JavaScript Origins:
- Developed by Netscape as LiveScript, later became a joint venture with Sun in 1995, renamed JavaScript.
- Standardized by ECMA-262 and ISO-16262.
- Initially designed as an XHTML-embedded scripting language.

JavaScript Components:
1. **The Core:**
   - Operators, expressions, statements, and subprograms.

2. **Client Side:**
   - Objects supporting browser control and user interactions.
   - Used for making XHTML documents responsive to user inputs.

3. **Server Side:**
   - Objects facilitating server-side tasks, e.g., communication with a DBMS.

JavaScript Versions:
- Evolved over time with various versions.

JavaScript vs. Java:
- Only related in syntax.
- Dynamically typed and interpreted.
- JavaScript is not a full-fledged object-oriented language.

Uses of JavaScript:
- Load distribution from server to client.
- User interactions through forms.
- Event-driven computation for dynamic XHTML documents.
- Can replace Java applets, integral part of XHTML documents.

JavaScript Execution:
- Executed entirely by the browser.
- No continuous exchange of information with the server.
- Can issue HTTP requests and load other pages.
- Does not require Java VM, resulting in fast execution.

Event-Driven Computation:
- Actions executed in response to user actions.
- Supports user interactions through XHTML form elements.
- Efficient for input data validation on the client, saving server and internet time.

JavaScript in XHTML Documents:
- Scripts can appear in the head or body.
- Head scripts for content requested or reacting to user interactions.
- Body scripts interpreted once found by the interpreter.

Object Orientation:

- JavaScript is object-based, not object-oriented.
- No support for class-based inheritance.
- Objects are collections of properties, either data or method properties.
- Root object is Object; all objects derive from it.

JavaScript Objects:
- Collections of properties (data or methods).
- Accessible through variables.
- Objects are dynamic; properties can be added or deleted.

Embedding in XHTML Docs:
- Directly or indirectly using `<script>` tags.
- Syntactic characteristics include identifiers, case sensitivity, reserved words, comments, and script hiding.

Primitives:
- Number, String, Boolean, Undefined, Null.
- Wrapper objects for Number and String.

Numeric Operators:
- Basic arithmetic operators.

Math and Number Objects:
- Math object provides mathematical functions.
- Number object has properties like toString() and isNaN().

String Catenation Operator:
- Concatenation specified with the + sign.

Implicit and Explicit Type Conversions:
- Implicit conversions during operations.
- Explicit conversions using constructors or functions like parseInt() and parseFloat().

String Object:
- Properties like length.
- Methods for string manipulation.

Declaring Variables:
- Dynamically typed using `var`.
- Variables can be implicitly or explicitly declared.

Date Object:
- Created with the Date constructor.
- Methods for local time details.

Screen Output:
- Document and Window objects model HTML and browser display.

- Methods like `write` and dialog box methods (alert, confirm, prompt) for output.

Control Statements:
- Selection (if, if...else) and iteration (while, for, do...while) statements.
- Compound statements and control expressions.

Conditionals and Loops:
- Conditionals with `if`, `else`, and `switch`.
- Loops with `while`, `for`, `do...while`, and `for-in` (for objects).

Object Creation and Modification:
- Constructors create and initialize properties.
- Properties accessed using dot notation or square brackets.
- Object nesting and property deletion with `delete`.
- `for-in` loop for listing object properties.

### Arrays:
- Arrays in JavaScript can contain primitive values or references to other objects.
- They can be created using `new Array()` or array literals (`[]`).
- Array length is dynamic and is stored in the `length` property.
- The `length` property is writable (`myList.length = 150`), but it doesn't mean all 150 elements are assigned.

#### Array Methods:
1. **`join()` Method:**
   - Converts array elements to strings and concatenates them into a single string.
   - Example: `names.join(":")` results in `"Mary : Murray : Murphy : Max"`.

2. **`reverse()` Method:**
   - Reverses the order of elements in the array.

3. **`sort()` Method:**
   - Converts elements to strings and sorts them.
   - Example: `names.sort()` results in `["Mary", "Max", "Murphy", "Murray"]`.

4. **`concat()` Method:**
   - Concatenates parameters to the end of the array.
   - Example: `names.concat("Moo", "Meow")` extends the array.

5. **`slice()` Method:**
   - Returns a part of the array specified by subscripts.
   - Example: `list.slice(1, 3)` results in `[4, 6]`.

6. **`toString()` Method:**
   - Returns array elements as a string separated by commas.

7. **`push()` and `pop()` Methods:**
   - Implement stack concepts in arrays.
   - Example: `deer = list.pop()`, `list.push("Blitzen")`.

8. **`shift()` and `unshift()` Methods:**
   - Remove and add elements at the beginning of the array.
   - Example: `deer = list.shift()`, `list.unshift("Dasher")`.

### Functions:
- Function definition includes the function's header and body.
- A function header consists of the reserved word `function`, name, and parameters.
- The return value is specified by the `return` statement; if not used, `undefined` is returned.
- Functions are objects and can be treated as such.

#### Function Parameters:
- Passed by value, but reference variables are pass-by-reference.
- No type or number of parameters checking.
- Excess parameters are ignored, and excess formal parameters are set to `undefined`.
- Parameters are sent through the `arguments` array.

#### Sort Method:
- Sorts array elements alphabetically.
- A comparison operation can be supplied to customize sorting.

### Constructors:
- Constructors create and initialize properties for newly created objects.
- Called by the `new` operator, with `this` referring to the new object.

### Pattern Matching Using Regular Expressions:
- Two approaches: RegExp object and String object methods.
- Metacharacters have special meanings and can be matched by preceding them with a backslash.
- Character classes, quantifiers, anchors, and modifiers enhance pattern matching.

### Errors in Scripts:
- JavaScript interpreter detects syntax errors and undefined variable uses.
- Debugging can be enabled in browsers to display error messages.
- Debuggers like IE7 debugger, Firebug, or Venkman for FX2 help identify and solve issues during execution.

## JavaScript Execution Environment:

- The `Window` object provides the largest enclosing referencing environment for scripts.
- Implicitly defined `Window` properties include `document` (a reference to the `Document` object) and `frames` (an array of references to the frames of the document).

## Document Object Model (DOM):

- DOM 0 is supported by all JavaScript-enabled browsers, but there is no written specification.
- DOM 1 (1998) focused on XHTML and XML.
- DOM 2 (2000) specifies a style sheet object model, document traversals, and a comprehensive event model.

- DOM 3 (2004) deals with content models for XML, document validation, document views, formatting, key events, and event groups.
- DOM 4 (2015) was published by WHATWG (Web Hypertext Application Technology Working Group).
- The DOM is an abstract model defining the interface between XHTML documents and application programs.
- XHTML elements are represented as objects, and element attributes are represented as properties.

## DOM Structure:

- Documents in the DOM have a tree-like structure.

## Element Access in JavaScript:

1. **DOM Address:**
   - Example: `Document.forms[0].elements[0]`.
   - Problem: Prone to issues if the document structure changes.

2. **Element Names:**
   - Requires elements and ancestors (except `body`) to have name attributes.
   - Example: `document.myForm.pushMe`.
   - Problem: XHTML 1.1 standard may not allow the name attribute in form elements.

3. **`getElementById` Method:**
   - Example: `document.getElementById("pushMe")`.
   - Useful for elements with ids.
   - Form elements often have ids and names both set to the same value.

## Events and Event Handling:

- An event is a notification that something specific has occurred.
- An event handler is a script executed in response to an event.
- Events include attributes like `onclick` and can be registered using various methods.

## DOM 2 Event Model:

- DOM 2 is more sophisticated and powerful than DOM 0.
- Modularized interface with submodules like HTML Events and Mouse Events.
- Events create an object passed to the handler, containing information about the event.
- Event propagation involves capturing, execution, and bubbling phases.
- Handlers registered using `addEventListener` method.

## Event Propagation:

- Event phases: capturing, execution, and bubbling.
- `stopPropagation` method stops further propagation.
- `preventDefault` method prevents the default action associated with an event.

## Event Handler Registration:

- DOM 0 uses attribute assignment and property assignment.
- DOM 2 uses `addEventListener` method for registration.
- `removeEventListener` method deletes the registration.

## Navigator Object:

- The `navigator` object indicates the browser being used.
- Properties like `appName` and `appVersion` provide browser information.

## DOM2 Tree Traversal and Modification:

### Traversal:

- Properties: `parentNode`, `previousSibling`, `nextSibling`, `firstChild`, `lastChild`, `nodeType`.

### Modification:

- Methods: `insertBefore(newChild, refChild)`, `replaceChild(newChild, oldChild)`, `removeChild(oldChild)`, `appendChild(newChild)`.

Dynamic HTML (DHTML):
- **Definition:** XHTML document allowing changes after display.
- **Implementation:** Embedded JavaScript manipulating DOM objects.
- **Browser Support:** Varies; DOM 0, DOM 2 (Firefox 2, not IE 8).

Element Positioning:
- **Before HTML 4.0:** Limited positioning options (tables, frames).
- **CSS-P (1997):** Introduces positioning properties.
  - `left`, `top`: Distances from reference point.
  - `position`: Absolute, relative, or static.

Absolute Positioning:
- Element placed at specific coordinates disregarding others.
- Example: `<p style="position: absolute; left: 100px; top: 200px;">`.

Relative Positioning:
- Does not affect initial position.
- Use for later displacement; `top` and `left` specify movement.

Static Positioning:
- Default; placed as if `position: relative`.
- Cannot have initial or changed `top`/`left`.

Moving Elements:
- Absolute/relative positioning allows movement.
- Example: User-input coordinates change element position.

Element Visibility:
- Toggle visibility by changing `style` property.
- Set `visibility` from `visible` to `hidden` or vice versa.

Changing Colors and Fonts:
- **Colors:** Modify `color` and `background-color`.
  - User input adjusts document colors.
- **Fonts:** Change size, style, weight dynamically.
  - Use mouse events for interactive changes.

Dynamic Content:
- Address element content with `value` property.
- Example: Help box content changes based on mouse position.

Stacking Elements:
- Third dimension using `z-index`.
- Higher `z-index` brings element to top of stack.

Locating the Mouse Cursor:
- Event object provides coordinates.
  - `clientX`/`clientY`: Browser window.
  - `screenX`/`screenY`: Display screen.

Reacting to a Mouse Click:
- Events like `mousedown` and `mouseup`.
- Example: Display message on mouse click.

Slow Movement of Elements:
- Use `setTimeout` for delayed execution.
- Small increments create smooth motion.

Dragging and Dropping Elements:
- Implemented using `mousedown`, `mouseup`, `mousemove`.
- Example: `grabber` function for drag-and-drop.
- DOM 0 for `mousedown`, DOM 2 for the rest.

## What is jQuery?

jQuery is a JavaScript library created by John Resig in 2006. Its primary goal is to simplify client-side scripting and enhance the interactivity of web pages. jQuery achieves this by providing a concise syntax and a set of powerful features that make common tasks, such as DOM manipulation and event handling, more accessible.

### Key Features:

1. **CSS Selectors:**
   - jQuery uses CSS selector syntax to easily select and manipulate elements on a webpage. This simplifies the process for developers who are already familiar with CSS.

2. **DOM Manipulation:**
   - jQuery simplifies the process of accessing and modifying the Document Object Model (DOM). This allows developers to dynamically change the content and structure of a webpage.

3. **Event Handling:**
   - jQuery provides a streamlined approach to handling events, making it easier to respond to user actions like clicks, keypresses, and more.

4. **Animations and Effects:**
   - It offers built-in methods for creating animations and effects, allowing developers to enhance the visual appeal of their web applications.

5. **AJAX:**
   - jQuery simplifies AJAX requests, making it easier to retrieve and send data asynchronously without having to write complex JavaScript code.

6. **Cross-browser Compatibility:**
   - jQuery handles browser inconsistencies, ensuring that the same code works consistently across different browsers.

7. **Plugins:**
   - Developers can extend jQuery's functionality through plugins, which are additional scripts that integrate seamlessly with the library.

## Why is jQuery Useful?

### Accessibility:

- jQuery's syntax is accessible to both programmers and non-programmers. Using CSS selectors for element selection makes it intuitive, especially for those familiar with CSS.

### Browser Inconsistencies:

- jQuery abstracts away the need for developers to address browser-specific issues. It provides a consistent interface, handling the integration of effects into each browser.

### Additional Functionality:

- jQuery's modular nature allows developers to add functionality through plugins when the core library lacks specific features. This flexibility keeps the library lightweight when additional functionality is not needed.

## Selecting Elements in jQuery:

### The `$()` Function:

- The `$()` function is used to select one or more elements on the document using CSS selectors. It returns a jQuery object representing the selected elements.

### Using CSS Selectors:
- jQuery supports various CSS selectors, including:
  - All Selector (`*`), Element Selector (`element`), Class Selector (`.class`), ID Selector (`#id`).

### Extended Selectors:

- **Descendant Selector:**
  - Selects elements that are descendants of a specified element.

- **Child Selector:**
  - Selects all specified elements that are children of a given parent element.

- **First, Last, Nth-child Selector:**
  - Selects elements based on their position within a parent element.

- **Not Selector:**
  - Selects elements that do not match a specified selection.

- **Attribute Selectors:**
  - Selects elements based on the presence or value of attributes.

### Extended jQuery Selectors:

- **Attribute Not Equal Selector:**
  - Selects elements with an attribute whose value is not equal to the specified value.

- **First, Last, Element at Index Selector:**
  - Selects a single element from a collection based on its position.

- **Even and Odd Selectors:**
  - Selects even or odd elements within a collection.

## Notes for VIVA Purpose:

- jQuery simplifies web development through its concise syntax and powerful features.
- It enhances accessibility by using CSS selectors for element manipulation.
- jQuery abstracts browser inconsistencies, ensuring a consistent experience across different browsers.
- The library's modular design allows for the addition of functionality through plugins.
- jQuery's `$()` function and CSS selector support simplify element selection and manipulation.
- Extended selectors provide more specific ways to select and manipulate elements.
- jQuery is widely used for its ease of use, cross-browser compatibility, and extensive community support.

Event Handling with jQuery: Simplifying Cross-Browser Compatibility

Introduction:
- jQuery simplifies event handling, eliminating the need to worry about browser differences.
- Methods like `click()` and `keydown()` replace traditional approaches, enhancing ease of use.

Document Ready:
- Use `ready()` function to ensure all elements are loaded before executing jQuery code.
- `ready()` vs. `load()` - former triggers when elements are loaded, while the latter waits for all media, useful for scripts involving images.

Basic Event Handling:
- jQuery provides shorthand methods for common events, like `click()`.
- Example: `$("#change-size").click(function() {...});`

Event Types:
- **Mouse Events:**
  - `click` and `dblclick` for single and double clicks.
  - `mousedown` and `mouseup` for specific phases of a click.
  - `mouseover`, `mouseout`, `mouseenter`, and `mouseleave` for mouse movements.

- **Keyboard Events:**
  - `keydown`, `keyup`, and `keypress` for key-related actions.

Event Capturing and Bubbling:
- Events have two phases: capturing and bubbling.
- jQuery always uses bubbling, ensuring consistent behavior across browsers.

Handling Mouseout Effectively:
- Use `mouseleave()` instead of `mouseout()` to prevent undesired bubbling.

Using `on()` Method:
- Introduced in jQuery 1.7 to replace `bind()`, `delegate()`, and `live()`.
- Allows binding multiple events and event delegation.
- Can be used for both existing and future elements.

Event Delegation:
- Methods like `delegate()` allow handling events on elements not yet in the DOM.

Triggering Events:
- `trigger()` method simulates an event without user interaction.
- Useful for automating actions tied to specific events.

Other Events:
- Events without shorthand methods can be handled using `on()`.
- Example: `$("#my-form").on("reset", function() {...});`

Conclusion:

- jQuery simplifies event handling, ensuring cross-browser compatibility.
- Understand the nuances of different events and leverage jQuery's methods for efficient scripting.

Notes for VIVA Purpose:
- Event handling in jQuery streamlines cross-browser compatibility.
- `on()` method introduced in jQuery 1.7 combines features of older methods.
- Event delegation is useful for dynamically added elements.
- `trigger()` allows simulating events programmatically.
- Familiarity with these concepts is crucial for efficient jQuery scripting.

PHP Introduction:

- **Origins and Uses:**
  - PHP (Personal Home Page or PHP: Hypertext Preprocessor) was created by Rasmus Lerdorf in 1994. Initially, it was developed for tracking visitors to his website.
  - The first public version of PHP was released in 1995.
  - PHP is open-source and is a server-side scripting language.
  - It is commonly used for form handling, file processing, and database access.
  - PHP supports 15 different databases, email protocols (POP3 and IMAP), and distributed object architectures (COM and CORBA).

- **PHP Overview:**
  - PHP is embedded in HTML documents and executed on the server side.
  - Similar to JavaScript, but it runs on the server.
  - It is an alternative to CGI, ASP.NET, and Java servlets.
  - PHP file extensions include .php, .php3, and .phtml.
  - PHP processor has two modes: copy mode (XHTML) and interpret mode (PHP).

- **Syntactic Characteristics:**
  - PHP code can be internal or external to XHTML documents.
  - Internal: Enclosed in `<?php ... ?>`.
  - External: Included using `include("myScript.inc")`.
  - Every variable begins with a `$`, followed by a letter or underscore, consisting of letters, underscores, or digits.
  - PHP is dynamically typed and supports both procedural and object-oriented programming.

- **Variables:**
  - No type declarations; unassigned variables have the value NULL.
  - `unset` function sets a variable to NULL.
  - `IsSet($variable)` checks if a variable is not NULL.
  - Reserved words are not case-sensitive.

- **Primitives:**
  - Four scalar types: Boolean, integer, double, and string.
  - Two compound types: array and object.
  - Two special types: resource and NULL.
  - PHP has functions for type checking like `is_int()`, `is_string()`, etc.

- **Strings:**
  - Characters are single bytes.
  - String literals can use single or double quotes.
  - Single-quoted literals do not interpolate variables.
  - Double-quoted literals interpolate variables and recognize escape sequences.

- **Arithmetic Operators and Expressions:**
  - PHP has standard arithmetic operators like +, *, /, %, ++, and --.
  - Integer division results in a double if not integral.
  - Predefined functions operate on numeric values.

- **Arrays:**
  - PHP arrays are a combination of traditional arrays and associative arrays.
  - Can have integer or string keys.
  - Arrays can be created using assignment or the `array()` construct.

- **Control Statements:**
  - Relational and Boolean operators are similar to JavaScript.
  - Selection statements include if, elseif, else, and switch.
  - Loops include while, for, and do-while.

- **Functions:**
  - Functions are defined using `function function_name([formal_parameters])`.
  - Parameters can be passed by value or reference.
  - Lifetime of a variable in a function is from its appearance to the end of the function's execution.

- **Pattern Matching:**
  - PHP supports string pattern matching using regular expressions.
  - POSIX and Perl-compatible regular expressions are available.
  - Functions like `preg_match` and `preg_split` are used for pattern matching.

- **Form Handling:**
  - Forms can be handled in a separate document specified in the action attribute.
  - PHP provides implicit arrays `$_GET` and `$_POST` for form data handling.
  - Values can be accessed using keys matching form element names.

File Handling in PHP:

- **Opening a File:**
  - Use `fopen(filename, use_indicator)` to open a file.
  - Example: `$file_var = fopen("test.dat", "r") or die("Error - test.dat can't be opened");`
  - Use `file_exists(filename)` to check if a file exists before opening it.
  - Use `fclose(file_var)` to close a file.

- **Reading from a File:**

  1. Read all or part of the file into a string variable using `fread($file_var, #bytes)`.
  2. Read the lines of the file into an array using `file(file_name)`.
  3. Another function `file_get_contents("testdata.dat")` reads the entire contents of the file.
  4. Read one line from the file using `fgets(file_var, #bytes)`.
  5. Read one character at a time using `fgetc(file_var)`.

- **Writing to a File:**
  - Use `fwrite($file_var, $out_data)` to write to a file.
  - `file_put_contents("savedata.dat", $str)` is a function to write the value of the second parameter to the file specified in the first parameter.
  - Files can be locked with `flock(file_var, operation)`.

Cookies in PHP:
- Cookies store information about sessions on the browser.
- Created using `setcookie(cookie_name, cookie_value, lifetime)`.
- Example: `setcookie("voted", "true", time() + 86400)` creates a cookie named "voted" with a one-day lifetime.
- Cookies must be created before any XHTML is generated because they are stored in the HTTP header.
- Retrieve cookies using `$_COOKIES` array.
- Check if a specific cookie exists with `isset($_COOKIE["cookie_name"])`.

Session Tracking:
- An alternative to cookies.
- Uses a session array to store information about client requests during a session.
- Created and maintained by `session_start()`.
- Session variables stored in `$_SESSION` array.
- Example: Counting the number of pages visited.

Database Access with PHP and MySQL:
- Two-tier and three-tier architectures.
- MySQL is a widely used SQL implementation.
- Connect to a database using `mysql_connect()`.
- Select a database using `mysql_select_db("database_name")`.
- Execute queries with `mysql_query(query_string)`.

Potential Problems:
- Special characters in databases and handling them in PHP.
- Magic quotes (`magic_quotes_gpc`) can be turned on or off.
- Issues with SQL commands obtained from user input.

PHP Scripts:
- Rows of query results are PHP arrays.
- Results can be fetched using `mysql_fetch_array`.
- Values and keys can be extracted from result rows.

Combining Files:
- Combining query collection and processing into a single PHP file.

- Using a hidden input element (`<input type="hidden" name="stage" value="1" />`) to distinguish between displaying a form and processing a query.
- PHP code checks the value of the hidden element (`$stage`) to determine the action.

Note for VIVA:
- Understanding file handling in PHP: opening, reading, and writing.
- Working with cookies and session tracking for user customization.
- Database access with PHP and MySQL: connecting, querying, and handling results.
- Awareness of potential issues with special characters and user input.
- Combining files for efficient PHP script management.

1. Associate Arrays in PHP:
  - An associative array in PHP is a collection of key-value pairs.
  - Example: `$person = ['name' => 'John', 'age' => 25];`
  - Access values using keys, e.g., `$name = $person['name'];`

2. Array Methods in PHP:
  - PHP offers various array functions like `count()`, `array_push()`, `array_pop()`, etc.
  - `count($array)` returns the number of elements.
  - `array_push($array, $value)` adds a value to the end of an array.
  - `array_pop($array)` removes and returns the last element.

3. Types of Arrays in PHP:
  - Indexed arrays: Numerically indexed arrays.
  - Associative arrays: Key-value pairs.
  - Multidimensional arrays: Arrays within arrays.

4. Sorting Methods of Arrays in PHP:
  - `sort($array)` sorts an indexed array in ascending order.
  - `asort($array)` preserves key-value associations during sorting.
  - `ksort($array)` sorts an associative array by keys.

5. JS Arrays:
  - Similar to PHP, JavaScript arrays hold multiple values.
  - Declaration: `let fruits = ['apple', 'orange', 'banana'];`
  - Access elements: `let firstFruit = fruits[0];`

6. Scope in PHP and JS:
  - PHP:
    - Variables outside functions are global.
    - Inside functions, use `global` to access global variables.
  - JS:
    - Variables declared with `var` have function scope.
    - Variables declared with `let` and `const` have block scope.

7. CSS Position:
  - CSS `position` property controls an element's positioning.

  - Values include `static`, `relative`, `absolute`, and `fixed`.
  - Example: `position: relative;`

8. jQuery Selectors:
  - jQuery selectors are patterns used to select and manipulate HTML elements.
  - Examples: `$('div')` selects all div elements, `$('#myId')` selects an element by ID.

9. Why Use jQuery:
  - Simplifies JavaScript code and event handling.
  - Cross-browser compatibility.
  - Provides AJAX capabilities with ease.

10. Ready() and Load() in jQuery:
   - `$(document).ready(function(){ ... });` ensures the code runs when the DOM is ready.
   - `$(window).load(function(){ ... });` waits for all assets to be loaded.
   - Used for executing code at specific points during page loading in jQuery.

Introduction to XML:
XML, or eXtensible Markup Language, is a meta-markup language developed in the mid-1990s by the World Wide Web Consortium (W3C). As a meta-markup language, XML is designed to define other markup languages that can describe various types of documents. Its roots can be traced back to the Standard Generalized Markup Language (SGML), a predecessor that was too complex for practical use.

HTML, initially developed using SGML, was designed for structuring web documents. However, it had limitations such as a fixed set of tags and attributes, making it challenging for users to define their own. Additionally, HTML did not impose any restrictions on the arrangement or order of tags, leading to potential issues.

Two Problems with HTML:
1. **Limited Tags and Attributes:** Users couldn't define new tags or attributes, forcing a fixed set to fit all document types.
2. **No Tag Order Restrictions:** Tags could appear in any order, causing potential structure issues.

SGML Challenges:
Despite being powerful, SGML was too large and complex, making it difficult to build parsers. To address these challenges, a lighter version of SGML was needed.

XML - A Solution:
XML, introduced in 1996, aimed to provide a simpler alternative to SGML. It became an ISO standard in 1998 and has since seen various versions. Unlike HTML, XML does not predefine tags, allowing users to define their own markup languages, referred to as "tag sets."

Key Points about XML:
- XML is not a replacement for HTML; it's a meta-markup language for defining other markup languages.
- It serves as a universal way to store and transport data, with a focus on meaning rather than layout.
- XML documents are plain text, easily readable by both humans and applications.
- XML uses an XML processor to parse documents, ensuring consistency across different applications.

XML Syntax:
- XML documents consist of data elements, markup declarations, and processing instructions.
- All XML documents start with an XML declaration specifying the version and encoding.
- XML tags must follow specific naming rules, are case-sensitive, and must be well-formed.

XML Document Structure:
- XML documents have a single root element, and all other elements must be nested inside it.
- Entities, logically related collections of information, help manage large documents efficiently.
- XML entities can be general or parameter entities, allowing for flexible referencing and reuse.

Document Type Definitions (DTDs):
- DTDs provide structural rules for a collection of XML documents, ensuring consistent and uniform structure.
- They define elements, attributes, and entities, allowing users to establish a standard for their data.
- DTDs can be internal or external, with external DTDs preferred for reusability.

Declaring Elements and Attributes:
- Element declarations in DTDs specify the structure of elements and their relationships.
- Attributes are declared separately, providing additional information about elements.
- DTDs use modifiers, choices, and parentheses to define content models.

XML Validation:
- Some XML parsers are validating parsers, checking documents against DTDs for structure conformity.
- Well-formedness is a prerequisite for XML documents, and validating parsers report inconsistencies.

Internal and External DTDs:
- DTDs can be included internally within an XML document or stored externally.
- External DTDs are preferred for reuse across multiple documents.

Namespaces:
- A markup vocabulary consists of element types and attribute names in a markup language.
- XML documents may define their tag sets and use those of other tag sets, leading to potential conflicts.
- XML namespaces are collections of element names in XML documents. They are identified by URIs.
- Namespace declarations have the form `<element_name xmlns[:prefix] = URI>`. The prefix is optional and is a shorthand for the URI.
- The prefix is useful for shortening long URIs and dealing with illegal XML characters.
- A namespace declaration is usually at the root of a document, and the prefix is used to attach names in the XML document.
- Example: `<gmcars xmlns:gm = "http://www.gm.com/names">`, and you can use `<gm:pontiac>` in the document.
- Purposes of prefixes include shorthand and handling illegal characters in URIs.
- Multiple namespaces can be declared on one element, allowing it to use names from different namespaces.

XML Schemas:
- DTDs have drawbacks, including a syntax unrelated to XML and limited control over data types.
- XML schemas, represented by XML documents, offer more control over data types and can be parsed with an XML parser.

- XML schemas resemble class definitions in object-oriented programming, and documents conforming to a schema are instances of that schema.
- Schemas define the structure and data types of XML documents.
- Namespaces in XML schemas are represented by URIs, often starting with the author's website address.
- Schemas are defined in the namespace `http://www.w3.org/2001/XMLSchema`.
- Schemas define namespaces, and the target namespace is specified using `targetNamespace` attribute.
- Schema elements can be global or local, and their visibility is determined by where they are declared.

Defining Schema:
- Schemas are written using names from the `http://www.w3.org/2001/XMLSchema` namespace.
- The schema element has `xmlns:xsd` to specify the schema of schemas.
- Target namespace is specified with `targetNamespace` attribute, and default namespace with `xmlns`.
- `elementFormDefault` determines whether non-top-level elements are in the target namespace.
- Examples:
  - `<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema" targetNamespace = "http://cs.uccs.edu/planeSchema" xmlns = "http://cs.uccs.edu/planeSchema" elementFormDefault = "qualified">`
  - `<schema xmlns = "http://www.w3.org/2001/XMLSchema" targetNamespace ="http://cs.uccs.edu/planeSchema” xmlns:plane = “http://cs.uccs.edu/planeSchema” elementFormDefault = "qualified">`

Defining Schema Instances:
- XML instances must specify namespaces and schema location.
- `xmlns` in the root element defines the default namespace.
- `xmlns:xsi` specifies the XMLSchema-instance namespace.
- `xsi:schemaLocation` specifies the schema's namespace and filename.

Overview of Data Types:
- Two categories of user-defined XML schema data types: simple and complex.
- Simple types are restricted to strings, while complex types can have attributes and include other types.
- XML schema defines 44 data types, and user-defined types are derived from existing types.

Validating Instances of Schemas:
- Schema validation tools, such as xsv, determine if an XML instance conforms to a schema.
- Tools can be used online or downloaded.

Displaying XML Documents:
- XML browsers should have default style sheets for raw XML documents.
- CSS or XSLT style sheets can be used for styling XML documents.
- CSS style sheets list element names with associated attributes.
- Processing instruction `<?xml-stylesheet?>` connects XML documents to CSS style sheets.

Note for VIVA Purpose:
- XML namespaces are used to avoid conflicts in XML documents with different tag sets.
- XML schemas provide more control over data types compared to DTDs.
- Schemas define the structure and data types of XML documents.
- Namespaces in schemas are represented by URIs, often starting with the author's website address.

- Schema instances must specify namespaces and schema location for validation.
- Tools like xsv can validate XML instances against schemas.
- CSS style sheets can be used for styling XML documents.

eXtensible Stylesheet Language (XSL):
- XSL began as a standard for presentations of XML documents and was split into three parts: XSL (XSL-FO) for formatting objects, XSLT for transformations, and XPath for identifying nodes in XML documents.
- XSLT style sheets are XML documents that can be validated against DTDs. They are used to transform XML documents into different forms or formats.
- XPath is a language for expressions used to identify parts of XML documents, such as specific elements or elements with particular attribute values.
- XSLT is a functional style programming language with functions, parameters, names, selection, and conditional expressions.
- An XSLT processor takes an XML document and an XSLT document as input, where the XSLT document is the program to be executed. The output is a new document called the XSL document.
- XSLT style sheets consist primarily of templates, which use XPath to describe element/attribute patterns in the input XML document. The template-driven model is the primary processing model.
- XSLT style sheets can specify page layout, orientation, writing direction, margins, page numbering, etc.
- The processing instruction `<?xml-stylesheet type="text/xsl" href="XSLT_style_sheet"?>` is used to connect an XSLT style sheet to an XML document.
- An XSLT style sheet is an XML document with a root element `stylesheet` that defines namespaces.

XML Processors:
- XML processors have purposes like checking syntax for well-formedness, replacing entity references, copying default values into the document, and validating document structure if a DTD or schema is specified.
- Two ways to check well-formedness: using a browser with an XML parser or a stand-alone XML parser.
- XML processors can use two standard APIs: DOM (Document Object Model) and SAX (Simple API for XML).
- DOM represents XML documents as objects in memory and provides random access to the document. SAX processes XML documents sequentially and is event-driven, raising events for each syntactic structure.
- DOM is advantageous for multiple accesses, rearrangement of documents, and random access to document parts. SAX is advantageous for large documents and is faster.

Web Services:
- Web services allow different software components in different places, written in different languages and on different platforms, to connect and interoperate.
- Web services have three roles: service providers, service requestors, and a service registry.
- WSDL (Web Services Description Language) is used to describe available services and their message protocols. UDDI (Universal Description, Discovery, and Integration) is used to create a Web services registry.
- SOAP (Simple Object Access Protocol) is an XML-based specification defining forms of messages and RPCs for exchanging information among distributed systems.
- RESTful Web services, gaining popularity, use HTTP methods like PUT, GET, DELETE, and POST. They do not require XML messages or WSDL definitions and are often better integrated with HTTP and web browsers.