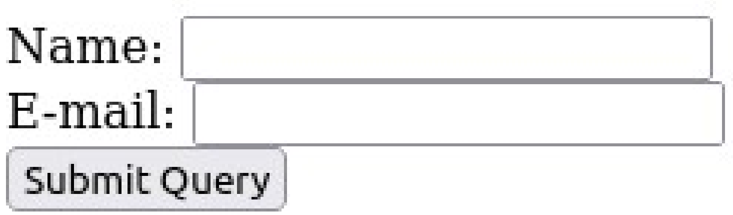
Handling Forms

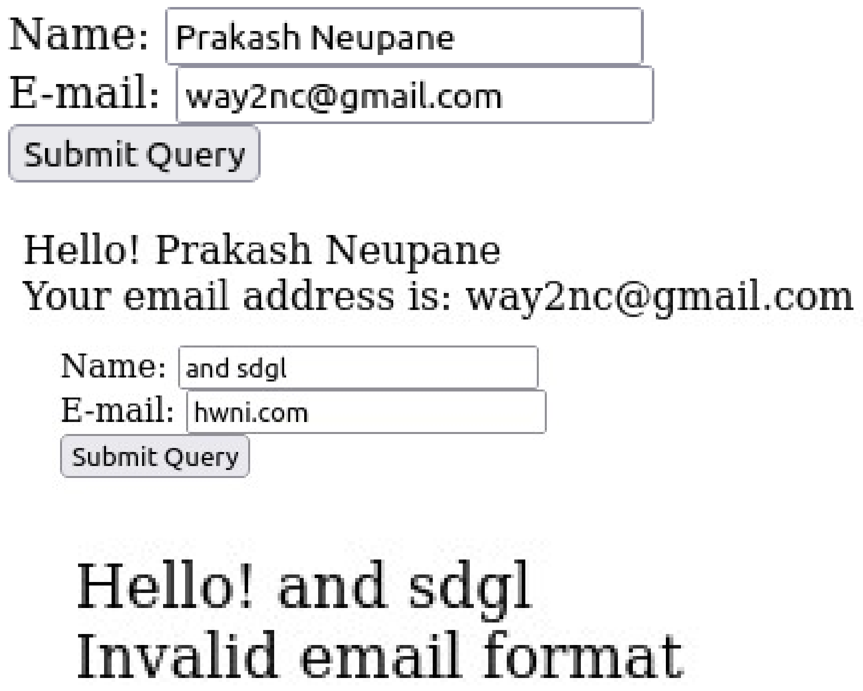
Building forms

```
 <html>
<body>
<form action="Hello.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
</body>
</html>
```



Building forms

```
 <html>
<body>
Hello! <?php echo $_POST["name"]; ?><br>
<?php
$email = ($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  echo "Invalid email format";
} else {
  echo "Your email address is: " . $email;
}
?>
</body>
</html>
```


Name: Prakash Neupane
E-mail: way2nc@gmail.com
Submit Query

```
Hello! Prakash Neupane
Your email address is: way2nc@gmail.com
```

Name: and sdgl
E-mail: hwni.com
Submit Query

```
Hello! and sdgl
Invalid email format
```

Building forms

- **Communication Design:** HTTP is crafted to facilitate communication between clients and servers.

- **Request-Response Mechanism:** It operates through a request-response model, where a client, typically a browser, sends an HTTP request to a server, and the server replies with a response.
- **Information Exchange:** The response not only includes status details about the request but may also deliver the content that was requested by the client.

Retrieving Form Data

- **The GET Method**
- GET is used to request data from a specified resource.
- Note that the query string (name/value pairs) is sent in the URL of a GET request:
- ***http://localhost/form/form2.php?name=af&email=way2nc%40gmail.com***

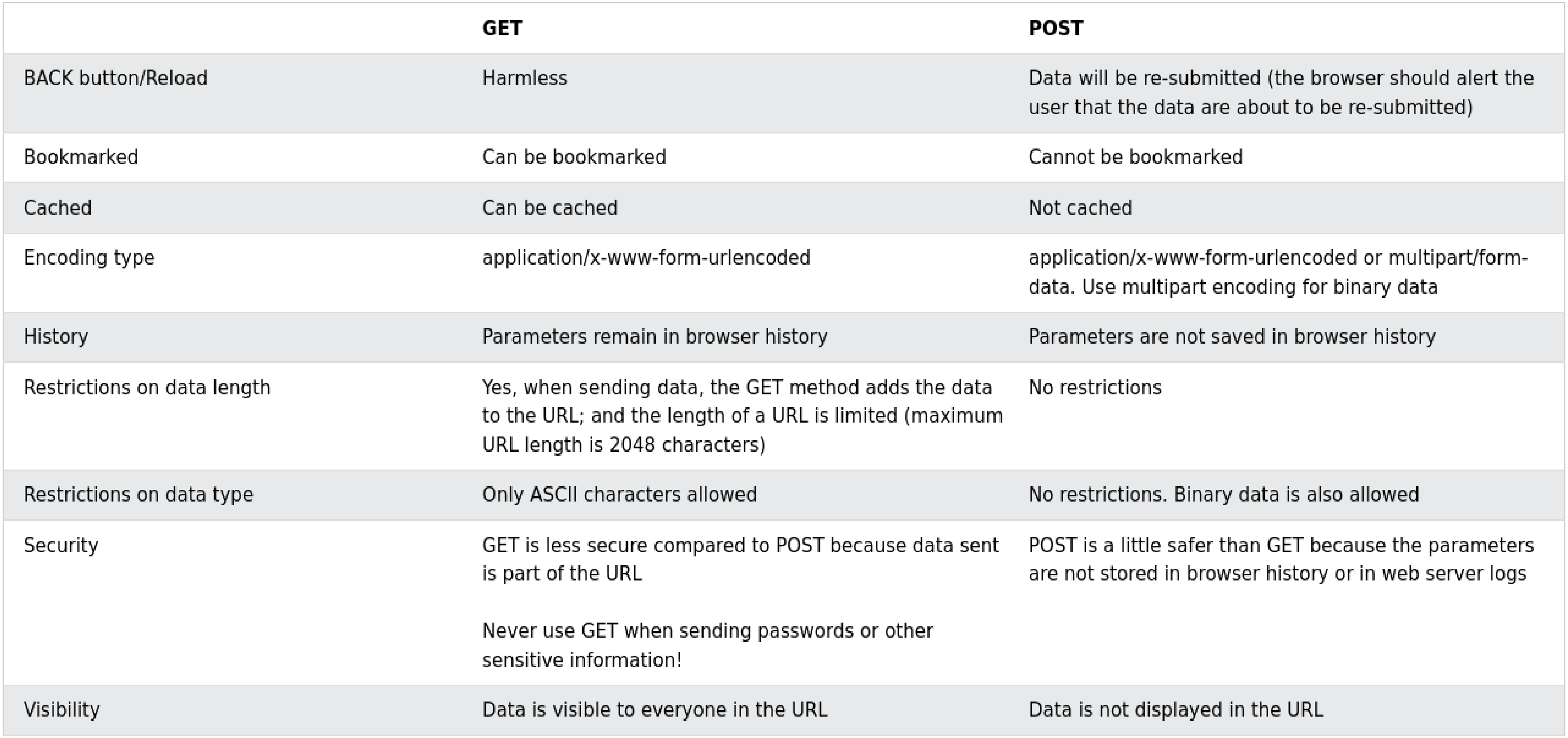
- **Caching and Bookmarking**: GET requests are cacheable and bookmarkable.
- **Browser History and Length Restrictions:** GET requests are recorded in browser history and have length restrictions.
- **Data Security and Modification Limitations:** Avoid using GET for sensitive data; it's designed for data retrieval, not modification.

Retrieving Form Data

- **The POST Method**
- POST is used to send data to a server to create/update a resource.
- The data sent to the server with POST is stored in the request body of the HTTP request:
- ***http://localhost/form/Hello.php***

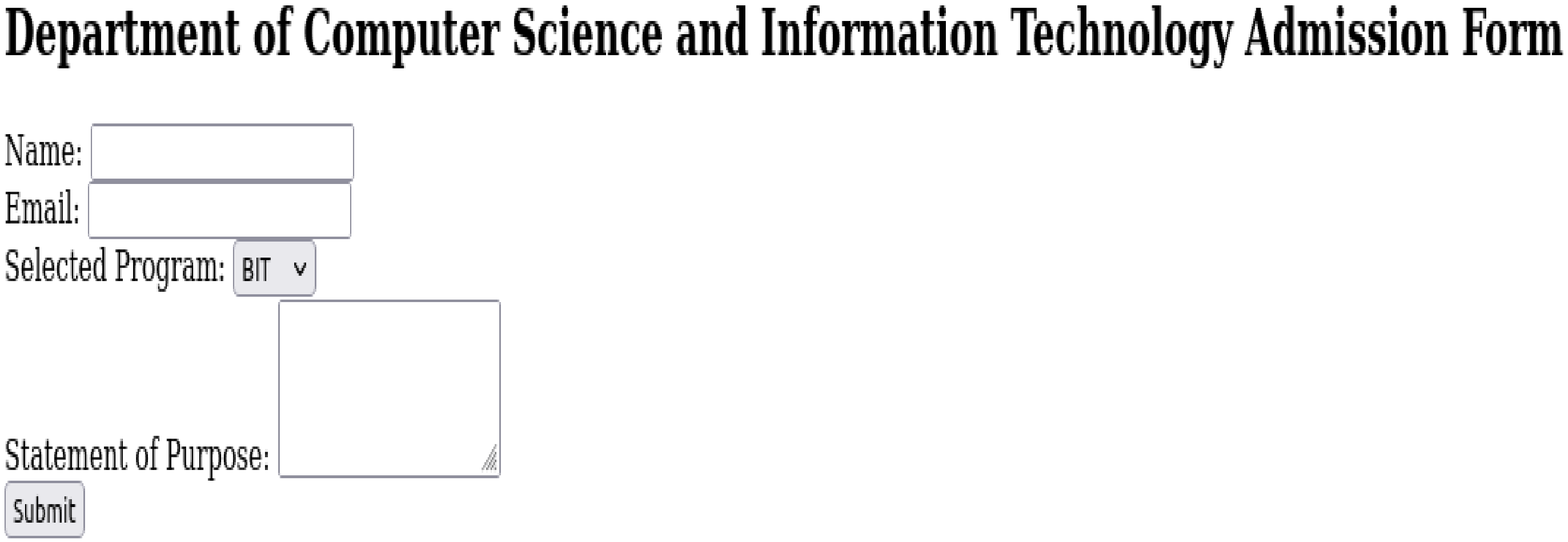
- **Some notes on POST requests:**
- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

Retrieving Form Data



	GET	POST
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

Processing Forms


Department of Computer Science and Information Technology Admission Form

Name:

Email:

Selected Program: BIT

Statement of Purpose:

Processing Forms

```
<!DOCTYPE html>
<html>
<head>
  <title>CCT Admission Form</title>
</head>
<body>
  <h2>Department of Computer Science and Information Technology Admission Form</h2>

  <form action="process_admission.php" method="post">
–       Name: <input type="text" name="name" required><br>
–       Email: <input type="email" name="email" required><br>
–       Selected Program:
–       <select name="program" required>
–         <option value="CSIT">BIT</option>
–         <option value="BIT">CSIT</option>
–       </select><br>
–       Statement of Purpose: <textarea name="sop" rows="4" required></textarea><br>
–       <input type="submit" value="Submit">
  </form>
</body>
</html>
```

Processing Forms

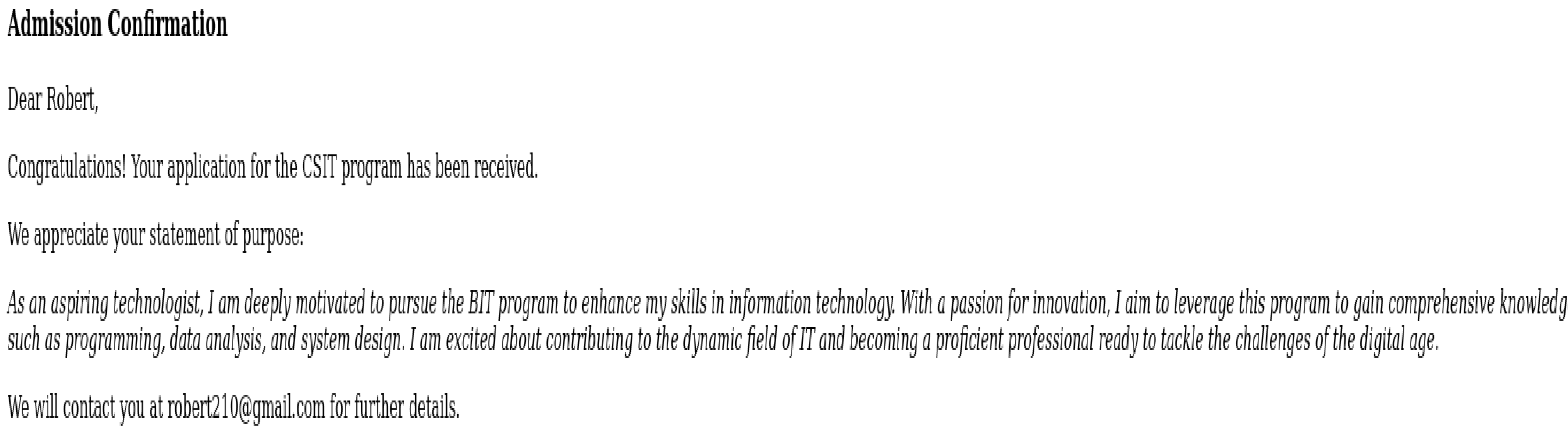
```
<!DOCTYPE html>
<html>
<head>
  <title>CCT Form Processing</title>
</head>
<body>

  <?php
  // Check if form is submitted
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Retrieve form data
    $name = $_POST["name"];
    $email = $_POST["email"];
    $program = $_POST["program"];
    $sop = $_POST["sop"];

    // Display admission confirmation
    echo "<h3>Admission Confirmation</h3>";
    echo "<p>Dear $name,</p>";
    echo "<p>Congratulations! Your application for the $program program has been received.</p>";
    echo "<p>We appreciate your statement of purpose:</p>";
    echo "<p><em>$sop</em></p>";
    echo "<p>We will contact you at $email for further details.</p>";
  } else {
    // If form is not submitted, display an error
    echo "Error: Form not submitted.";
  }
  ?>

</body>
</html>
```

Processing Forms



Admission Confirmation

Dear Robert,

Congratulations! Your application for the CSIT program has been received.

We appreciate your statement of purpose:

As an aspiring technologist, I am deeply motivated to pursue the BIT program to enhance my skills in information technology. With a passion for innovation, I aim to leverage this program to gain comprehensive knowledg such as programming, data analysis, and system design. I am excited about contributing to the dynamic field of IT and becoming a proficient professional ready to tackle the challenges of the digital age.

We will contact you at robert210@gmail.com for further details.

Setting Response Header

- Setting response headers in PHP is essential for controlling various aspects of the HTTP response that the server sends to the client.
- This includes
  - defining content types,
  - handling redirects, and
  - managing caching.
- Here's a simple example of setting response headers in PHP form handling, along with their significance:

Setting Response Header

```
<?php
// Check if form is submitted
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Retrieve form data
    $name = $_POST["name"];
    $email = $_POST["email"];

    // Validate email
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        // Set response header for error
        header("HTTP/1.1 400 Bad Request");
        echo "Invalid email format";
    } else {
        // Set response header for success
        header("HTTP/1.1 200 OK");

        // Display personalized greeting
        echo "Hello, $name! Your email is: $email";
    }
} else {
    // If form is not submitted, set response header for error
    header("HTTP/1.1 404 Not Found");
    echo "Error: Form not submitted.";
}
?>
```

Setting Response Header

- Setting Response Headers:
  - header("HTTP/1.1 400 Bad Request");:
    - This header is set when the email is not valid, indicating a bad request.
  - header("HTTP/1.1 200 OK");:
    - This header is set when the form is successfully processed, indicating a successful response.
  - header("HTTP/1.1 404 Not Found");:
    - This header is set when the form is not submitted, indicating a not found error.

Setting Response Header

- Significance:
- **Status Codes:**
  - Setting appropriate HTTP status codes informs the client about the success or failure of the request. For example, a 200 status code indicates success, while a 400 status code indicates a bad request.
- **Error Handling:**
  - By setting headers, you can control how errors are communicated to the client. For instance, you can provide a specific status code and a corresponding error message.
- **Redirects**:
  - Headers are also used for redirects (header("Location: new_page.php");). This is crucial for directing users to different pages after form submission or other actions.

Setting Response Header

- Task1: Build a User Registration Form:
  - Create a registration form with fields such as username, email, password, and confirm password.
  - Validate the form data on the server side (e.g., check if the email is valid, password meets complexity requirements).
- Task2: Form with Response Headers:
  - Develop a form that requires server-side validation.
  - Set appropriate response headers based on the form validation result (e.g., 200 for success, 400 for bad request).
  - Display different messages to the user based on the response headers.