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

Q2)

=> ① For sending sensitive information like password to the backend, you should always use the POST method rather than GET.

② POST data is not visible in the URL unlike GET parameters which are appended to the URL.

③ It means that sensitive data sent by POST is not exposed anywhere.

④ POST can handle large amount of data while GET has limitations.

⑤ Eg:

```
<?php
  if ($_SERVER["REQUEST_METHOD"]=="POST")
  {
    $username = $_POST['username'];
    $password = $_POST['password'];

    //Remaining Logic
  }
?>
```

```
<form method="POST" action="login.php">
<input type="text" name="username">
<input type="password" name="password">
<input type="submit" value="Login">
</form>
```

GET	POST
① Using back / reload button is harmless	① Using back / reload button, data will be re-submitted.
② Can be bookmarked	② Cannot be bookmarked
③ Can be cached	③ Not cached.
④ Parameters remain in browser history	④ Parameters are not saved in browser history
⑤ Data is visible to everyone in URL	⑤ Data is not displayed in URL

Q3) Form validation in PHP

=> (1) Form validation is a crucial aspect of web development that ensures the data submitted by users through the forms is complete, accurate and secure.

(2) when a user submits the form, the data is sent to the server using the POST method.

(3) The developer defines validation rules to check whether the user enters the right kind of information.

(4) For eg:

(i) A name field must not be empty

(ii) A email address should have '@' and 'example.com'

(iii) A password must be atleast 8 characters long.

(5) Example of name field validation

(i) Initializing the name variable and error message

```
$name = "";
$nameErr = "";
```

(ii) checking if the form is submitted

```
if ($_SERVER["REQUEST_METHOD"] == "POST"){
    // Logic
}
```

(iii) Validating the name field

```
if (empty ($_POST["name"])){
    $nameErr = "Name is required";
}
else {
```

$name = $_POST["name"]

(iv) Displaying the name field and error message in HTML

```
<form method = "post"
    action = "<php echo $_SERVER["PHP_SELF"]; ?>
Name : <input type = "text" name = "name">
    <? php echo $nameErr; ?>
</form>
```

(v) The form displays an input field for the user to display their name.

(vi) The value entered is sent via POST method

(vii) If the name field is left empty, it will show error else it will display the name.

Q. Explain the different data types in PHP.

=>

1. Scalar Types

Scalar types are single-value types, meaning they hold a single value. PHP supports the following scalar data types:

a) Integer

- Represents whole numbers, both positive and negative, without any decimal point.
- Example:

```
$age = 25;  // Integer
```

b) Float (or Double)

- Represents numbers that have a decimal point.
- Example:

```
$price = 10.50;  // Float
```

c) String

- Represents a sequence of characters enclosed in either single or double quotes.
- Example:

```
$name = "John";  // String
$greeting = 'Hello, world!';  // String
```

d) Boolean

- Represents two possible values: TRUE or FALSE.
- Often used in conditional statements.
- Example:

```
$isLoggedIn = true; // Boolean
```

2. Compound Types

Compound types hold multiple values. PHP has the following compound data types:

a) Array

- A collection of values that can be indexed either numerically or associatively (using named keys).
- Example :

```
$colors = array("Red", "Green", "Blue");
```

b) Object

- A collection of data (properties) and functions (methods) that operate on the data.
- Objects are instances of classes.
- Example:

```
class Car {
  public $brand;
  public $model;
  public $year;

  function __construct($brand, $model, $year) {
    $this->brand = $brand;
    $this->model = $model;
    $this->year = $year;
  }
}

$myCar = new Car("Toyota", "Camry", 2020);  // Object
```

3. Special Types

These data types do not store any values but are used for specific purposes in PHP.

a) NULL

- Represents a variable with no value. A variable is set to NULL either explicitly or when it is not initialized.
- Example

```
$var = NULL;  // NULL
```

Q. Discuss various control structures in PHP with suitable examples.
=>

1. Conditional Statements

a) if Statement

```
if (condition) {
  // code to be executed if condition is TRUE
}
```

b) if...else Statement

```
if (condition) {
  // code if TRUE
} else {
  // code if FALSE
}
```

c) if...elseif...else Statement

```
if (condition1) {
  // code if condition1 is TRUE
} elseif (condition2) {
  // code if condition2 is TRUE
} else {
  // code if no condition is TRUE
}
```

d) switch Statement

```
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 no cases match
}
```

2. Looping Structures

a) while Loop

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

b) do...while Loop

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

c) for Loop

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

d) foreach Loop

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

3. Jump Statements

a) break Statement

```
break; // Exit from loop or switch
```

b) continue Statement

```
continue; // Skip the current iteration and continue with the next
```

c) return Statement

```
return; // Exit from function and optionally return a value
```

Q. **Explain the structure of an XML document with an example.**
=>

a. An XML (eXtensible Markup Language) document is structured to represent data in a hierarchical manner, allowing the information to be stored in a readable, text-based format.
b. The structure is designed to be both human-readable and machine-readable.

c. Components of an XML Document:

1. **XML Declaration**

2. The XML declaration is optional but typically placed at the top of an XML document. It defines the XML version and the character encoding used in the document.

    a. **Syntax:**

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

    b. **Explanation:**
        i. version: Defines the version of XML being used, usually "1.0".
        ii. encoding: Specifies the character encoding used in the document (e.g., UTF-8).

3. **Root Element**

Every XML document has a single root element that contains all other elements. It is the outermost tag that holds all content within the document.

a. Syntax:

```
<root>
  <!-- Content goes here -->
</root>
```

4. **Elements**

Elements are the primary building blocks of XML documents. They consist of a start tag, content, and an end tag.

a. Syntax:

```
<element>content</element>
```

b. Explanation:
i. start tag: <element> marks the beginning of the element.
ii. end tag: </element> marks the end of the element.
iii. content: The actual data or nested elements inside the start and end tags.

5. **Attributes**

Elements can contain attributes that provide additional information. Attributes are specified within the start tag.

a. **Syntax:**

<element attribute="value">content</element>

b. **Explanation:**

i. attribute: Defines the name of the attribute.

ii. value: The value assigned to the attribute.

6. **Nested Elements**

XML allows elements to be nested inside one another, forming a hierarchy of data.

a. **Syntax:**

```
<parent>
  <child>content</child>
</parent>
```

7. **Comments**

XML allows comments to be added for clarity or notes. Comments are enclosed within <!-- and -->.

a. **Syntax:**

<!-- This is a comment -->