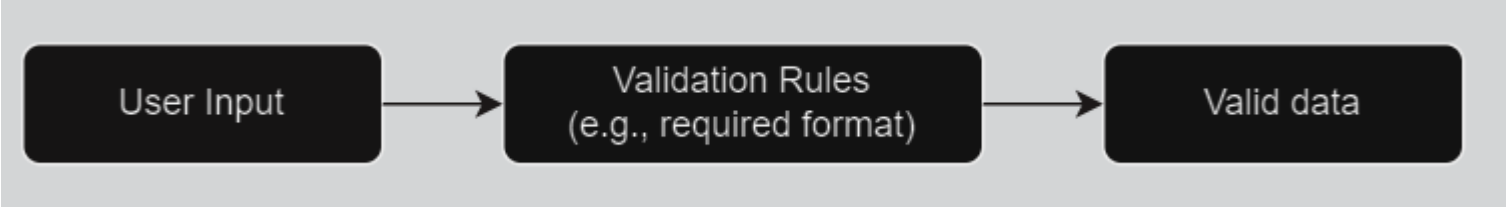
FORM VALIDATION AND SANITIZATION

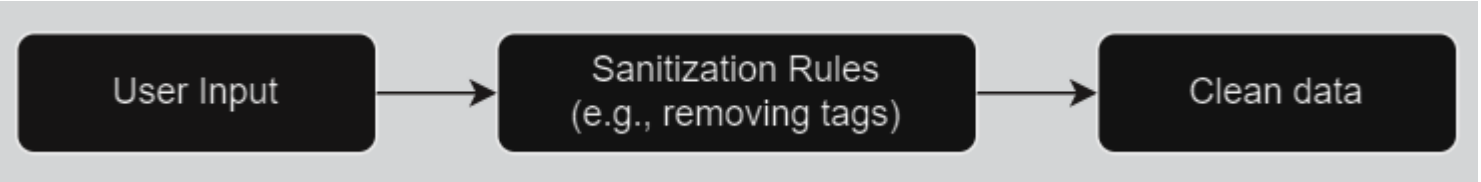
What is Form Validation?

- **Definition:** Checking user input against defined criteria.
- **Purpose:** Ensures accuracy and appropriateness of data.



What is Form Sanitization?

- **Definition:** Cleaning user input to remove unwanted characters.
- **Purpose:** Prevents security issues.



Importance of Validation and Sanitization

- **Security**

✓ Prevents Attacks: Proper validation and sanitization help protect against common attacks like SQL injection, cross-site scripting (XSS), and command injection.

✓ Data Integrity: Ensures that only valid data is processed, reducing the risk of malicious inputs affecting your application.

- **User Experience**

✓ Immediate Feedback: Real-time validation can guide users to correct mistakes, improving their experience and reducing frustration.

✓ Guided Inputs: Clearly defined validation rules help users understand the required format and expectations.

Importance of Validation and Sanitization

- **Data Integrity**

✓ Consistency: Ensures that the data collected meets specific criteria, leading to more accurate and reliable data in databases.

✓ Business Logic: Validated data aligns with application requirements, maintaining the integrity of business processes.

- **Compliance**

✓ Legal Requirements: Many industries have regulations that require data protection and validation practices, ensuring compliance with standards like GDPR or HIPAA.

Importance of Validation and Sanitization

- **Performance**

✓ Resource Optimization: Filtering out invalid data early in the process reduces the load on server resources and databases.

- **Maintainability**

✓ Easier Debugging: Clear validation rules make it easier to trace issues back to user inputs, simplifying debugging and maintenance.

Validation Techniques

- Basic Required Field Validation
- Ensure that fields are not left empty.

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = trim($_POST["name"]);

    if (empty($name)) {
        echo "Name is required.";
    }
}
```

Validation Techniques

- **Email Validation**
- Check if the input is a valid email address.

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $email = trim($_POST["email"]);

    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        echo "Invalid email format.";
    }
}
```

Validation Techniques

- **Number Validation**

  Ensure that a field contains only numeric values.

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $age = trim($_POST["age"]);

    if (!is_numeric($age)) {
        echo "Age must be a number.";
    }
}
```

Validation Techniques

- **String Length Validation**

Check that a string meets minimum and maximum length requirements.

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $password = trim($_POST["password"]);

    if (strlen($password) < 6 || strlen($password) > 12) {
        echo "Password must be between 6 and 12 characters.";
    }
}
```

Validation Techniques

- **Regular Expression Validation**

Use regex to validate specific formats, such as phone numbers.

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $phone = trim($_POST["phone"]);

    if (!preg_match("/^[0-9]{10}$/", $phone)) {
        echo "Phone number must be 10 digits.";
    }
}
```

Validation Techniques

- **Cross-Field Validation**

Ensure that one field is dependent on another.

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $file = $_FILES["file"];

if ($file["error"] != 0)        //// Check for errors
 {
        echo "Error uploading file.";
    }

    // Validate file type
    $allowedTypes = ['image/jpeg', 'image/png'];
    if (!in_array($file["type"], $allowedTypes)) {
        echo "Only JPEG and PNG files are allowed.";
    }
}
```

Validation Techniques

- **CSRF Protection**
- Validate against Cross-Site Request Forgery by using tokens.

```
session_start();
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if ($_POST['token'] !== $_SESSION['token']) {
        echo "CSRF token validation failed.";
    }
}
```

Sanitization Techniques

- **HTML Escaping**

This technique converts special characters to HTML entities to prevent XSS attacks.

```
$name = htmlspecialchars($_POST['name'], NT_QUOTES, 'UTF-8');
```

- **Strip Tags**

Removes all HTML and PHP tags from a string, which is useful when you want plain text.

```
$comment = strip_tags($_POST['comment']);
```

Sanitization Techniques

- **Trimming Whitespace**

Removes whitespace from the beginning and end of a string.

```
$email = trim($_POST['email']);
```

- **Type Casting**

This ensures the variable is of a specific type, such as an integer.

```
$age = (int)$_POST['age'];
```

Sanitization Techniques

- **Regular Expressions**

You can use regex to filter out unwanted characters or formats.

```
$phone = preg_replace("/[^0-9]/", "", $_POST['phone']);
// Only allow digits
```

- **Sanitizing for SQL**

Use prepared statements to avoid SQL injection attacks.

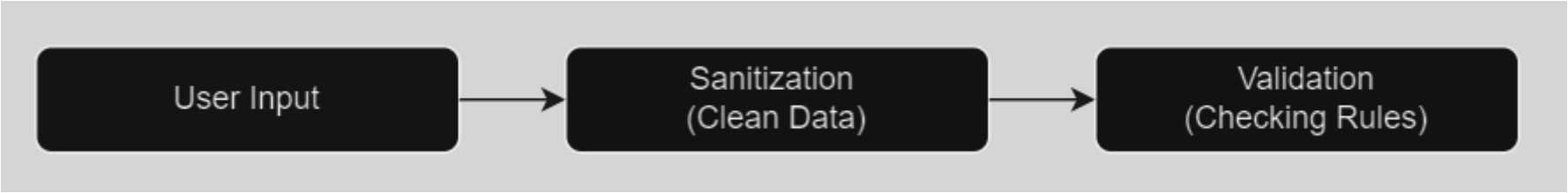
```
$stmt = $pdo->prepare("INSERT INTO users (name, email) VALUES
(:name, :email)");
$stmt->execute([
    'name' => htmlspecialchars($_POST['name']),
    'email' => filter_var($_POST['email'], FILTER_SANITIZE_EMAIL)
]);
```

Combining Validation and Sanitization

- Sanitize Before Validate.

- Why Order Matters?

  Sanitizing first ensures validation operates on clean data.



PHP Global Variables - Superglobals

- Some predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.
- The PHP superglobal variables are:

- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET

PHP Global Variables - Superglobals

- **$_SERVER["PHP_SELF"] variable**

The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.

- **preg_match() function**

The preg_match() function searches a string for pattern, returning true if the pattern exists, and false otherwise

htmlspecialchars() function

- The htmlspecialchars() function converts special characters to HTML entities.
- This means that it will replace HTML characters like < and > with < and >.
- This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.