Discussion

1- What are the differences between GET & POST methods?

GET sends data in the URL, making it visible, while POST sends data in the request body, keeping it hidden.

GET has limitations on the length of data, while POST can handle larger data payloads.

2- Write a PHP script to create a form for entering three names and their emails and then print the names with the emails on the web page as sorting ascending. (Use GET method in your form)

```
<!DOCTYPE html>
<html>
<head>
  <title>Name and Email Form</title>
</head>
<body>
  <h2>Name and Email Form</h2>
  <form method="GET" action="">
    <label for="name1">Name 1:</label>
    <input type="text" name="name1" id="name1" required><br>

    <label for="email1">Email 1:</label>
    <input type="email" name="email1" id="email1" required><br><br>

    <label for="name2">Name 2:</label>
    <input type="text" name="name2" id="name2" required><br>
```

```
    <label for="email2">Email 2:</label>
    <input type="email" name="email2" id="email2" required><br><br>

    <label for="name3">Name 3:</label>
    <input type="text" name="name3" id="name3" required><br>

    <label for="email3">Email 3:</label>
    <input type="email" name="email3" id="email3" required><br><br>

    <input type="submit" value="Submit">
</form>

<?php
if ($_SERVER['REQUEST_METHOD'] === 'GET') {
    $names = array($_GET['name1'], $_GET['name2'], $_GET['name3']);
    $emails = array($_GET['email1'], $_GET['email2'], $_GET['email3']);

    array_multisort($names, $emails);

    echo "<h2>Sorted Names and Emails:</h2>";
    for ($i = 0; $i < count($names); $i++) {
        echo "<p>Name: " . $names[$i] . ", Email: " . $emails[$i] . "</p>";
    }
}
```

```
?>
</body>
</html>
```

3- There are many other methods besides those mentioned in this lecture that could be used to avoid hacking operations. Mention some of them and give an example of one of them.

1. Input Validation: Validate and sanitize user input to prevent malicious data from being processed. For example, using functions like `filter_var()` or regular expressions to validate email addresses or implementing server-side validation for form inputs.
2. Output Escaping: Escape user-generated content when displaying it to prevent cross-site scripting (XSS) attacks. For instance, using `htmlspecialchars()` to encode HTML entities before outputting user-supplied data.
3. Parameterized Queries or Prepared Statements: Use parameterized queries or prepared statements when interacting with databases to prevent SQL injection attacks. This involves separating SQL code from user-supplied data and binding parameters to avoid direct concatenation of user input into queries.
4. Password Hashing: Store passwords securely by hashing them with a strong cryptographic algorithm like bcrypt or Argon2. This protects user passwords even if the database is compromised.

```
<?php
$username = $_POST['username'];
 if (preg_match('/^[a-zA-Z0-9_]+$/', $username)) {
   } else {
echo "Invalid username. Usernames can only contain letters, numbers, and underscore.";
}
?>
```