MODULE 5: Web Extension: (PHP & XML)

PHP

Q.1) Explain the different data types of PHP.

⇒

PHP, a popular server-side scripting language, supports a variety of data types that enable developers to work with different kinds of data efficiently. Understanding these data types is essential for effective programming in PHP. Below is a detailed explanation of the different data types in PHP:

1. String

- **Definition**: A string is a sequence of characters enclosed in quotes. Strings can be defined using single quotes (`' '`) or double quotes (`" "`).
- **Characteristics**:
    - Supports escaping special characters using backslashes (e.g., `\\'`, `\\"`, `\\\\`, etc.).

    - Strings can be concatenated using the dot (`.`) operator.
- **Example**:

```
$string1 = "Hello, World!";
$string2 = 'PHP is awesome!';
$combinedString = $string1 . ' ' . $string2; // Output: "Hello, World! PHP is awesome!"
```

2. **Integer**

- **Definition**: An integer is a whole number, which can be positive, negative, or zero, without any decimal point.
- **Characteristics**:
    - Can be specified in decimal, hexadecimal (preceded by `0x`), or octal (preceded by `0`).
    - The size of integers is platform-dependent, typically 32 or 64 bits.
- **Example**:

```
$age = 30; // Decimal
$hexadecimal = 0x1E; // Hexadecimal (30 in decimal)
$octal = 036; // Octal (30 in decimal)
```

3. **Float (or Double)**

- **Definition**: A float (or double) is a number that includes a decimal point or is expressed in exponential notation.
- **Characteristics**:
    - Used for precise calculations and representing non-integer numbers.
    - Can handle large numbers due to floating-point representation.
- **Example**:

```
$price = 19.99; // Float
$scientific = 1.5e3; // 1500 in scientific notation
```

4. **Boolean**

- **Definition**: A boolean represents a truth value and can either be `true` or `false`.
- **Characteristics**:
    - Often used in control structures and logical expressions.
    - PHP treats various values as truthy or falsy when evaluating conditions.
- **Example**:

```
$isAvailable = true; // Boolean true
$isSoldOut = false; // Boolean false
```

5. **Array**

- **Definition**: An array is a data structure that can hold multiple values under a single variable name, allowing for indexed or associative arrays.
- **Characteristics**:
    - Indexed arrays are numerically indexed, while associative arrays use named keys.
    - Can hold mixed data types.
- **Example**:

```
// Indexed Array
$colors = array("red", "green", "blue"); // or $colors = ["red", "green",
"blue"];

// Associative Array
$person = array("name" ⇒ "John", "age" ⇒ 30); // or $person = ["nam
e" ⇒ "John", "age" ⇒ 30];
```

6. **Object**

- **Definition**: An object is an instance of a class and can contain properties (variables) and methods (functions) that define the behavior of the object.
- **Characteristics**:

    - Encapsulates data and functions together, promoting object-oriented programming principles.

- **Example**:

```
class Car {
    public $color;

    public function setColor($color) {
        $this→color = $color;
    }

    public function getColor() {
        return $this→color;
    }
}

$myCar = new Car();
$myCar→setColor("blue");
echo $myCar→getColor(); // Output: blue
```

7. NULL

- **Definition**: NULL is a special data type that represents a variable with no value assigned to it.
- **Characteristics**:
    - It is a type itself, and a variable can be assigned NULL explicitly.
    - Used to check the absence of a value or to unassign a variable.
- **Example**:

```
$var = NULL; // Explicitly setting a variable to NULL
$emptyVar; // This will also be NULL since it's not assigned a value
```

8. Resource

- **Definition**: A resource is a special variable that holds a reference to an external resource, such as a database connection or a file handle.

- **Characteristics**:
    - Resources are not a data type in the traditional sense but represent a special kind of reference to external entities.
- **Example**:

```
$conn = mysqli_connect("localhost", "username", "password", "databa
se");
// $conn is a resource representing a connection to the MySQL databas
e
```

Summary of PHP Data Types

- **Scalability**: PHP can handle various data types, allowing developers to choose the most suitable one for their needs.
- **Dynamic Typing**: PHP is dynamically typed, meaning the data type of a variable is determined at runtime, allowing flexibility in variable assignment.
- **Type Juggling**: PHP can automatically convert data types based on the context, e.g., using an integer in a string context.

Conclusion

Understanding the different data types in PHP is fundamental to writing efficient and effective code. Each data type serves a specific purpose, allowing developers to manage and manipulate data according to their application's needs. By leveraging these data types, PHP developers can build robust and dynamic web applications.

This comprehensive explanation covers the different data types in PHP and provides examples to illustrate their usage, suitable for a detailed discussion on the topic.

Q.2) discuss about various control structures used in PHP. Give suitable example for each.

⇒

In PHP, control structures allow developers to dictate the flow of execution of the program. These structures enable you to create conditional logic, loops, and branching, which are essential for building dynamic applications. This answer will discuss the various control structures in PHP, providing examples for each.

1. Conditional Statements

a. `if` Statement

The `if` statement is used to execute a block of code only if a specified condition is true.

Syntax:

```
if (condition) {
    // Code to be executed if condition is true
}
```

Example:

```
$age = 20;
if ($age >= 18) {
    echo "You are eligible to vote.";
}
```

Output: You are eligible to vote.

b. `if...else` Statement

The `if...else` statement provides an alternative block of code to execute if the condition is false.

Syntax:

```
if (condition) {
    // Code to be executed if condition is true
} else {
    // Code to be executed if condition is false
}
```

Example:

```
$age = 16;
if ($age >= 18) {
    echo "You are eligible to vote.";
} else {
    echo "You are not eligible to vote.";
}
```

Output: You are not eligible to vote.

c. `else if` Statement

The `else if` statement allows for multiple conditions to be checked.

Syntax:

```
if (condition1) {
    // Code to be executed if condition1 is true
} else if (condition2) {
    // Code to be executed if condition2 is true
} else {
    // Code to be executed if both conditions are false
}
```

Example:

```
$score = 85;
if ($score >= 90) {
    echo "Grade: A";
} else if ($score >= 80) {
    echo "Grade: B";
} else {
    echo "Grade: C";
}
```

Output: Grade: B

d. `switch` Statement

The `switch` statement allows a variable to be tested for equality against a list of values (cases).

Syntax:

```
switch (variable) {
    case value1:
        // Code to be executed if variable equals value1
        break;
    case value2:
        // Code to be executed if variable equals value2
        break;
    default:
        // Code to be executed if variable doesn't match any case
}
```

Example:

```
$day = 3;
switch ($day) {
    case 1:
        echo "Monday";
        break;
    case 2:
        echo "Tuesday";
        break;
    case 3:
        echo "Wednesday";
        break;
    default:
        echo "Invalid day";
}
```

Output: Wednesday

2. **Loops**

Loops are used to execute a block of code repeatedly until a certain condition is met.

a. `while` Loop

The `while` loop continues executing as long as the specified condition is true.

Syntax:

```
while (condition) {
    // Code to be executed
}
```

Example:

```
$count = 1;
while ($count <= 5) {
    echo $count . " ";
    $count++;
}
```

Output: 1 2 3 4 5

b. `do...while` Loop

The `do...while` loop is similar to the `while` loop, but it ensures that the block of code is executed at least once before checking the condition.

Syntax:

```
do {
    // Code to be executed
} while (condition);
```

Example:

```
$count = 1;
do {
    echo $count . " ";
    $count++;
} while ($count <= 5);
```

Output: 1 2 3 4 5

c. `for` Loop

The `for` loop is used when the number of iterations is known beforehand. It consists of three parts: initialization, condition, and increment/decrement.

Syntax:

```
for (initialization; condition; increment/decrement) {
    // Code to be executed
}
```

Example:

```
for ($i = 1; $i <= 5; $i++) {
    echo $i . " ";
}
```

Output: 1 2 3 4 5

d. `foreach` Loop

The `foreach` loop is specifically designed for iterating over arrays.

Syntax:

```
foreach ($array as $value) {
    // Code to be executed
}
```

Example:

```
$colors = array("red", "green", "blue");
foreach ($colors as $color) {
    echo $color . " ";
}
```

Output: red green blue

3. Break and Continue Statements

These statements alter the flow of loops.

a. `break`

The `break` statement is used to terminate a loop early.

Example:

```
for ($i = 1; $i <= 10; $i++) {
    if ($i == 5) {
        break; // Exit the loop when i is 5
    }
    echo $i . " ";
}
```

Output: 1 2 3 4

b. `continue`

The `continue` statement skips the current iteration and continues with the next iteration of the loop.

Example:

```
for ($i = 1; $i <= 5; $i++) {
    if ($i == 3) {
        continue; // Skip the iteration when i is 3
    }
    echo $i . " ";
}
```

Output: 1 2 4 5

4. Ternary Operator

The ternary operator is a shorthand for `if...else` statements. It takes three operands.

Syntax:

```
(condition) ? value_if_true : value_if_false;
```

Example:

```
$age = 18;
$message = ($age >= 18) ? "Eligible to vote" : "Not eligible to vote";
echo $message;
```

Output: Eligible to vote

Conclusion

Control structures in PHP are essential for creating dynamic and responsive applications. They allow for conditional execution of code and repeated execution of code blocks, enabling developers to manage the flow of the program effectively. Understanding and utilizing these control structures is crucial for any PHP developer, as they form the foundation of logic in programming. This discussion covered various control structures, including conditional statements, loops, and the use of break and continue statements, along with practical examples to illustrate their usage.

Q.3) if you wanted to send a sensitive information like password to the backend, which among the get and the post method in PHP, would you use? Justify your answer and distinguish between these methods?

⇒

When sending sensitive information such as passwords to the backend in PHP, it is advisable to use the **POST** method instead of the **GET** method. Below is an explanation of why the POST method is preferred, along with a detailed distinction between the two methods.

Why Use POST for Sensitive Information

1. **Data Visibility**:

    - **GET**: Parameters are appended to the URL, making them visible in the browser's address bar. For example, if a password is sent via GET, it would look like this: `example.com/login.php?username=user&password=pass123`. This makes sensitive data easily accessible to anyone who can see the URL, including through browser history, server logs, and network monitoring tools.

    - **POST**: Data is sent in the body of the request, not the URL, which keeps it hidden from casual observation. The URL remains clean, and sensitive data such as passwords is not exposed.

2. **Data Length Limitations**:

- **GET**: The amount of data that can be sent is limited due to URL length restrictions (usually around 2048 characters, but this can vary by browser). This makes it unsuitable for larger payloads, such as lengthy passwords or multiple fields of sensitive information.
- **POST**: There is no significant limit on the size of the data sent, making it suitable for sending more extensive information.

3. **Caching**:
   - **GET**: Requests made using GET can be cached by browsers and intermediaries, which can lead to security issues, especially when sensitive information is stored in caches.
   - **POST**: Typically, POST requests are not cached, reducing the risk of sensitive data being stored inadvertently.
4. **Idempotency**:
   - **GET**: This method is idempotent, meaning that repeated identical requests will have the same effect as a single request. This can lead to issues when trying to submit sensitive information multiple times, such as duplicate logins.
   - **POST**: This method is not idempotent, which means that repeated submissions will have different effects (e.g., creating multiple records), making it a better choice for operations that change server state, such as logging in.

Q.4) explain how form validation works in PHP with suitable example.

⇒

Form validation in PHP is essential for ensuring that user inputs are both accurate and secure before they are processed by the server. Validation can include checking for required fields, ensuring that data is in the correct format (like email addresses), and sanitizing inputs to prevent security issues like SQL injection and XSS (Cross-Site Scripting).

Steps for Form Validation in PHP

1. **Create the HTML Form**: Start by creating a form in HTML that collects user inputs.
2. **Check Form Submission**: In your PHP script, check if the form has been submitted.
3. **Validate Inputs**: Validate each field according to your criteria (e.g., required fields, valid email format).
4. **Display Errors**: If validation fails, display error messages to the user.
5. **Process the Data**: If validation succeeds, process the data (e.g., save to a database, send an email).

Example of Form Validation in PHP

Let's create a simple example that validates a user registration form with the following fields: username, email, and password.

HTML Form

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Registration Form</title>
</head>
<body>
    <h2>Registration Form</h2>
    <form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?
>" method="post">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username" required>
        <span class="error"><?php echo $usernameErr; ?></span>
        <br>

        <label for="email">Email:</label>
        <input type="text" id="email" name="email" required>
        <span class="error"><?php echo $emailErr; ?></span>
        <br>

        <label for="password">Password:</label>
        <input type="password" id="password" name="password" required>
        <span class="error"><?php echo $passwordErr; ?></span>
        <br>

        <input type="submit" value="Register">
    </form>
</body>
</html>
```

PHP Script for Validation

```
<?php
// Initialize error variables
$usernameErr = $emailErr = $passwordErr = "";
$username = $email = $password = "";

// Validate form on submission
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Validate username
    if (empty($_POST["username"])) {
        $usernameErr = "Username is required";
    } else {
        $username = sanitize_input($_POST["username"]);
    }

    // Validate email
    if (empty($_POST["email"])) {
        $emailErr = "Email is required";
    } else {
        $email = sanitize_input($_POST["email"]);
        // Check if email format is valid
        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $emailErr = "Invalid email format";
        }
    }

    // Validate password
    if (empty($_POST["password"])) {
        $passwordErr = "Password is required";
    } else {
        $password = sanitize_input($_POST["password"]);
        // Check password length
        if (strlen($password) < 6) {
            $passwordErr = "Password must be at least 6 characters";
        }
    }

    // If no errors, process the data
    if (empty($usernameErr) && empty($emailErr) && empty($passwordErr))
{
        // Code to save the data to the database or perform other actions
        echo "Registration successful!";
    }
}

// Function to sanitize user input
function sanitize_input($data) {
    return htmlspecialchars(stripslashes(trim($data)));
}
?>
```

// (for our understanding) Explanation of the Code

1. **Form Creation**: The HTML form is created with fields for username, email, and password. Each field has a corresponding error message span to display validation errors.
2. **Form Submission Check**: The PHP script checks if the form has been submitted using the `$_SERVER["REQUEST_METHOD"]` variable.
3. **Validation Logic**:
    - For each field, the script checks if it is empty. If it is, an appropriate error message is set.
    - For the email field, it uses `filter_var()` to check if the email format is valid.
    - For the password field, it checks if the password length is at least six characters.

4. **Sanitization**: The `sanitize_input()` function is used to clean user input by removing unnecessary characters (like extra spaces), escaping special characters, and preventing XSS attacks.
5. **Data Processing**: If there are no validation errors, the script proceeds to process the data (e.g., save to a database). //

Q.5) explain the features of PHP and write a PHP program to print the factorial of number.

⇒

Features of PHP:

1. **Open Source**: PHP is free to use, and developers can modify it as per their requirements.
2. **Cross-Platform**: PHP runs on various platforms (Windows, Linux, macOS), allowing developers to build applications that work across different environments.
3. **Simplicity**: PHP is easy to learn and use for beginners. Its syntax is similar to C and Java, making it familiar for many developers.
4. **Speed**: PHP scripts execute quickly, enhancing web application performance.
5. **Database Support**: PHP supports multiple databases (MySQL, PostgreSQL, SQLite), allowing seamless interaction with various data sources.
6. **Error Reporting**: PHP has robust error reporting features, which help in debugging and identifying issues during development.
7. **Large Community**: A vast community of PHP developers provides support, tutorials, and frameworks (like Laravel, Symfony) to enhance development efficiency.
8. **Integration**: PHP easily integrates with various web technologies (HTML, CSS, JavaScript) and web services (RESTful APIs).
9. **Session Management**: PHP provides built-in support for session management, allowing developers to maintain user states across requests.
10. **File Handling**: PHP supports various file operations, enabling file creation, reading, and writing directly from the server.

PHP Program to Print the Factorial of a Number:

```
<?php
// Function to calculate factorial
function factorial($n) {
    if ($n < 0) {
        return "Factorial is not defined for negative numbers.";
    } elseif ($n == 0 || $n == 1) {
        return 1; // Base case
    } else {
        return $n * factorial($n - 1); // Recursive call
    }
}

// Example: Get user input
$number = 5; // Change this number to test other values
$result = factorial($number);

// Print the result
echo "The factorial of $number is $result.";
?>
```

// Explanation of the Code:

- The function `factorial($n)` computes the factorial of a given number using recursion.
- Base cases handle negative numbers (returns a message) and 0 or 1 (returns 1).
- For other positive integers, the function calls itself to calculate the factorial.
- The output is displayed using the `echo` statement. //

Q.6) how to define variable in PHP and explain scope of variable.

⇒

Defining Variables in PHP:

- Variables in PHP are defined using the dollar sign (`$`) followed by the variable name.
- Variable names must start with a letter or underscore, followed by any number of letters, numbers, or underscores.

Example of Variable Definition:

```
<?php
$variableName = "Hello, PHP!"; // Defining a variable
$number = 42; // Integer variable
$floatNumber = 3.14; // Float variable
?>
```

Scope of Variables:
The scope of a variable determines where the variable can be accessed within the code. PHP has four main types of variable scope:

1. **Global Scope**:
    - Variables declared outside any function are in the global scope.
    - They can be accessed from anywhere outside functions but not inside functions unless declared as global.
    - Example:

    ```
    <?php
    $globalVar = "I'm a global variable";

    function showGlobalVar() {
        global $globalVar; // Accessing the global variable
        echo $globalVar;
    }

    showGlobalVar(); // Output: I'm a global variable
    ?>
    ```

2. **Local Scope**:
    - Variables declared within a function are local to that function.
    - They can only be accessed within the function where they are defined.

- Example:

```
<?php
function localScope() {
    $localVar = "I'm a local variable";
    echo $localVar; // Accessible here
}

localScope(); // Output: I'm a local variable
// echo $localVar; // This will cause an error (undefined variable)
?>
```

3. **Static Scope**:
    - A static variable maintains its value even after the function has ended.
    - It is declared with the `static` keyword.
    - Example:

```
<?php
function staticExample() {
    static $count = 0; // Static variable
    $count++;
    echo $count;
}

staticExample(); // Output: 1
staticExample(); // Output: 2
staticExample(); // Output: 3
?>
```

4. **Function Parameter Scope**:
    - Variables defined as parameters in a function are local to that function and can be accessed only within it.
    - Example:

```
<?php
function parameterScope($param) {
    echo $param; // Accessible here
}

parameterScope("I'm a parameter!"); // Output: I'm a parameter!
// echo $param; // This will cause an error (undefined variable)
?>
```

Summary

- **Defining Variables**: Use `$` followed by the name.
- **Variable Scope**: Determines accessibility and can be global, local, static, or function parameter-based.
- Understanding variable scope is crucial for managing data and avoiding conflicts in larger applications.

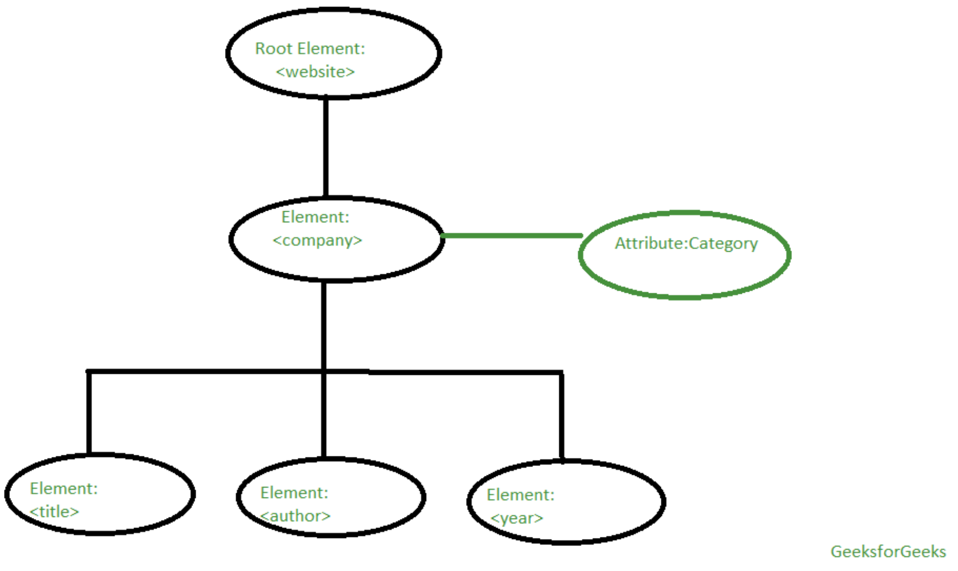
XML

Q.7) explain the structure of XML documents with example.

⇒

XML (eXtensible Markup Language) is a markup language that defines rules for encoding documents in a format that is both human-readable and machine-readable.

XML documents are structured hierarchically and contain various components that work together to represent data.



Key Components of XML Structure

1. **Prolog**:
    - The prolog is the first line of an XML document and declares the XML version and the character encoding used in the document.
    - Example:

```
<?xml version="1.0" encoding="UTF-8"?>
```

2. **Root Element**:
    - An XML document must have a single root element that contains all other elements. This root element defines the overall structure of the XML document.
    - Example:

```
<library>
   <!-- Other elements will be nested here -->
</library>
```

3. **Elements**:
    - Elements are the building blocks of an XML document. They are defined by opening and closing tags and can contain text, attributes, and other nested elements.

- Example:

```xml
<book>
   <title>XML Fundamentals</title>
   <author>John Doe</author>
   <publisher>Tech Press</publisher>
   <year>2021</year>
</book>
```

4. **Attributes**:

   - Attributes provide additional information about elements. They are defined within the opening tag of an element.
   - Example:

```xml
<book id="b1">
   <title>XML Fundamentals</title>
   <author>John Doe</author>
   <publisher>Tech Press</publisher>
   <year>2021</year>
</book>
```

5. **Comments**:

   - Comments can be added to an XML document for clarification and are not processed by XML parsers.
   - Example:

```xml
<!-- This is a comment →
```

6. **Example of a Well-Formed XML Document**:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<library>
   <book id="b1">
      <title>XML Fundamentals</title>
      <author>John Doe</author>
      <publisher>Tech Press</publisher>
```

```xml
        <year>2021</year>
    </book>
    <book id="b2">
        <title>Learning XSLT</title>
        <author>Jane Smith</author>
        <publisher>Code Books</publisher>
        <year>2020</year>
    </book>
</library>
```

Q.8) What are benefits of using JSON over XML data? / diffenentiate between JSON and XML —[all paper for 5 or 10 marks]

⇒

Benefits of Using JSON Over XML Data

1. **Lightweight**:
    - JSON is more compact than XML, resulting in smaller data sizes and faster transmission over networks.
    - Example:
        - JSON:

          ```
          { "name": "John", "age": 30 }
          ```

        - XML:

          ```xml
          <person>
              <name>John</name>
              <age>30</age>
          </person>
          ```

2. **Easier to Read and Write**:
    - JSON syntax is simpler and more intuitive compared to XML, making it easier for developers to read and write.

- Example: JSON uses curly braces `{}` and square brackets `[]` for objects and arrays, respectively.

3. **Data Types**:
   - JSON supports a variety of data types (strings, numbers, arrays, booleans, null), whereas XML primarily represents data as strings.
   - Example:
     - JSON:

       ```
       { "price": 19.99, "inStock": true }
       ```

     - XML:

       ```
       <price>19.99</price>
       <inStock>true</inStock>
       ```

4. **Integration with JavaScript**:
   - JSON is natively supported by JavaScript, making it easier to work with in web applications. XML requires parsing to be used in JavaScript.
   - Example:
     - Parsing JSON:

       ```
       const data = JSON.parse('{"name": "John", "age": 30}');
       ```

5. **No Closing Tags**:
   - JSON does not require closing tags for elements, reducing verbosity.
   - Example:
     - JSON:

       ```
       { "name": "John" }
       ```

     - XML:

       ```
       <name>John</name>
       ```

Differences Between JSON and XML (In Tabular Form)

Feature	JSON	XML
Syntax	Lightweight and easy to read	Verbose and more complex
Data Representation	Uses key-value pairs	Uses nested elements and attributes
Data Types	Supports various data types	Primarily represents data as strings
Readability	Easier to read and write	More difficult to read due to verbosity
Natively Supported	Directly supported in JavaScript	Requires parsing in JavaScript
Structure	Curly braces `{}` and square brackets `[]`	Uses opening and closing tags
Closing Tags	Does not require closing tags	Requires closing tags for elements
Comments	No support for comments	Supports comments
Parsing	Directly parsed by JavaScript	Requires an XML parser

Q.9) What is DTD? Explain the internal DTD and external DTD?

⇒

DTD (Document Type Definition) is a set of markup declarations that define a document structure in XML.

It specifies the elements and attributes that can appear in a document, their relationships, and the order in which they can occur.

DTDs help ensure that XML documents are well-formed and valid according to specified rules.

Types of DTD

1. **Internal DTD**:

    - Internal DTDs are defined within the XML document itself. They are declared in the prolog section of the XML file.

- Example:

```
<?xml version="1.0"?>
<!DOCTYPE library [
  <!ELEMENT library (book+)>
  <!ELEMENT book (title, author, publisher, year)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT author (#PCDATA)>
  <!ELEMENT publisher (#PCDATA)>
  <!ELEMENT year (#PCDATA)>
]>
<library>
  <book>
    <title>XML Fundamentals</title>
    <author>John Doe</author>
    <publisher>Tech Press</publisher>
    <year>2021</year>
  </book>
</library>
```

2. **External DTD**:

- External DTDs are defined in a separate file. The XML document refers to the external DTD file in its prolog.
- Example:

```
<?xml version="1.0"?>
<!DOCTYPE library SYSTEM "library.dtd">
<library>
  <book>
    <title>XML Fundamentals</title>
    <author>John Doe</author>
    <publisher>Tech Press</publisher>
    <year>2021</year>
  </book>
</library>
```

- The corresponding `library.dtd` file would contain:

```
<!ELEMENT library (book+)>
<!ELEMENT book (title, author, publisher, year)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT year (#PCDATA)>
```

CODE

Q.10) create a well formed XML document to maintain library catalogue. It should contain the name of the book, author, publisher and year of publishing, format It in tabular manner using XSLT.

⇒

Q: Well-formed XML Document for Library Catalogue

1. XML Document

Below is an example of a well-formed XML document representing a library catalogue:

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
   <book>
      <name>The Great Gatsby</name>
      <author>F. Scott Fitzgerald</author>
      <publisher>Charles Scribner's Sons</publisher>
      <year>1925</year>
   </book>
   <book>
      <name>To Kill a Mockingbird</name>
      <author>Harper Lee</author>
```

```
        <publisher>J.B. Lippincott & Co.</publisher>
        <year>1960</year>
    </book>
    <book>
        <name>1984</name>
        <author>George Orwell</author>
        <publisher>Secker & Warburg</publisher>
        <year>1949</year>
    </book>
</library>
```

2. XSLT to Format as a Table

Here's an XSLT stylesheet to format the XML document in a tabular manner:

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

    <xsl:template match="/library">
        <html>
        <head>
            <title>Library Catalogue</title>
            <style>
                table {
                    width: 100%;
                    border-collapse: collapse;
                }
                table, th, td {
                    border: 1px solid black;
                }
                th, td {
                    padding: 8px;
                    text-align: left;
                }
            </style>
        </head>
        <body>
            <h2>Library Catalogue</h2>
        <table>
          <tr>
            <th>Name</th>
            <th>Author</th>
            <th>Publisher</th>
            <th>Year</th>
          </tr>
          <xsl:for-each select="book">
            <tr>
              <td><xsl:value-of select="name"/></td>
              <td><xsl:value-of select="author"/></td>
              <td><xsl:value-of select="publisher"/></td>
              <td><xsl:value-of select="year"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
      </html>
    </xsl:template>
  </xsl:stylesheet>
```

Q.11) an e-commerce website would like to accept the below mentioned data and would like to transfer the data using XML.

i) Product ID

ii) Product Name

iii) Product Cost

iv) Purchase Date

v) Purchased by

vi) Seller Name.

write the HTML, XML Code & DTD for given data

⇒

1. XML Document

Here's an example XML document for the e-commerce data:

2. DTD (Document Type Definition)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<products>
   <product>
      <productID>P001</productID>
      <productName>Wireless Mouse</productName>
      <productCost>25.99</productCost>
      <purchaseDate>2024-10-30</purchaseDate>
      <purchasedBy>John Doe</purchasedBy>
      <sellerName>Tech Store</sellerName>
   </product>
   <product>
      <productID>P002</productID>
      <productName>Mechanical Keyboard</productName>
      <productCost>59.99</productCost>
      <purchaseDate>2024-10-28</purchaseDate>
      <purchasedBy>Jane Smith</purchasedBy>
      <sellerName>Gadget World</sellerName>
   </product>
</products>
```

Here's a DTD that defines the structure of the XML document:

```
<!DOCTYPE products [
   <!ELEMENT products (product+)>
   <!ELEMENT product (productID, productName, productCost, purchaseDa
te, purchasedBy, sellerName)>
   <!ELEMENT productID (#PCDATA)>
   <!ELEMENT productName (#PCDATA)>
   <!ELEMENT productCost (#PCDATA)>
   <!ELEMENT purchaseDate (#PCDATA)>
   <!ELEMENT purchasedBy (#PCDATA)>
   <!ELEMENT sellerName (#PCDATA)>
]>
```

3. HTML Document to Display XML Data

Here’s a simple HTML document that can display the XML data, assuming you are using JavaScript to fetch and display the data:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.
0">
   <title>E-commerce Product List</title>
   <style>
      table {
         width: 100%;
         border-collapse: collapse;
      }
      table, th, td {
         border: 1px solid black;
      }
      th, td {
         padding: 8px;
         text-align: left;
      }
   </style>
</head>
<body>
   <h2>E-commerce Product List</h2>
   <table>
      <tr>
         <th>Product ID</th>
         <th>Product Name</th>
         <th>Product Cost</th>
         <th>Purchase Date</th>
         <th>Purchased By</th>
         <th>Seller Name</th>
      </tr>
      <tbody id="productTableBody">
         <!-- Product data will be inserted here -->
      </tbody>
```

```
    </table>

    <script>
      // Fetching the XML data and displaying it in the table
      const xmlData = `<?xml version="1.0" encoding="UTF-8"?>
      <products>
        <product>
          <productID>P001</productID>
          <productName>Wireless Mouse</productName>
          <productCost>25.99</productCost>
          <purchaseDate>2024-10-30</purchaseDate>
          <purchasedBy>John Doe</purchasedBy>
          <sellerName>Tech Store</sellerName>
        </product>
        <product>
          <productID>P002</productID>
          <productName>Mechanical Keyboard</productName>
          <productCost>59.99</productCost>
          <purchaseDate>2024-10-28</purchaseDate>
          <purchasedBy>Jane Smith</purchasedBy>
          <sellerName>Gadget World</sellerName>
        </product>
      </products>`;

      const parser = new DOMParser();
      const xmlDoc = parser.parseFromString(xmlData, "text/xml");
      const products = xmlDoc.getElementsByTagName("product");
      const tableBody = document.getElementById("productTableBody");

      for (let i = 0; i < products.length; i++) {
        const productID = products[i].getElementsByTagName("productID")
[0].textContent;
        const productName = products[i].getElementsByTagName("product
Name")[0].textContent;
        const productCost = products[i].getElementsByTagName("productC
ost")[0].textContent;
        const purchaseDate = products[i].getElementsByTagName("purcha
seDate")[0].textContent;
```

```
            const purchasedBy = products[i].getElementsByTagName("purchas
edBy")[0].textContent;
            const sellerName = products[i].getElementsByTagName("sellerNam
e")[0].textContent;

            const row = `<tr>
                <td>${productID}</td>
                <td>${productName}</td>
                <td>${productCost}</td>
                <td>${purchaseDate}</td>
                <td>${purchasedBy}</td>
                <td>${sellerName}</td>
            </tr>`;
            tableBody.innerHTML += row;
        }
    </script>
</body>
</html>
```