Forms and Superglobals

A form sends data to the server — learn $_GET and $_POST, the difference between the two methods, $_REQUEST and $_SERVER, and checking input safely.

Introduction

A website that cannot take input is a poster, not an app. Login boxes, search bars, comment forms, signup pages — all of them are HTML forms sending data to PHP. This lesson is the hinge of the whole series: everything after it (sessions, files, databases, the final project) is built on forms.

When a form is submitted, the browser sends its fields to the server, and PHP puts them into **superglobals** — special arrays that exist everywhere in your code without any setup. `$_GET` holds data from the address bar, `$_POST` holds data from a form sent with `method="post"`, `$_REQUEST` holds both, and `$_SERVER` holds facts about the request itself.

Two rules before you write a line. First: **never trust user input.** A visitor can send anything — empty fields, giant text, code. Check every value on the server. Second: front-end checks (like `required` in HTML) are help for the user, not security — anyone can bypass them, so PHP must check again.

(Roman Urdu: Form ka data PHP ko superglobals mein milta hai — `$_GET` address se, `$_POST` form se. User ki koi bhi cheez par bharosa mat karo — har value server par dobara jaancho.)

Key points

- **A form needs `action` (where it goes) and `method` (how it travels).** `action=""` sends it back to the same page.
- **`method="get"` puts the data in the address bar** — visible, bookmarkable, good for search; never for passwords.
- **`method="post"` hides the data in the request body** — invisible in the address, good for logins, comments and anything private.
- **`$_GET["name"]` reads a GET field; `$_POST["name"]` reads a POST field.** The key is the input's `name` attribute.
- **`$_REQUEST` holds GET + POST together** — handy, but prefer `$_GET` or `$_POST` so you always know where a value came from.
- **`$_SERVER["REQUEST_METHOD"]`** tells you whether the page was opened normally (`GET`) or by a form submit (`POST`).
- **`isset($x)` asks "does this exist and is it not null?"** — use it before reading any form field, or PHP warns you.
- **`empty($x)` is true for missing, empty, `"0"` or `0` values** — good for "did the user leave this blank?".
- **Every input's `name` attribute is its key.** An input without a `name` sends nothing at all.

- **Always escape user input when printing it** — `htmlspecialchars($_POST["name"])` — and validate it before saving it anywhere.

Examples

Example 1 — A GET form: search in the address bar

```
<?php
// search.php
if (isset($_GET["q"])) {
    echo "You searched for: " . htmlspecialchars($_GET["q"]);
}
?>
<form method="get" action="">
  <input type="text" name="q" placeholder="Search...">
  <button type="submit">Search</button>
</form>
```

What you will get:

```
After typing "chai" and submitting, the address becomes search.php?q=chai
and the page prints: You searched for: chai
```

GET is right for search: the address can be bookmarked and shared, and the data is not secret.

Example 2 — A POST form: the same page receives it

```
<?php
// hello.php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = trim($_POST["name"]);
    if ($name != "") {
        echo "Hello, " . htmlspecialchars($name) . "!";
    } else {
        echo "Please type your name.";
    }
}
?>
<form method="post" action="">
  <input type="text" name="name" placeholder="Your name">
  <button type="submit">Say hello</button>
</form>
```

What you will get:

```
Typing "Ali" prints: Hello, Ali!
Submitting empty prints: Please type your name.
```

The `REQUEST_METHOD` check means the PHP only runs after a submit — opening the page normally shows just the form. `trim` removes stray spaces before the empty check.

Example 3 — isset vs empty

```
<?php
// Pretend the form sent name="" (empty box) and no age field at all.
$_POST = ["name" => ""];
var_dump(isset($_POST["name"])); // the box exists, but empty
var_dump(empty($_POST["name"])); // empty, so true
var_dump(isset($_POST["age"]));  // never sent at all
var_dump(empty($_POST["age"]));  // missing counts as empty too
?>
```

What you will get:

```
bool(true)
bool(true)
bool(false)
bool(true)
```

`isset` answers "was it sent?"; `empty` answers "is there anything useful in it?". Check `isset` before reading, `empty` before trusting.

Example 4 — Checkboxes and selects: one name, many values

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST["langs"])) {
    echo "You picked: ";
    foreach ($_POST["langs"] as $lang) {
        echo htmlspecialchars($lang) . " ";
    }
}
?>
<form method="post" action="">
  <label><input type="checkbox" name="langs[]" value="PHP"> PHP</label>
  <label><input type="checkbox" name="langs[]" value="Python"> Python</label>
  <label><input type="checkbox" name="langs[]" value="JavaScript"> JavaScript</label>
  <button type="submit">Save</button>
</form>
```

What you will get:

```
Ticking PHP and Python prints: You picked: PHP Python
```

The `[]` after the name tells PHP to collect every ticked box into one array. Without `[]`, only the last ticked box arrives.

Example 5 — $_SERVER: facts about the request

```
<?php
echo $_SERVER["REQUEST_METHOD"];
echo "<br>";
echo $_SERVER["PHP_SELF"];
echo "<br>";
echo $_SERVER["REMOTE_ADDR"];
?>
```

What you will get:

```
GET
/hello.php
127.0.0.1
```

`REQUEST_METHOD` is GET or POST. `PHP_SELF` is the current file's address. `REMOTE_ADDR` is the visitor's address (on your own computer, always 127.0.0.1). Never print `PHP_SELF` raw into a form action — escape it, or leave `action=""` empty.

Example 6 — A tiny validated contact form

```
<?php
$error = "";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $email = trim($_POST["email"]);
    if ($email == "") {
        $error = "Email is required.";
    } elseif (strpos($email, "@") === false) {
        $error = "That does not look like an email.";
    } else {
        echo "Thanks! We will write to " . htmlspecialchars($email);
    }
}
if ($error != "") {
    echo "<p style='color:red'>$error</p>";
}
?>
<form method="post" action="">
  <input type="text" name="email" placeholder="Your email">
  <button type="submit">Send</button>
</form>
```

What you will get:

```
Empty submit: Email is required.
"ali" submit: That does not look like an email.
"ali@example.com" submit: Thanks! We will write to ali@example.com
```

This is the fixed flow for every form you will ever build: check the method, read, trim, validate, show errors or success — in that order.

Quick recap

- GET puts data in the address (search); POST hides it in the body (logins, comments).
- `$_GET` and `$_POST` hold the fields; the key is the input's `name`.
- `$_SERVER["REQUEST_METHOD"]` tells you if the form was submitted.
- `isset` before reading; `empty` before trusting; `trim` first.
- `name="langs[]"` collects many checkboxes into one array.
- Validate on the server every time — browser checks are help, not security.

Common mistakes

1. **Reading `$_POST["name"]` without isset** — opening the page normally triggers an "Undefined array key" warning. Fix: wrap the reading code in an `if ($_SERVER["REQUEST_METHOD"] == "POST")` check.
2. **Forgetting the `name` attribute on an input** — the field sends nothing, and PHP finds no key. Fix: every input that must send data needs a `name`.
3. **Using GET for a password** — the password lands in the address bar, browser history and server logs. Fix: passwords always travel by POST.
4. **Printing `$_POST` values raw** — a visitor can inject HTML or scripts into your page. Fix: `htmlspecialchars()` on every printed user value.
5. **Trusting `required` in the HTML** — anyone can remove it in DevTools and submit an empty form. Fix: validate again in PHP, always.
6. **Forgetting `[]` on multiple checkboxes** — only the last ticked box arrives. Fix: `name="langs[]"` for any group of boxes.
7. **Putting `$_SERVER["PHP_SELF"]` raw in the action** — an attacker can inject code into the address. Fix: use `action=""`, or escape it with `htmlspecialchars()`.

Practice

1. Build a GET search form that prints "You searched for: X" — submit "chai" and read the address bar.
2. Build a POST hello form that greets the name, or says "Please type your name." when the box is empty.
3. Add a second field (age) to the hello form and print "Ali is 20 years old." — cast the age with `(int)`.
4. Build a checkbox group of 3 languages with `name="langs[]"` and print every picked language with a foreach loop.
5. Write the email validation from Example 6 from memory, then test it with "", "ali", and "ali@example.com".
6. Deliberately leave the `name` attribute off an input and submit — write down in one line what PHP receives.
7. Make a form with a `<select>` for city (Lahore, Multan, Sahiwal) and print "You live in X" after submit.

Practice quiz

1. What is the difference between GET and POST?

- There is none
- GET puts the data in the address bar (good for search); POST hides it in the request body (good for logins)
- GET is faster, POST is slower
- GET is for text, POST is for numbers

Answer: GET puts the data in the address bar (good for search); POST hides it in the request body (good for logins)

2. How do you read a form field called "name" sent by POST?

- $name
- $_POST["name"]
- $_GET["name"]
- POST["name"]

Answer: $_POST["name"]

3. Is the required attribute in HTML real security?

- Yes, nothing can bypass it
- No, it is help for the user only; anyone can bypass it, so PHP must check every value again
- Yes, but only for email fields
- No, it does nothing at all

Answer: No, it is help for the user only; anyone can bypass it, so PHP must check every value again

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/ph-forms-superglobals · More free tools and notes at https://scopetreasure.com