Module 5

Web Extension PHP and XML

Introduction to XML

XML stands for Extensible Markup Language. XML is a software- and hardware-independent tool for storing and transporting data.

What is XML?

- XML stands for eXtensible Markup Language
- XML is a markup language much like HTML
- XML was designed to store and transport data
- XML was designed to be self-descriptive
- XML is a W3C Recommendation

XML (eXtensible Markup Language) is **not** a programming language. Instead, it is a **markup language** designed to store and transport data. Here's why it's different:

- **Markup Language**: XML uses tags to structure and describe the data. These tags don't perform actions like a programming language does (e.g., loops, conditions, or functions); they merely provide a framework to organize data.
- **Self-descriptive**: XML is often used to make the data readable both by humans and machines. It describes the data using nested elements (tags).
- **No Logic or Algorithms**: Unlike programming languages like Python or Java, XML doesn't have features for executing code, making decisions, or manipulating data at runtime.

XML Does Not DO Anything

Maybe it is a little hard to understand, but XML does not DO anything.

This note is a note to Ram from Jani, stored as XML:

```
<note>
  <to>Ram</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The XML above is quite self-descriptive:

- It has sender information
- It has receiver information
- It has a heading
- It has a message body

But still, the XML above does not DO anything. XML is just information wrapped in tags.

Difference between HTML and XML

HTML	XML
It was written in 1993.	It was released in 1996.
HTML stands for Hyper Text Markup Language.	XML stands for Extensible Markup Language.
HTML is static in nature.	XML is dynamic in nature.
It was developed by WHATWG.	It was developed by Worldwide Web Consortium.
It is termed as a presentation language.	It is neither termed as a presentation nor a programming language.
It has an extension of .html	It has an extension of .xml
HTML is not Case sensitive.	XML is Case sensitive.
HTML tags are predefined tags.	XML tags are user-defined tags.
There are limited number of tags in HTML.	XML tags are extensible.
HTML does not preserve white spaces.	White space can be preserved in XML.

HTML	XML
HTML tags are used for displaying the data.	XML tags are used for carry the data not for displaying.
In HTML, closing tags are not necessary.	In XML, closing tags are necessary.
HTML is used to display the data.	XML is used to store data.

XML Tree Structure

An XML document has a self descriptive structure. It forms a tree structure which is referred as an XML tree. The tree structure makes easy to describe an XML document.

A tree structure contains root element (as parent), child element and so on. It is very easy to traverse all succeeding branches and sub-branches and leaf nodes starting from the root.

Syntax of tree structure

```
<root>
  <parent>
      <child1>Content 1</child1>
      <child2>
          <subchild>Content 2</subchild>
       </child2>
   </parent>
</root>
```

Example of an XML document

```
    <?xml version="1.0"?>
    <college>
      <student>
        <firstname>Ram</firstname>
        <lastname>Patil</lastname>
        <contact>09990449935</contact>
        <email>rajp@gmail.com</email>
        <address>
           <city>Mumbai</city>
```

```
         <state>Maharashtra</state>
         <pin>201007</pin>
      </address>
   </student>
</college>
```

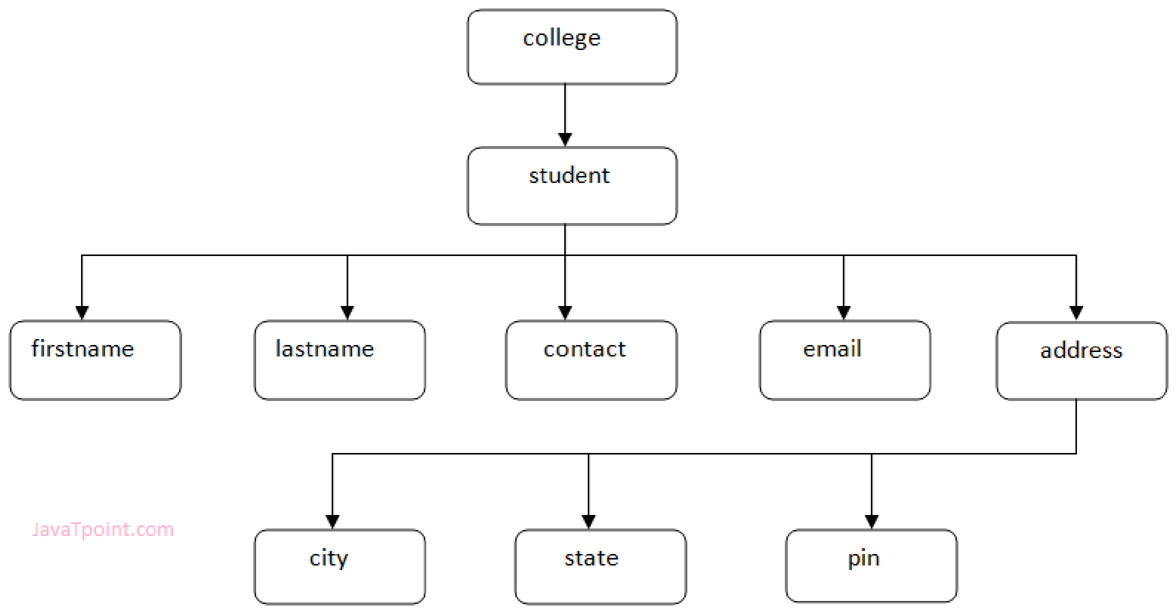


Fig : Example of XML Document

In the above example, first line is the XML declaration. It defines the XML version 1.0. Next line shows the root element (college) of the document. Inside that there is one more element (student). Student element contains five branches named <firstname>, <lastname>, <contact>, <Email> and <address>.<address> branch contain 3 sub-branches named <city>,<state> and <pin>.

What is an XML Element?

An XML element is everything from (including) the element's start tag to (including) the element's end tag.

1. **Start Tag**: This marks the beginning of an element.
- It starts with a < and contains the element's name.
- For example: <name>.
2. **Content** (optional): This is the actual data or text between the start and end tags. The content can be:
- Text
- Nested elements (child elements)
- Empty (in case of self-closing tags)
3. **End Tag**: This marks the end of the element.
- It starts with </ followed by the element's name, and closes with a >.
- For example: </name>

An element can contain:

- text
- attributes
- other elements
- or a mix of the above

Example :

```
<bookstore>

  <book category="children">

    <title>Harry Potter</title>

    <author>J K. Rowling</author>

    <year>2005</year>

    <price>29.99</price>

  </book>

  <book category="web">

    <title>Learning XML</title>

    <author>Erik T. Ray</author>

    <year>2003</year>

    <price>39.95</price>

  </book>

</bookstore>
```

In the example above:

<title>, <author>, <year>, and <price> have **text content** because they contain text (like Harry Potter). <bookstore> and <book> have **element contents**, because they contain elements. <book> has an **attribute** (category="children").

XML Naming Rules

XML elements must follow these naming rules:

- Element names are case-sensitive
- Element names must start with a letter or underscore
- Element names cannot start with the letters xml (or XML, or Xml, etc)
- Element names can contain letters, digits, hyphens, underscores, and periods
- Element names cannot contain spaces

Any name can be used, no words are reserved (except xml).

Best Naming Practices

1. Create descriptive names, like this: <person>, <firstname>, <lastname>.
2. Create short and simple names, like this: <book_title> not like this: <the_title_of_the_book>.
3. Avoid "-". If you name something "first-name", some software may think you want to subtract "name" from "first".
4. Avoid ".". If you name something "first.name", some software may think that "name" is a property of the object "first".
5. Avoid ":". Colons are reserved for namespaces (more later).
6. Non-English letters like éòá are perfectly legal in XML, but watch out for problems if your software doesn't support them!

XML DTD

- DTD stands for Document Type Definition.
- An XML document with correct syntax is called "Well Formed".
- An XML document validated against a DTD is both "Well Formed" and "Valid".
- A Document Type Definition **(DTD)** describes the tree structure of a document and something about its data.
- The purpose of a DTD is to define the structure and the legal elements and attributes of an XML document
- A DTD can be declared inside an XML document as inline or as an external recommendation.
- The DTD doesn’t support any Data type.

There are 2 data types, PCDATA and CDATA

- PCDATA is parsed character data.
- CDATA is character data, not usually parsed.

Example :

XML Document

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "Note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Note.dtd:

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

The DTD above is interpreted like this:

- !DOCTYPE note - Defines that the root element of the document is note
- !ELEMENT note - Defines that the note element must contain the elements: "to, from, heading, body"
- !ELEMENT to - Defines the to element to be of type "#PCDATA"
- !ELEMENT from - Defines the from element to be of type "#PCDATA"
- !ELEMENT heading - Defines the heading element to be of type "#PCDATA"
- !ELEMENT body - Defines the body element to be of type "#PCDATA"

Internal DTD and External DTD :

Internal DTD :

- Internal DTD is a type of Document Type Definition (DTD) in XML that is written within the XML document itself.
- It specifies the structure and rules for the elements and attributes of the XML document.
- An internal DTD is enclosed within the <!DOCTYPE> declaration of the XML document and is defined using a set of predefined keywords and syntax.
- Internal DTDs are suitable for smaller XML documents where the complexity of the structure is not very high.
- It is easier to maintain and modify the internal DTD as it is part of the XML document itself.

Syntax:

```
<!DOCTYPE root_element[ <!ELEMENT element_name (element_content)>
<!ELEMENT another_element_name (another_element_content)>
]>
```

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE customers[ <!ELEMENT customers (customer+)>
   <!ELEMENT customer (name, email, phone)>
   <!ELEMENT name (#PCDATA)>
   <!ELEMENT email (#PCDATA)>
   <!ELEMENT phone (#PCDATA)>
]>
```

```
  <customers>
    <customer>
      <name>Satyam Nayak</name>
      <email>Satyam@Nayak.com</email>
      <phone>112-123-1234</phone>
    </customer>
    <customer>
      <name>Sonu N</name>
      <email>Sonu@N.com</email>
      <phone>112-455-9969</phone>
    </customer>
  </customers>
```

External DTD :

- External DTD is a type of Document Type Definition (DTD) in XML that is located outside of the actual XML document it describes.
- It can be stored in a separate file or accessed via a URL, and it defines the structure, rules, and constraints for the elements and attributes within an XML document.
- By using an external DTD, multiple XML documents can share the same set of rules and constraints, leading to more consistency and easier maintenance.
- External DTD can also be updated independently without having to modify the XML documents themselves.

Syntax:

```
<!DOCTYPE root_element SYSTEM "DTD_file_name">
```

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE customers SYSTEM "customers.dtd">
<customers>
  <customer>
    <name>Satyam Nayak</name>
    <email>Satyam@nayak.com</email>
    <phone>122-112-1234</phone>
  </customer>
  <customer>
    <name>Sonu N</name>
    <email>Sonu@N.com</email>
    <phone>112-554-9969</phone>
  </customer>
</customers>
```

XML Schema

- An XML Schema describes the structure of an XML document. It is like DTD but provides more control on XML structure.
- The XML Schema language is also referred to as XML Schema Definition (XSD)
- An XML document is called "well-formed" if it contains the correct syntax. A well-formed and valid XML document is one which have been validated against Schema.
- The XML Schema support Data type.

XML Schema Data types

There are two types of data types in XML schema.

1. simpleType
2. complexType

Example :

```
<xs:element name="note">
<xs:complexType>
  <xs:sequence>
    <xs:element name="to" type="xs:string"/>
    <xs:element name="from" type="xs:string"/>
    <xs:element name="heading" type="xs:string"/>
    <xs:element name="body" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
</xs:element>
```

The Schema above is interpreted like this:

- <xs:element name="note"> defines the element called "note"
- <xs:complexType> the "note" element is a complex type
- <xs:sequence> the complex type is a sequence of elements
- <xs:element name="to" type="xs:string"> the element "to" is of type string (text)
- <xs:element name="from" type="xs:string"> the element "from" is of type string
- <xs:element name="heading" type="xs:string"> the element "heading" is of type string
- <xs:element name="body" type="xs:string"> the element "body" is of type string

Difference between xml schema and xml DTD

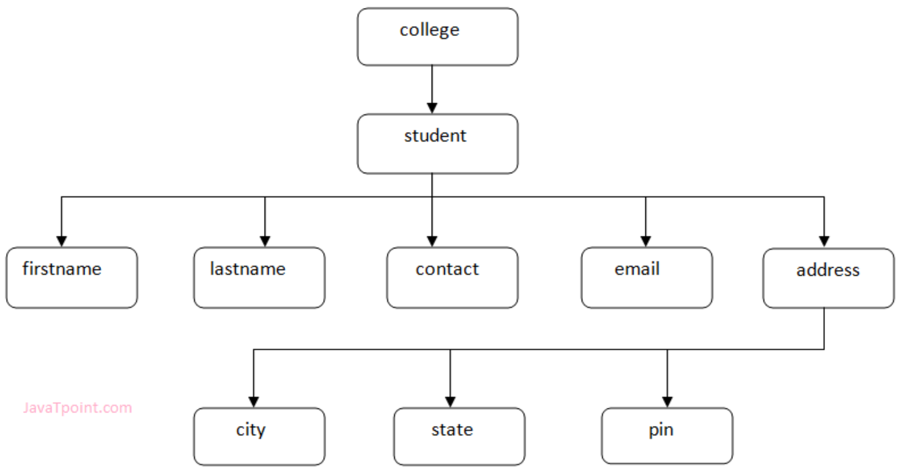
DTD	**XSD**
DTD stands for Document Type Definition	XSD stands for XML Schema Definition.
It doesn't support namespace.	It supports namespace.
It is comparatively harder than XSD.	It is relatively more simpler than DTD.
It doesn't support datatypes.	It supports datatypes.
It is not extensible in nature.	It is extensible in nature.
It doesn't give us much control on structure of XML document.	It gives us more control on structure of XML document.
It specifies only the root element.	Any element which is made global can be done as root as markup validation.
It doesn't have any restrictions on data used.	It specifies certain data restrictions.
It is not much easy to learn .	It is simple in learning.
File here is saved as .dtd	File in XSD is saved as .xsd file.
It uses #PCDATA which is a string data type.	It uses fundamental and primitive data types.

XML DOM

1. DOM is an acronym stands for Document Object Model.
2. It defines a standard way to access and manipulate documents.
3. The Document Object Model (DOM) is a programming API for HTML and XML documents.
4. It defines the logical structure of documents and the way a document is accessed and manipulated.
5. It defines the structure of an XML document in a tree-like format where each element, attribute, and piece of text becomes an object.
6. These objects can be manipulated, allowing you to read, change, add, or delete elements within the XML document.

Key Concepts of the XML DOM:

1. **Tree Structure**:
   - The XML document is represented as a tree of nodes. The topmost node is the **root** (usually the document's root element), and every other part of the XML document (elements, attributes, text, comments) is a node within this tree.
2. **Node Types**:
   - **Element Node**: Represents an XML element (e.g., <name>).
   - **Attribute Node**: Represents an attribute of an element (e.g., id="123").
   - **Text Node**: Represents the text content inside an element.
   - **Comment Node**: Represents comments within the XML(ex-<!-- comment -->).
   - **Document Node**: The top-most node representing the entire XML document.
3. **Parent-Child Relationships**:
   - Nodes have **parent-child** relationships, where an element can be a **parent** of one or more child nodes.
   - Each element can contain nested child elements, text, attributes, etc.



XSLT(eXtensible Stylesheet Language Transformations)

- XSLT (eXtensible Stylesheet Language Transformations) is the recommended style sheet language for XML.
- XSLT is far more sophisticated than CSS. With XSLT you can add/remove elements and attributes to or from the output file. You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.
- **XSLT** is a language used to transform XML documents into different formats, such as HTML, plain text, or another XML structure.
- It is part of the **XSL (Extensible Stylesheet Language)** family, which is designed for formatting and transforming XML data.

Example :

Input XML:

```
<bookstore>
   <book>
      <title>Learning XML</title>
      <author>ABC</author>
      <price>800</price>
   </book>
</bookstore>
```

XSLT Stylesheet:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:template match="/">
      <html>
         <body>
            <h2>Bookstore</h2>
            <table border="1">
               <tr bgcolor="blue">
                  <th>Title</th>
                  <th>Author</th>
                  <th>Price</th>
               </tr>
                  <tr>
                     <td><xsl:value-of select=" bookstore/book/title" /></td>
                     <td><xsl:value-of select=" bookstore/book/author" /></td>
                     <td><xsl:value-of select=" bookstore/book/price" /></td>
                  </tr>
```

```
          </table>
        </body>
      </html>
  </xsl:template>
</xsl:stylesheet>
```

An ecommerce 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. **Purchase By**
VI. **Seller Name**

Write the HTML, XML Code and DTD for the given data.

HTML Code :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Product Purchase Form</title>
</head>
<body>
  <h1>Product Purchase Form</h1>

  <form action="process_form.php" method="POST">
    <label for="product_id">Product ID:</label>
    <input type="text" id="product_id" name="product_id" required><br><br>

    <label for="product_name">Product Name:</label>
    <input type="text" id="product_name" name="product_name" required><br><br>

    <label for="product_cost">Product Cost:</label>
    <input type="number" id="product_cost" name="product_cost" required><br><br>

    <label for="purchase_date">Purchase Date:</label>
    <input type="date" id="purchase_date" name="purchase_date" required><br><br>
```

```
        <label for="purchase_by">Purchased By:</label>
        <input type="text" id="purchase_by" name="purchase_by" required><br><br>

        <label for="seller_name">Seller Name:</label>
        <input type="text" id="seller_name" name="seller_name" required><br><br>

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

XML Code :

```
<?xml version="1.0" encoding="UTF-8"?>
<productPurchase>
    <product>
        <productID>12345</productID>
        <productName>Wireless Mouse</productName>
        <productCost>29.99</productCost>
        <purchaseDate>2024-11-06</purchaseDate>
        <purchaseBy>John Doe</purchaseBy>
        <sellerName>ABC Electronics</sellerName>
    </product>
</productPurchase>
```

DTD :

```
<!ELEMENT productPurchase (product+)>
<!ELEMENT product (productID, productName, productCost, purchaseDate, purchaseBy,
sellerName)>
<!ELEMENT productID (#PCDATA)>
<!ELEMENT productName (#PCDATA)>
<!ELEMENT productCost (#PCDATA)>
<!ELEMENT purchaseDate (#PCDATA)>
<!ELEMENT purchaseBy (#PCDATA)>
<!ELEMENT sellerName (#PCDATA)>
```

Create a well formed XML Document to maintain Library Catalogue. It should contain name of the Book, author, Publisher and year of publishing. Format it in tabular manner using XSLT.

XML Code :

```
<?xml version="1.0" encoding="UTF-8"?>
<libraryCatalogue>
  <book>
    <name>The Great Gatsby</name>
    <author>F. Scott Fitzgerald</author>
    <publisher>Charles Scribner's Sons</publisher>
    <yearOfPublishing>1925</yearOfPublishing>
  </book>
</libraryCatalogue>
```

XSLT :

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

  <!-- Template to match the root and generate the HTML table -->
  <xsl:template match="/">
    <html>
      <head>
        <title>Library Catalogue</title>
        <style>
          table, th, td {
            border: 1px solid black;
          }
          th {
            background-color: Blue;
          }
        </style>
      </head>
      <body>
        <h1>Library Catalogue</h1>
        <table>
          <tr>
            <th>Book Name</th>
            <th>Author</th>
            <th>Publisher</th>
```

```
                <th>Year of Publishing</th>
              </tr>
              <!-- Apply templates to each 'book' element -->
              <xsl:for-each select="libraryCatalogue/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="yearOfPublishing" /></td>
                </tr>
              </xsl:for-each>
            </table>
          </body>
        </html>
      </xsl:template>

    </xsl:stylesheet>
```

Introduction to PHP

PHP is a server-side scripting language created primarily for web development but it is also used as a general-purpose programming language. Unlike client-side languages like JavaScript, which are executed on the user’s browser, PHP scripts run on the server. The results are then sent to the client’s web browser as plain HTML.

What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use

What is a PHP File?

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code is executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"

What Can PHP Do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data.

Example :

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello World in PHP</title>
</head>
<body>
<?php
  // This line prints the Hello World message
  echo "Hello, World!";
?>
</body>
</html>
```

Features of PHP

1. **Simple**
   PHP is particularly famous for its simplicity. It is organized and easy to learn. Even beginners won't face any hard time learning and using PHP. It is a very well-organized programming language, and it comes with a lot of pre-defined functions, which makes the task of the programmer easy. There is no need to include libraries in PHP like C. With a lot of pre-defined functions, PHP is easy to optimize as well.
2. **Interpreted language**
   PHP is an interpreted language, which means there is no need for compilation. Interpreters run through a program line by line and execute the code.
3. **Fast Performance**
   PHP scrips are usually faster than other scripting languages. Users can load their web pages faster.
4. **Free and open-source**
   PHP is open-source, which means it can be downloaded and used freely. There is absolutely no need to acquire a license to use it and no payment is required to use it.
5. **Case-sensitive**
   PHP is case sensitive scripting language at the time of variable declaration. In PHP, all keywords (eg. if, else, while, echo, etc.), classes, functions and user-defined functions are not case-sensitive.
6. **Platform independent**
   We can run PHP on any device and operating system (Microsoft Windows, macOS, Linux, RISC OS, or Unix). We can easily connect it with various databases and is also compatible with almost all web servers used today
7. **Objective oriented**
   PHP supports object-oriented programming features like data encapsulation, inheritance, abstraction, polymorphism, etc. The Object-oriented programming feature was added in PHP5.

Comment in PHP

Single Line Comments

- Single line comments start with //.
- Any text between // and the end of the line will be ignored (will not be executed).
- You can also use # for single line comments.

Multi-line Comments

- Multi-line comments start with /* and end with */.
- Any text between /* and */ will be ignored.

Variables in PHP

In PHP, a variable starts with the $ sign, followed by the name of the variable.
A variable can have a short name (like $x and $y) or a more descriptive name ($age, $carname, $total_volume).

Rules for PHP variables:

- A variable starts with the $ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive ($age and $AGE are two different variables).

Example

```
<html>
<body?
<?php
        $number = 10;        // Integer
        $pi = 3.14;          // Float
        $name = "Alice";     // String
        $isLoggedIn = true;  // Boolean
        Echo $number;
        Echo $pi;
        Echo $name;
        Echo $isLoggedIn;
?>
</body>
<html>
```

Scope of Variable :

In PHP, a variable starts with the $ sign, followed by the name of the variable.
A variable can have a short name (like $x and $y) or a more descriptive name ($age, $carname, $total_volume).

Rules for PHP variables:

1. A variable starts with the $ sign, followed by the name of the variable
2. A variable name must start with a letter or the underscore character
3. A variable name cannot start with a number
4. A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

5. Variable names are case-sensitive ($age and $AGE are two different variables).

Example

```
<html>
<body?
<?php
        $number = 10;        // Integer
        $pi = 3.14;          // Float
        $name = "Alice";     // String
        $isLoggedIn = true;  // Boolean
        Echo $number;
        Echo $pi;
        Echo $name;
        Echo $isLoggedIn;
?>
</body>
<html>
```

PHP Variable Scope

The scope of a variable is defined as its range in the program under which it can be accessed. In other words, "The scope of a variable is the portion of the program within which it is defined and can be accessed."

PHP has three types of variable scopes:

1. Local variable

2. Global variable

3. Static variable

1. Local variable

The variables that are declared within a function are called local variables for that function. These local variables have their scope only in that particular function in which they are declared. This means that these variables cannot be accessed outside the function, as they have local scope.

Ex :

```
<?php
  function local_var()
  {
    $num = 45;  //local variable
```

```
        echo "Local variable declared inside the function is: ". $num;
    }
    local_var();
?>
```

2. Global variable

The global variables are the variables that are declared outside the function. These variables can be accessed anywhere in the program. To access the global variable within a function, use the GLOBAL keyword before the variable.

Ex :

```
<?php
    $name = "Sanaya Sharma";        //Global Variable
    function global_var()
    {
        global $name;
        echo "Variable inside the function: ". $name;
        echo "</br>";
    }
    global_var();
    echo "Variable outside the function: ". $name;
?>
```

3. Static variable

It is a feature of PHP to delete the variable, once it completes its execution and memory is freed. Sometimes we need to store a variable even after completion of function execution.

Ex :

```
<?php
    function static_var()
    {
        static $num1 = 3;       //static variable
        $num1++;
        echo "Static: " .$num1 ."</br>";
    }

//first function call
    static_var();

    //second function call
    static_var();
?>
```

PHP Data Types

Variables can store data of different types, and different data types can do different things. PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL
- Resource

PHP String

- A string is a sequence of characters, like "Hello world!".
- A string can be any text inside quotes.

Example : $x = "Hello world!";

PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation.

Example : $x = 5985;

PHP Float

A float (floating point number) is a number with a decimal point or a number in exponential form.
Example : $x = 10.365;

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.
Example : $x = true; $y = false;

PHP Array

An array stores multiple values in one single variable.
Example : $cars = array("Volvo","BMW","Toyota");

Control structure used in PHP

PHP allows us to perform actions based on some type of conditions that may be logical or comparative. Based on the result of these conditions i.e., either TRUE or FALSE, an action would be performed as asked by the user. It's just like a two- way path. If you want something then go this way or else turn that way. To use this feature, PHP provides us with four conditional statements:

- **if** statement
- **if…else** statement
- **if…elseif…else** statement
- **switch** statement

Let us now look at each one of these in details:

1. **if Statement**: This statement allows us to set a condition. On being TRUE, the following block of code enclosed within the if clause will be executed.

Syntax :

```
if (condition){
   // if TRUE then execute this code
}
```

Example:

```
<?php
$x = 12;
if ($x > 0) {
   echo "The number is positive";
}
?>
```

2. **if…else Statement**:

We understood that if a condition will hold i.e., TRUE, then the block of code within if will be executed. But what if the condition is not TRUE and we want to perform an action? This is where else comes into play. If a condition is TRUE then if block gets executed, otherwise else block gets executed.

Syntax:

```
if (condition) {

   // if TRUE then execute this code

}

else{

   // if FALSE then execute this code }
```

Example:

```
<?php

$x = -12;

if ($x > 0) {

   echo "The number is positive";

}

else{

   echo "The number is negative";

}

?>
```

1. **if…elseif…else Statement**:

This allows us to use multiple if…else statements. We use this when there are multiple conditions of TRUE cases.
Syntax:

```
if (condition) {

   // if TRUE then execute this code

}

elseif {

   // if TRUE then execute this code

}

elseif {

   // if TRUE then execute this code

}

else {

   // if FALSE then execute this code

}
```

Example:

```
<?php
$x = "August";
if ($x == "January") {
   echo "Happy Republic Day";
}
elseif ($x == "August") {
   echo "Happy Independence Day!!!";
}
else{
   echo "Nothing to show";
}
?>
```

Form validation works in PHP

An HTML form contains various input fields such as text box, checkbox, radio buttons, submit button, and checklist, etc. These input fields need to be validated, which ensures that the user has entered information in all the required fields and also validates that the information provided by the user is valid and correct.
There is no guarantee that the information provided by the user is always correct. PHP validates the data at the server-side, which is submitted by HTML form. You need to validate a few things:

1. Empty String
2. Validate String
3. Validate Numbers
4. Validate Email
5. Validate URL
6. Input length

Empty String
The code below checks that the field is not empty. If the user leaves the required field empty, it will show an error message. Put these lines of code to validate the required field.
Ex:

```
if (emptyempty ($_POST["name"])) {
   $errMsg = "Error! You didn't enter the Name.";
            echo $errMsg;
} else {
   $name = $_POST["name"];
}
```

Validate String

The code below checks that the field will contain only alphabets and whitespace, for example - name. If the name field does not receive valid input from the user, then it will show an error message:

Ex:

```
$name = $_POST ["Name"];
if (!preg_match ("/^[a-zA-z]*$/", $name) ) {
   $ErrMsg = "Only alphabets and whitespace are allowed.";
         echo $ErrMsg;
} else {
   echo $name;
}
```

Validate Number

The below code validates that the field will only contain a numeric value. **For example -** Mobile no. If the *Mobile no* field does not receive numeric data from the user, the code will display an error message:

Ex:

```
$mobileno = $_POST ["Mobile_no"];
if (!preg_match ("/^[0-9]*$/", $mobileno) ){
   $ErrMsg = "Only numeric value is allowed.";
   echo $ErrMsg;
} else {
   echo $mobileno;
}
```

Validate Email

A valid email must contain @ and . symbols. PHP provides various methods to validate the email address. Here, we will use regular expressions to validate the email address.

The below code validates the email address provided by the user through HTML form. If the field does not contain a valid email address, then the code will display an error message:

```
$email = $_POST ["Email"];
$pattern = "^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,3})$^";
if (!preg_match ($pattern, $email) ){
   $ErrMsg = "Email is not valid.";
         echo $ErrMsg;
} else {
   echo "Your valid email address is: " .$email;
}
```

Input Length Validation

The input length validation restricts the user to provide the value between the specified range, for Example - Mobile Number. A valid mobile number must have 10 digits.

The given code will help you to apply the length validation on user input:

Ex :

```
    $mobileno = strlen ($_POST ["Mobile"]);
    $length = strlen ($mobileno);

    if ( $length < 10 && $length > 10) {
       $ErrMsg = "Mobile must have 10 digits.";
             echo $ErrMsg;
    } else {
       echo "Your Mobile number is: " .$mobileno;
    }
```

Validate URL

The below code validates the URL of website provided by the user via HTML form. If the field does not contain a valid URL, the code will display an error message, i.e., "URL is not valid".

Ex :

```
    $websiteURL = $_POST["website"];
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
      $websiteErr = "URL is not valid";
      echo $websiteErr;
    } else {
       echo "Website URL is: " .$websiteURL;
    }
```

PHP program to print factorial of number.

```
<HTML>

<BODY>

        <?php

        $num = 4;

        $factorial = 1;

        for ($x=$num; $x>=1; $x--)

        {
```

```
  $factorial = $factorial * $x;
 }
 echo "Factorial of $num is $factorial";
 ?>
 </BODY>
</HTML>
```

Q. If you wanted to send sensitive information like password to the backend which among the GET and POST method in PHP would you use, Justify your answer and distinguish between these two methods.

In PHP (and in web development in general), POST is the more appropriate method to send sensitive information, such as passwords, to the backend.

1. **Security and Data Handling**:

   POST

   - The POST method sends data in the **body of the HTTP request**, which means it is **not visible in the URL**. This makes it more secure for sensitive data like passwords, as the data is not exposed in browser history or server logs.
   - For example, sending a password using POST looks like this:

     ```
     POST /login HTTP/1.1

     Host: example.com

     Content-Type: application/x-www-form-urlencoded

     username=johndoe&password=mysecretpassword
     ```

   GET

   - **GET**: The GET method appends data to the URL in the form of query parameters, which means the data is **visible in the URL**. This is a significant security risk because URLs can be logged in browser history, web server logs, and are potentially visible in the browser's address bar.
   - For example, sending a password using GET might look like this:

     https://example.com/login?username=johndoe&password=mysecretpassword

2. **Data Size**:
   - **POST** has no such limitations and can handle much larger payloads, making it more suitable for sending large amounts of data (like complex passwords or form data).
   - **GET** has limitations on the amount of data you can send, as the data is appended to the URL. URLs typically have a length limit (around 2048 characters for most browsers).
3. **Data Visibility**:
   - **POST** is generally more **secure** because the data is sent in the request body, which is not logged or stored in the browser's history.
   - **GET** is **less secure** because anyone who can access the URL (through browser history, proxy logs, or server logs) can see the data being transmitted.

GET	**POST**
1) In case of Get request, only limited amount of data can be sent because data is sent in header.	In case of post request, large amount of data can be sent because data is sent in body.
2) Get request is not secured because data is exposed in URL bar.	Post request is secured because data is not exposed in URL bar.
3) Get request can be bookmarked.	Post request cannot be bookmarked.
4) Get request is idempotent . It means second request will be ignored until response of first request is delivered	Post request is non-idempotent.
5) Get request is more efficient and used more than Post.	Post request is less efficient and used less than get.
6) Request made through GET method are stored in Browser history.	Request made through POST method is not stored in Browser history.
7) Request made through GET method are stored in cache memory of Browser.	Request made through POST method are not stored in cache memory of Browser.
8) In GET method only ASCII characters are allowed.	In POST method all types of data is allowed.
9) GET requests are only used to request data (not modify)	POST requests can be used to create and modify data.