

Forms

Intro PHP

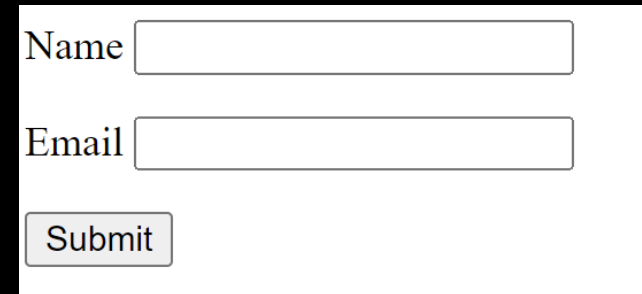
Download the code:

<https://uwpac.ca/code/forms.zip>

HTML Forms and PHP

HTML forms are the primary means by which we capture information from users.

```
<form action="01_html.php" method="post">  
  <input type="text" name="name">  
  <input type="email" name="email">  
  <input type="submit" name="register">  
</form>
```



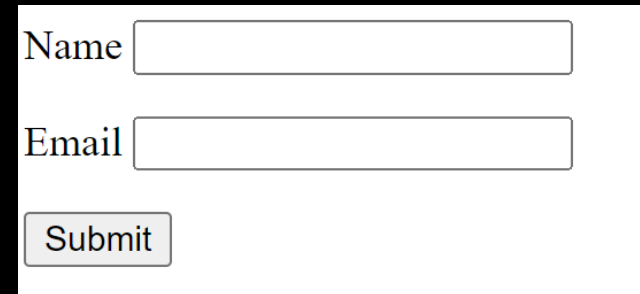
- If the **action** is not specified, the form will submit to the current page.
- If the **method** is not specified, the form will submit as a GET request.

01_html.php

HTML Forms and PHP

HTML forms are the primary means by which we capture information from users.

GET request variables are appended to the URL in the browser address bar.



A screenshot of a web form. It contains two text input fields. The first field is labeled 'Name' and the second is labeled 'Email'. Below these fields is a button labeled 'Submit'.

<http://example.com/?name=Davey&email=davey@example.com>

POST request variables are created as the body of the request HEADERS, and are not visible in the browser.

As a rule of thumb, all forms should submit using the POST method, unless you have a very good reason not to.

01_html.php

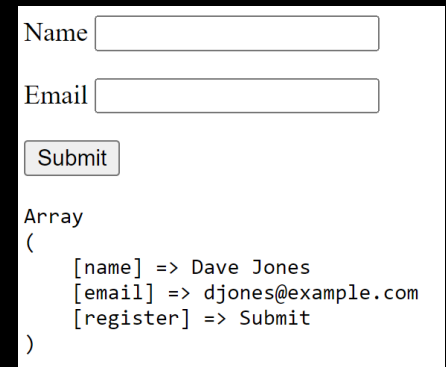
A search form might use the GET method, allowing it to be bookmarked

Form Variables

HTML forms are the primary means by which we capture information from users.

```
<form action="01_html.php" method="post">
  <input type="text" name="name">
  <input type="email" name="email">
  <input type="submit" name="register">
</form>
```

When a form is submitted, its inputs become available as variables in PHP, with the variable having the same name as the **'name'** attribute used in the form.



The screenshot shows a web form with two text input fields labeled 'Name' and 'Email', and a 'Submit' button. Below the form, the PHP output is displayed as an array:

```
Array
(
    [name] => Dave Jones
    [email] => djones@example.com
    [register] => Submit
)
```

Values submitted using a **POST** request will be available inside the **\$_POST** SuperGlobal array.

Values submitted using a **GET** request will be available inside the **\$_GET** SuperGlobal array.

Outputting Form Data

Like any variable, we can output form variables to HTML. We must, however, treat all form variables as tainted and possibly malicious.

```
echo "<ul>";  
    echo "<li>Name: " . htmlentities($_POST['name']) . "</li>";  
    echo "<li>Name: " . htmlentities($_POST['email']) . "</li>";  
echo "</ul>";
```

Use the `htmlspecialchars()` function to escape all variables before outputting to HTML. This is the only way to protect your site and your users from Cross Site Scripting (XSS) attacks.

Tip: create a utility function to handle escaping:

```
function e($str)  
{  
    if(is_string($str)) {  
        return htmlspecialchars($str, ENT_QUOTES, "UTF-8");  
    }  
    return $str;  
}
```

Validating Submitted Data

Validation of form data is a big part of web development, and a core part of FIEO security principles

FIEO – Filter Input, Escape Output

Filter Input – Validate what the user submits to us and ensure it's what we expect. i.e., make sure text fields contain only text, that numbers are valid numbers, etc

Escape output – Sanitize user-provided data before using it. The process varies for specific contexts. So far, we have looked at only one context: outputting to HTML.

```
foreach($_POST as $key => $value) {  
    $_POST[$key] = trim($value);  
}  
  
if(empty($_POST['name'])) {  
    $errors[] = 'Name is a required.';  
}  
  
if(empty($_POST['email'])) {  
    $errors[] = 'Email is required.';  
}
```

Ensuring that users have filled in required fields is only one part of validation. We might also want to check the length of string values, whether they contain special characters, etc.

You may end up validating a single field multiple times to ensure it is valid.

Making Form Fields Sticky

Users have clearly defined expectations when using forms. One of the biggest is that the fields they have filled in don't get wiped away if they make a mistake!

```
$errors = [];  
  
if(isset($_POST['register'])) {  
    foreach($_POST as $key => $value) {  
        if(empty($_POST[$key])) {  
            $errors[] = "$key is a required field";  
        }  
    }  
}  
  
$name = (isset($_POST['name'])) ? $_POST['name'] : '';  
$email = (isset($_POST['email'])) ? $_POST['email'] : '';
```

Making Form Fields Sticky

Users have clearly defined expectations when using forms. One of the biggest is that the fields they have filled in don't get wiped away if they make a mistake!

```
<?php  
  
$name = (isset($_POST['name'])) ? $_POST['name'] : '';  
$email = (isset($_POST['email'])) ? $_POST['email'] : '';  
  
?>
```

```
<form method="post">  
  <input type="text" name="name" value="<?=$name?>">  
  <input type="email" name="email" value="<?=$email?>">  
  <input type="submit" name="register"></p>  
</form>
```

Note, this is unsafe... this is a new context that requires escaping.

05_sticky.php

Making Form Fields Sticky

Users have clearly defined expectations when using forms. One of the biggest is that the fields they have filled in don't get wiped away if they make a mistake!

```
Array
(
    [0] => email is a required field
)
```

Name

Email

Now the user gets to see errors, if she has any, but also retains any values she previously entered into the form.

This is a basic form requirement.

Escaping for HTML attributes

Users have clearly defined expectations when using forms. One of the biggest is that the fields they have filled in don't get wiped away if they make a mistake!

```
<?php
```

```
$name = (isset($_POST['name'])) ? $_POST['name'] : '';  
$email = (isset($_POST['email'])) ? $_POST['email'] : '';
```

```
?>
```

```
<form method="post">  
  <input type="text" name="name" value="<?=htmlentities($name, ENT_QUOTES)?>">  
  <input type="email" name="email" value="<?=htmlentities($email, ENT_QUOTES)?>">  
  <input type="submit" name="register">  
</form>
```

When escaping for the context of an HTML attribute, we must use the `ENT_QUOTES` flag in `htmlentities()` or `htmlspecialchars()`

Template tags (alternative syntax)

As well as alternative short syntax for echoing in HTML, PHP also has template friendly syntax for loops and conditions.

```
<?php if(count($errors) > 0) : ?>
    <ul>
        <?php foreach($errors as $error) : ?>
            <li><?=$error?></li>
        <?php endforeach; ?>
    <./ul>
<?php endif; ?>
```

- name is a required field
- email is a required field

Name

Email

Submit

Working with complex form fields

Not all form fields are as simple to process as text inputs. Checkboxes, Radio buttons, and select boxes, in particular, can be challenging.

Name

Email

Membership Level

Gender ☐ Male ☐ Female ☐ Undisclosed

Hobbies

☐ Astronomy ☐ Cycling ☐ Movies

☐ Reading ☐ Hiking ☐ Other

```
Array
(
    [name] => Dave Jones
    [email] => djones@example.com
    [level] => silver
    [gender] => male
    [hobbies] => Array
        (
            [0] => astronomy
            [1] => cycling
            [2] => movies
        )
    [register] => Submit
)
```

Values in complex fields may come through to PHP as arrays, or even multi-dimensional arrays. **How do we make those sticky?**

Making complex fields sticky

This is the code required to make the select drop-down sticky. Note, we have to check every option!

Name

Email

Membership Level

Gender ☒ Male ☐ Female ☐ Undisclosed

Hobbies

☒ Astronomy ☒ Cycling ☒ Movies

☐ Reading ☐ Hiking ☐ Other

```
<?php $level = $_POST['level'] ?? ''; ?>
```

```
<select name="level">
```

```
  <option value="">Select Level</option>
```

```
  <option <?=$level == 'bronze' ? 'selected' : ''?>
    value="bronze">Bronze</option>
```

```
  <option <?=$level == 'silver' ? 'selected' : ''?>
    value="silver">Silver</option>
```

```
  <option <?=$level == 'gold' ? 'selected' : ''?>
    value="gold">Gold</option>
```

```
  <option <?=$level == 'platinum' ? 'selected' : ''?>
    value="platinum">Platinum</option>
```

```
</select>
```

09_complex_fields_sticky.php

The checkboxes, which come through to PHP as an array, need to be checked in a slightly different way.

09_complex_fields_sticky.php

Next:
Strings