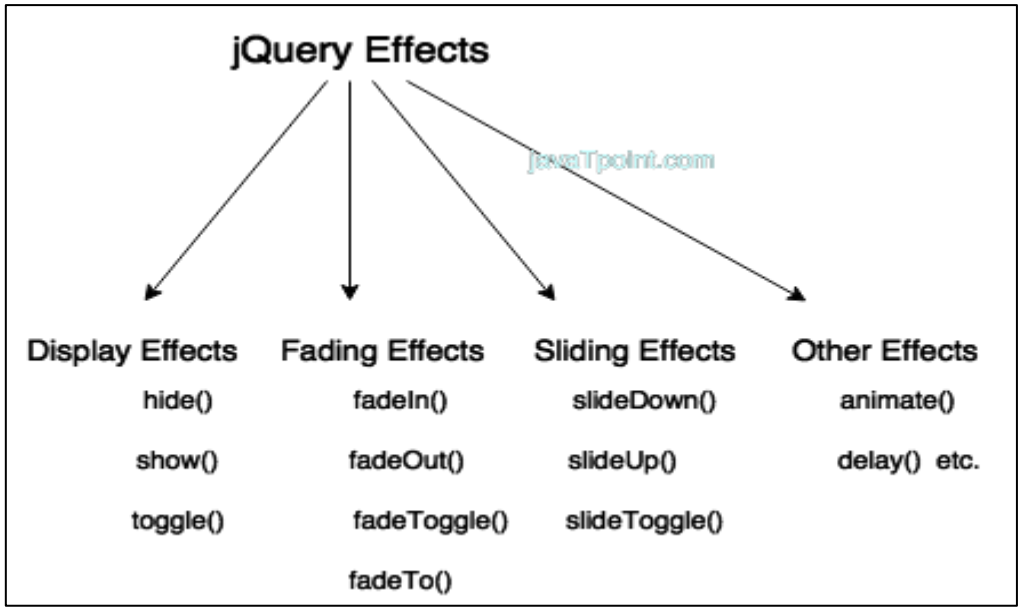
Reduce the set of matched elements to those that match the selector or pass the function's test.

```
// Filter elements
$("li").filter(".selected");
```

These jQuery methods make it easy to manipulate the HTML content and structure dynamically, providing a more interactive and responsive user experience on a web page.

jQuery EFFECT

jQuery provides a set of built-in effects that allow you to add animations and transitions to your web page elements. These effects can enhance the user experience and make the interface more dynamic. Here are some common jQuery effects:



1. Hide and Show Effects:

hide() Method:

Hides the selected elements with a sliding motion.

// Example: Hide a paragraph with a sliding motion

```
$("p").hide("slow");
```

show() Method:

Shows the selected elements with a sliding motion.

// Example: Show a hidden paragraph with a sliding motion

```
$("p").show("fast");
```

2. Toggle Effect:

toggle() Method:

Toggles between hiding and showing the selected elements.

// Example: Toggle visibility of a paragraph with a sliding motion

```
$("p").toggle("medium");
```

3. Fade Effects:

fadeIn(), fadeOut(), and fadeToggle() Methods:

fadeIn(): Fades in the selected elements.

fadeOut(): Fades out the selected elements.

fadeToggle(): Toggles between fading in and fading out.

```
// Example: Fade in a paragraph
$("p").fadeIn("slow");
// Example: Fade out a paragraph
$("p").fadeOut("fast");
// Example: Toggle fading of a paragraph
$("p").fadeToggle("medium");
```

4. Slide Effects:

slideDown(), slideUp(), and slideToggle() Methods:

slideDown(): Slides down the selected elements.

slideUp(): Slides up the selected elements.

slideToggle(): Toggles between sliding down and sliding up.

```
// Example: Slide down a div
$("div").slideDown("slow");

// Example: Slide up a div
$("div").slideUp("fast");

// Example: Toggle sliding of a div
$("div").slideToggle("medium");
```

5. Animate Method:

animate() Method:

Creates custom animations on selected elements.

```
// Example: Animate the width and opacity of a div
$("div").animate({
    width: "200px",
    opacity: 0.5
}, 1000);
```

6. Callback Function:

Many of these methods also accept a callback function that gets executed once the animation is complete:

```
// Example: Hide a paragraph and execute a function after the animation
$("p").hide("slow", function(){
   alert("Animation complete!");
});
```

These are just a few examples of the many effects jQuery provides. Effects can be combined, chained, and customized to create a wide variety of dynamic and visually appealing interactions on your web page.

jQuery css()

In jQuery, the css() method is used to get or set the CSS properties of HTML elements. It provides a convenient way to dynamically manipulate the styling of elements on a web page. Here are examples of using the css() method in various scenarios:

1. Setting CSS Properties:

Setting a Single Property:

```
// Example: Set the color of paragraphs to red
$("p").css("color", "red");
```

Setting Multiple Properties:

```
// Example: Set multiple CSS properties for a div
$("div").css({
   "color": "blue",
   "font-size": "16px",
   "background": "#f0f0f0"
});
```

2. Getting CSS Properties:

Getting a Single Property:

```
// Example: Get the font-size of the first paragraph
var fontSize = $("p:first").css("font-size");
console.log("Font Size: " + fontSize);
```

Getting Multiple Properties:

```
// Example: Get multiple CSS properties of an image
var imgProperties = $("img").css(["width", "height", "border"]);
console.log(imgProperties);
```

3. Modifying Existing Styles:

Modifying Existing Styles:

```
// Example: Increase the font-size of paragraphs by 2 pixels
$("p").css("font-size", function(index, value) {
  return parseFloat(value) + 2 + "px";
});
```

4. Manipulating Classes:

Adding a Class:

```
// Example: Add a class to a div
$("div").addClass("highlight");
```

Removing a Class:

```
// Example: Remove a class from a div
$("div").removeClass("highlight");
```

Toggling a Class:

```
// Example: Toggle a class on a button click
$("button").click(function() {
  $("div").toggleClass("highlight");
});
```

5. Chaining:

The css() method supports method chaining, allowing you to perform multiple operations in a single line.

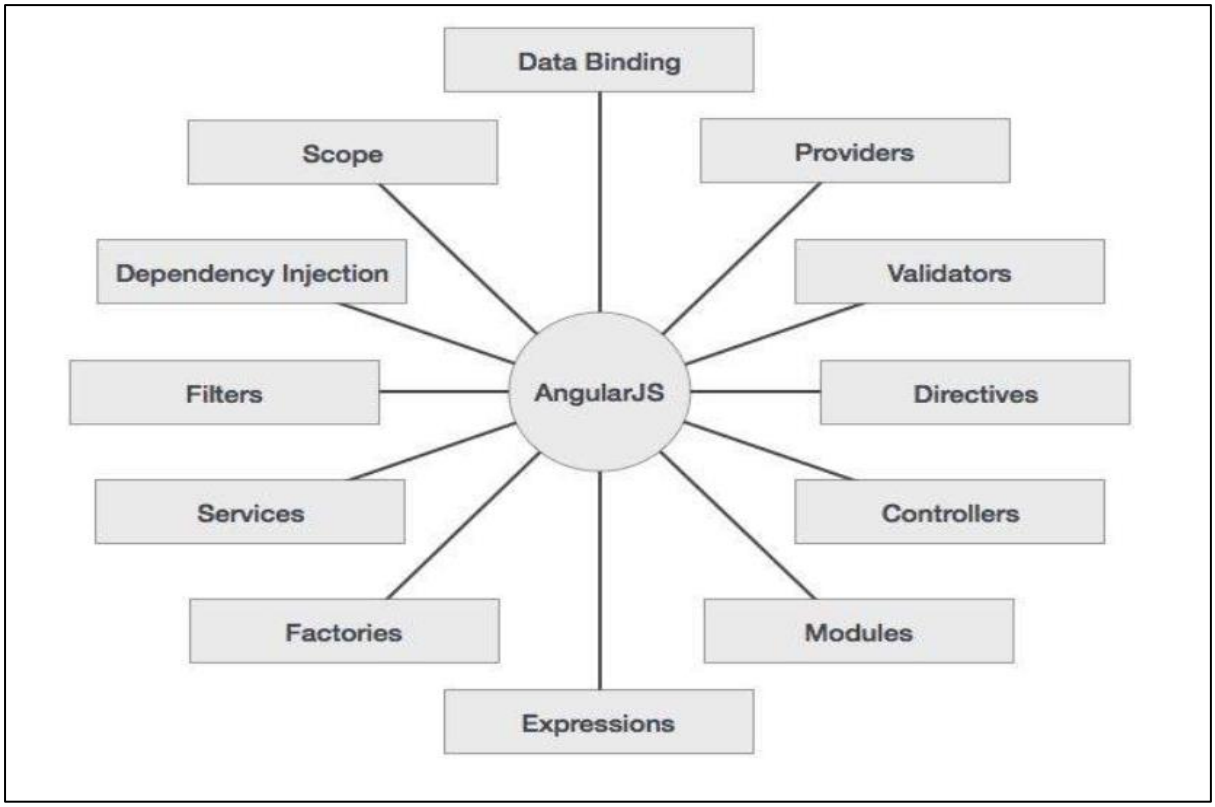
```
// Example: Chain multiple CSS operations
$("p")
  .css("color", "green")
  .css("font-weight", "bold")
  .hide("slow");
```

These examples demonstrate how the css() method in jQuery can be used to manipulate the styling of HTML elements, either by setting or retrieving CSS properties. It provides a flexible and efficient way to dynamically update the appearance of elements on your web page.

INTRODUCTION TO ANGULAR JS

AngularJS is a JavaScript-based open-source front-end web application framework maintained by Google. It's designed to make both the development and testing of such applications easier. AngularJS is part of the broader Angular framework family, with later versions referred to as just "Angular."

Key Features and Concepts:



1. **Two-way Data Binding:**
    a. AngularJS provides two-way data binding, meaning changes in the user interface automatically update the application state and vice versa.
2. **MVC Architecture:**
    a. AngularJS follows the Model-View-Controller (MVC) architectural pattern, which helps in organizing and structuring code.
3. **Directives:**
    a. Directives are markers on DOM elements that tell AngularJS to attach a specified behavior to that DOM element or transform the DOM element and its children.

4. **Dependency Injection:**
    a. AngularJS has a built-in dependency injection subsystem that helps developers create, understand, and test applications more efficiently.
5. **Templates:**
    a. HTML templates with additional AngularJS-specific elements are used to define the UI of an AngularJS application. These templates are compiled by the browser and the resulting views are used to render the UI.
6. **Controllers:**
    a. Controllers in AngularJS are JavaScript functions that are bound to a particular scope and help in organizing and controlling the application logic.
7. **Modules:**
    a. Modules are containers for different parts of an application, like controllers, services, filters, etc. They help in organizing and structuring the application.
8. **Services:**
    a. Services are reusable components or objects that can be used across controllers. They are often used to organize and share code across the application.
9. **Filters:**
    a. Filters select a subset of items from an array and return a new array. They are used for formatting data to be displayed to the user.
10. **Routing:**
    a. AngularJS provides a powerful and flexible routing system that allows developers to build Single Page Applications (SPAs) where views are updated dynamically as the user interacts with the application.

Example:

Here's a simple example of an AngularJS application:

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <title>AngularJS Example</title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.0/angular.min.js"
></script>
</head>
```

```
<body>

<div ng-controller="myController">
  <input type="text" ng-model="name">
  <h1>Hello, {{ name }}</h1>
</div>

<script>
  var app = angular.module('myApp', []);
  app.controller('myController', function($scope) {
    $scope.name = 'World';
  });
</script>

</body>
</html>
```

This example creates a simple AngularJS application with a controller that binds a variable name to an input field and displays a greeting message. As the user types in the input field, the greeting message dynamically updates.

AngularJS has been widely used for building dynamic and interactive web applications. However, it's important to note that AngularJS has reached its end of life, and developers are encouraged to migrate to the newer versions of the Angular framework for continued support and updates.

AngularJS DIRECTIVES

AngularJS directives are markers on a DOM element that tell AngularJS to attach a specified behavior to that DOM element or transform the DOM element and its children. Directives are a core feature of AngularJS, allowing developers to extend HTML with new attributes and elements to create dynamic and reusable components. Here are some commonly used AngularJS directives:

1. ng-app:

Defines the root element of an AngularJS application. It indicates the starting point of the AngularJS application.

```
<div ng-app="myApp">
  <!-- Your AngularJS application content goes here -->
</div>
```

2. ng-controller:

Attaches a controller to a DOM element, defining the controller's scope.

```
<div ng-controller="myController">
  <!-- Controller-specific content goes here -->
</div>
```

3. ng-model:

Binds an HTML element (like an input field) to a property on the scope, enabling two-way data binding.

```
<input type="text" ng-model="username">
```

4. ng-repeat:

Iterates over a collection (array or object) and repeats a set of HTML elements for each item in the collection.

```
<ul>
  <li ng-repeat="item in items">{{ item.name }}</li>
</ul>
```

5. ng-click:

Specifies an expression to evaluate when an element is clicked.

```
<button ng-click="doSomething()">Click me</button>
```

6. ng-show and ng-hide:

Conditionally shows or hides an element based on the evaluation of an expression.

```
<p ng-show="isVisible">This is visible</p>
<p ng-hide="isVisible">This is hidden</p>
```

7. ng-if:

Conditionally includes or excludes an element based on the evaluation of an expression.

```
<div ng-if="isDisplayed">This is displayed</div>
```

8. ng-class:

Conditionally applies CSS classes based on the evaluation of an expression.

```
<div ng-class="{ 'highlight': isHighlighted, 'italic': isItalic }">Styled Text</div>
```

9. ng-src:

Binds the src attribute of an HTML element to an expression, typically used with images.

```
<img ng-src="{{ imageUrl }}">
```

10. ng-disabled:

Disables or enables an HTML element based on the evaluation of an expression.

```
<button ng-disabled="isDisabled">Click me</button>
```

11. ng-options:

Generates a dropdown list (<select>) based on the values of an array.

```
<select ng-model="selectedOption" ng-options="item.id as item.name for item in options"></select>
```

These are just a few examples of the many directives available in AngularJS. Directives play a crucial role in extending HTML to create dynamic, interactive, and reusable components in AngularJS applications.

AngularJS Expressions

AngularJS expressions are JavaScript-like code snippets enclosed in double curly braces ({{ }}). These expressions are used to bind data from the AngularJS scope to the HTML and are evaluated against the current scope. AngularJS expressions are commonly used to display data in the view, and they support a subset of JavaScript expressions.

Here are some key points about AngularJS expressions:

1. **Data Binding:** AngularJS expressions provide two-way data binding between the model (JavaScript variables in the scope) and the view (HTML elements). When the model changes, the view is automatically updated, and vice versa.

```
<p>{{ message }}</p>
```

2. Simple Expressions: Expressions can be as simple as displaying a variable value.

```
<p>{{ username }}</p>
```

3. Arithmetic Operations: Basic arithmetic operations can be performed in expressions.

```
<p>{{ num1 + num2 }}</p>
```

4. String Concatenation: Strings can be concatenated using expressions.

```
<p>{{ "Hello, " + username }}</p>
```

5. Object Property Access: You can access properties of objects defined in the scope.

```
<p>{{ user.name }}</p>
```

6. Array Access: You can access elements of an array using expressions.

```
<p>{{ myArray[0] }}</p>
```

7. Function Calls: You can call functions defined in the scope.

```
<p>{{ getGreeting() }}</p>
```

8. **Conditional Statements:** Expressions can include simple conditional statements.

```
<p>{{ isLogged ? 'Welcome' : 'Please log in' }}</p>
```

9. Filters: AngularJS expressions support filters for formatting data.

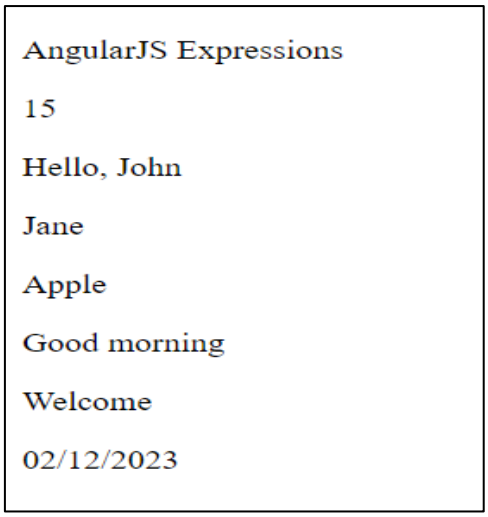
```
<p>{{ dateValue | date:'dd/MM/yyyy' }}</p>
```

10. Expression Limitations: AngularJS expressions do not support certain JavaScript features like loops, conditionals, and exceptions. They are intentionally limited for security reasons.

Example:

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <title>AngularJS Expressions</title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.0/angular.min.js"
></script>
</head>
<body ng-controller="myController">

  <p>{{ message }}</p>
```

```
    <p>{{ num1 + num2 }}</p>
    <p>{{ "Hello, " + username }}</p>
    <p>{{ user.name }}</p>
    <p>{{ myArray[0] }}</p>
    <p>{{ getGreeting() }}</p>
    <p>{{ isLogged ? 'Welcome' : 'Please log in' }}</p>
    <p>{{ dateValue | date:'dd/MM/yyyy' }}</p>

    <script>
      var app = angular.module('myApp', []);
      app.controller('myController', function($scope) {
        $scope.message = 'AngularJS Expressions';
        $scope.num1 = 5;
        $scope.num2 = 10;
        $scope.username = 'John';
        $scope.user = { name: 'Jane' };
        $scope.myArray = ['Apple', 'Banana', 'Orange'];
        $scope.getGreeting = function() {
          return 'Good morning';
        };
        $scope.isLogged = true;
        $scope.dateValue = new Date();
      });
    </script>
  </body>
</html>
```



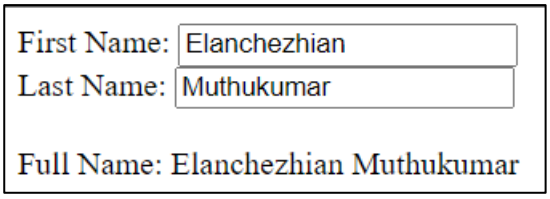
In this example, AngularJS expressions are used to display various data and perform simple operations in the view. The expressions are evaluated against the AngularJS controller's scope.

AngularJS CONTROLLER

AngularJS controllers are JavaScript functions that manage the application data and behavior. They are responsible for handling user input, updating the model, and interacting with services.

Example 1: Basic AngularJS Controller

```
<!DOCTYPE html>
<html>
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
</script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">
First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{firstName + " " + lastName}}
</div>

<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
   $scope.firstName = "Elanchezhian";
   $scope.lastName = "Muthukumar";
});
</script>

</body>
</html>
```

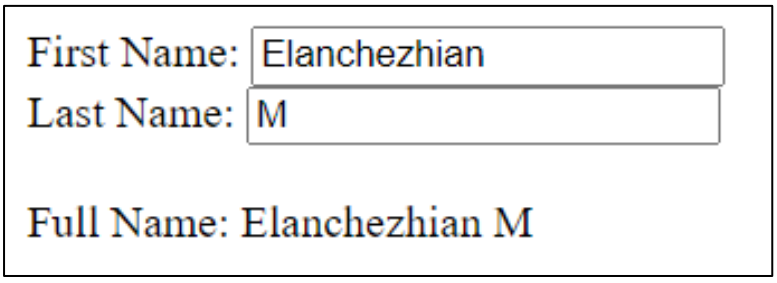


Explanation:

- ✓ ng-app="myApp" defines the AngularJS application named "myApp."
- ✓ ng-controller="myCtrl" associates the controller named "myCtrl" with the specified .
- ✓ $scope is the application object, and it holds variables (firstName and lastName) and functions.
- ✓ ng-model binds input fields to controller properties.
- ✓ The {{firstName + " " + lastName}} expression displays the full name.

Example 2: AngularJS Controller with Methods

```
<!DOCTYPE html>
<html>
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js">
</script>
<body>

<div ng-app="myApp" ng-controller="personCtrl">

First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{fullName()}}

</div>

<script>
var app = angular.module('myApp', []);
app.controller('personCtrl', function($scope) {
    $scope.firstName = "Elanchezhian";
    $scope.lastName = "M";
    $scope.fullName = function() {
        return $scope.firstName + " " + $scope.lastName;
    };
});
</script>

</body>
</html>
```



Explanation:

- ✓ This example adds a method, fullName(), to the controller.
- ✓ The input fields are still bound using ng-model.
- ✓ The expression {{fullName()}} calls the fullName function to display the full name.

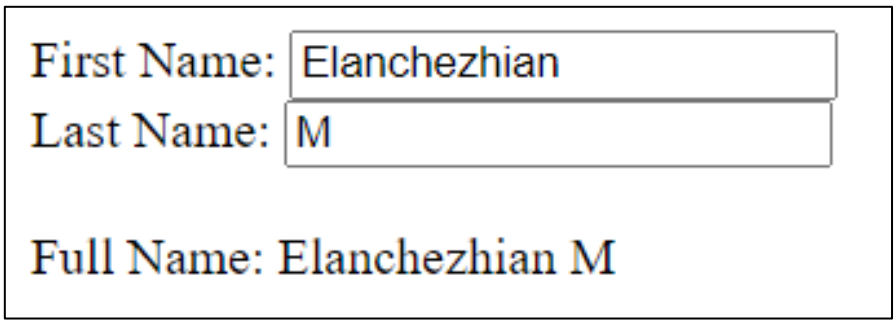
External Controller File

You can store controllers in external files. Example file named "personController.js":

```
angular.module('myApp', []).controller('personCtrl', function($scope) {
    $scope.firstName = "Elan";
    $scope.lastName = "M";
    $scope.fullName = function() {
        return $scope.firstName + " " + $scope.lastName;
    };
});
```

Include the external file in the HTML:

```
<!DOCTYPE html>
<html>
<script
src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.m
in.js"></script>
<body>
<div ng-app="myApp" ng-controller="personCtrl">
First Name: <input type="text" ng-model="firstName"><br>
Last Name: <input type="text" ng-model="lastName"><br>
<br>
Full Name: {{firstName + " " + lastName}}
</div>
<script src="personController.js"></script>
</body>
</html>
```

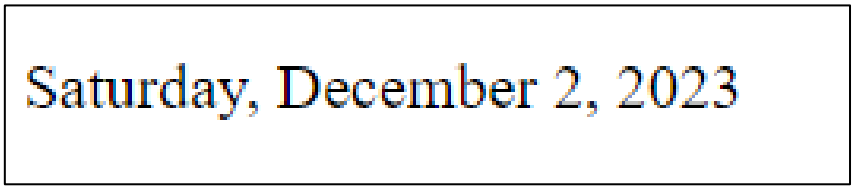


FILTERS:

AngularJS filters are used for formatting data in the view. They can be applied to expressions in the view to modify the way data is displayed.

Example:

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
   <title>AngularJS Filter Example</title>
   <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.0/angular.min.js"
></script>
</head>
<body ng-controller="myController">

   <p>{{ currentDate | date:'fullDate' }}</p>

   <script>
      var app = angular.module('myApp', []);
      app.controller('myController', function($scope) {
         $scope.currentDate = new Date();
      });
   </script>

</body>
</html>
```



SERVICES:

AngularJS services are objects or functions that carry out specific tasks and can be used throughout the application. They provide a way to organize and share code across controllers.

Example:

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <title>AngularJS Service Example</title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.0/angular.min.js"
></script>
</head>
<body ng-controller="myController">

  <h1>{{ greetingService.sayHello() }}</h1>

  <script>
    var app = angular.module('myApp', []);
    app.service('greetingService', function() {
      this.sayHello = function() {
        return 'Hello from the Service!';
      };
    });

    app.controller('myController', function($scope, greetingService) {
      $scope.greetingService = greetingService;
    });
  </script>
</body>
</html>
```

Hello from the Service!

EVENTS

In AngularJS, events are a mechanism for communication between different components of an application. Events allow one part of your application to notify other parts that something has occurred, and they provide a way for these components to react to those notifications.

Example: AngularJS Events

```
<!DOCTYPE html>
<html ng-app="myApp">

<head>
  <title>AngularJS Events Example</title>
  <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.0/angular.min.js"
></script>
</head>

<body ng-controller="ParentController">

  <div>
    <h2>Parent Component</h2>
    <button ng-click="broadcastEvent()">Broadcast Event</button>
  </div>

  <div ng-controller="ChildController">
    <h2>Child Component</h2>
    <div>{{ eventData }}</div>
  </div>

  <script>
    var app = angular.module('myApp', []);

    app.controller('ParentController', function ($scope, $rootScope) {
      $scope.broadcastEvent = function () {
        $rootScope.$broadcast('customEvent', { data: 'Event data from
Parent' });
        };
    });
```

```
        app.controller('ChildController', function ($scope) {
            $scope.$on('customEvent', function (event, args) {
                $scope.eventData = args.data;
            });
        });
    </script>

</body>

</html>
```

Explanation:

- ✓ There are two controllers: `ParentController` and `ChildController`.
- ✓ The `ParentController` broadcasts a custom event called `''customEvent''` when a button is clicked.
- ✓ The `ChildController` listens for the `''customEvent''` and updates the view with the received data.

Key Points:

- ✓ `$rootScope.$broadcast('eventName', eventData)` is used to broadcast an event from a parent scope.
- ✓ `$scope.$on('eventName', function(event, args) { /* handle event */ })` is used to listen for an event in a child scope.
- ✓ Events can carry data, and this data can be accessed through the `args` parameter.

FORMS IN AngularJS:

AngularJS provides a robust framework for handling forms in web applications. Forms consist of various input controls such as text fields, checkboxes, radio buttons, select boxes, and more. The ng-model directive is essential for achieving data-binding between these controls and the underlying AngularJS application.

Input Controls:

Text Input:

```
<input type="text" ng-model="firstname">
```

The ng-model directive binds the input controller to the application's data, creating a property named firstname.

Checkbox:

```
<input type="checkbox" ng-model="myVar">
```

Checkboxes have a boolean value (true or false). The ng-model directive binds the checkbox value to the application.

Radio Buttons:

```
<form>
  Pick a topic:
  <input type="radio" ng-model="myVar" value="dogs"> Dogs
  <input type="radio" ng-model="myVar" value="tuts"> Tutorials
  <input type="radio" ng-model="myVar" value="cars"> Cars
</form>
```

Radio buttons with the same ng-model but different values allow selection among multiple options.

Select Box:

```
<form>
  Select a topic:
  <select ng-model="myVar">
    <option value="">Choose...</option>
    <option value="dogs">Dogs</option>
    <option value="tuts">Tutorials</option>
    <option value="cars">Cars</option>
  </select>
</form>
```

The ng-model directive binds the selected option's value to the myVar property.

FORM VALIDATION:

AngularJS facilitates form validation to ensure data integrity. Validation messages can be displayed using directives like ng-show based on the form's state.

```
<input type="text" ng-model="firstname" required>
<div ng-show="userForm.firstname.$dirty &&
userForm.firstname.$error.required">
  First Name is required.
</div>
```

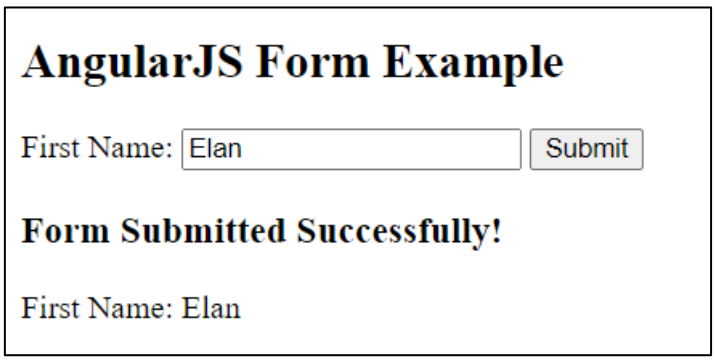
In this example, the error message is shown only when the field is dirty and the required condition is not met.

FORM EXAMPLE:

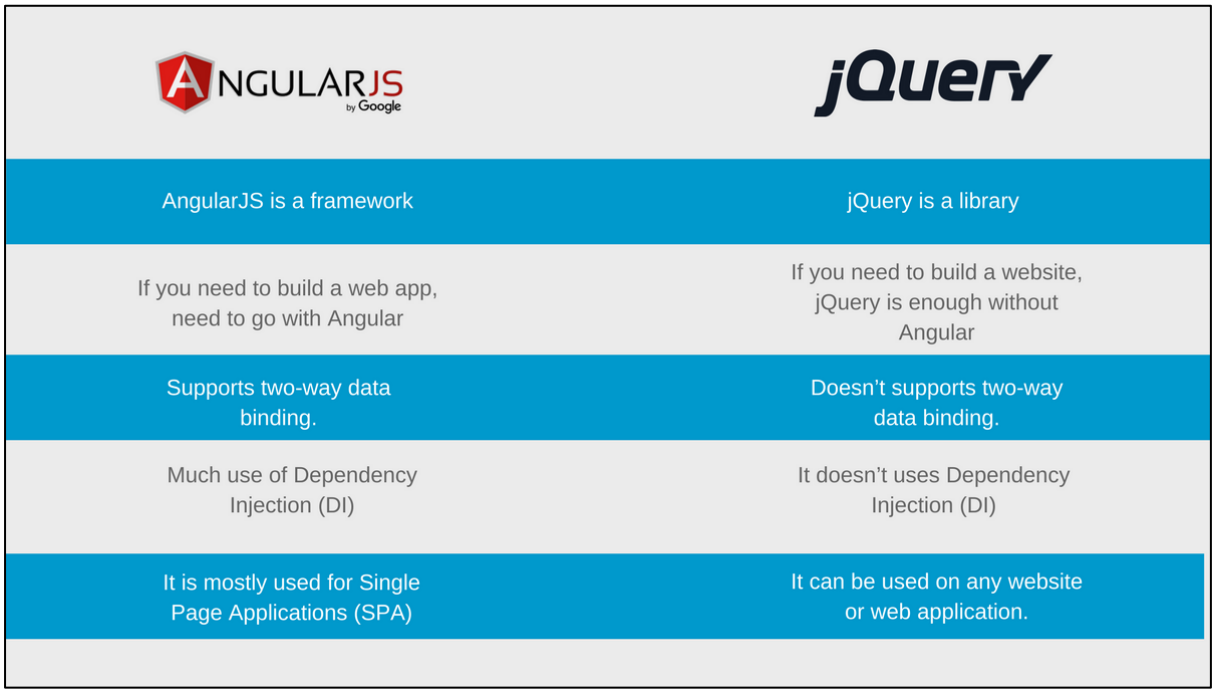
Here's a complete example of an AngularJS form with data-binding and validation:

```
<!DOCTYPE html>
<html ng-app="myApp">

<head>
   <title>AngularJS Forms</title>
   <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.0/angular.min.js"
></script>
</head>

<body ng-controller="formCtrl">

   <h2>AngularJS Form Example</h2>

   <form name="userForm" novalidate>
      <label for="firstName">First Name:</label>
      <input type="text" id="firstName" name="firstName" ng-
model="user.firstName" required>
      <div ng-show="userForm.firstName.$dirty &&
userForm.firstName.$error.required" style="color: red;">
         First Name is required.
      </div>
```

```
        <!-- Other form fields and validation messages -->

        <button ng-click="submitForm()" ng-
disabled="userForm.$invalid">Submit</button>
    </form>

    <!-- Display submitted data -->
    <div ng-if="submitted">
        <h3>Form Submitted Successfully!</h3>
        <p>First Name: {{ user.firstName }}</p>
        <!-- Display other submitted data -->
    </div>

    <script>
        var app = angular.module('myApp', []);
        app.controller('formCtrl', function ($scope) {
            $scope.user = {};
            $scope.submitted = false;

            $scope.submitForm = function () {
                if ($scope.userForm.$valid) {
                    // Form submission logic can be added here
                    $scope.submitted = true;
                }
            };
        });
    </script>
</body>
</html>
```



In this example, the form includes text input, validation messages, and a submit button. The submitted data is displayed after a successful form submission.



jQuery	AngularJS
jQuery is a library.	AngularJS is a framework.
If you want to build a web site, jQuery is enough without Angular.	If you want to build web application, need to go with Angular.
Doesn't supports two-way data binding.	Supports two-way data binding.
It doesn't uses Dependency Injection (DI)	Much use of Dependency Injection (DI)
It can be used on any website or web application.	It is mostly used for Single Page Applications (SPA)

Usage:
 	
$(document).ready(function(){
$("p").click(function(){
$(this).hide();
});
});
	Usage:
	
	 Email : <input type = "text" ng-model="email">
	
	 Your email id is {{email}}

We have talked a lot about the differences now let's focus on the pros and cons for using AngularJS.

Pros

- Easy to learn
- Perfect for Single Page Applications
- Easy to unit test
- Code less, get more functionality
- Provided re-usability with the help of components

Cons

- Applications written in AngularJS are not that secure.
- If the end-user disables JavaScript on their browsers then they only see the basic HTML page.