

## READ FORMS: REQUEST (POST OR GET)

An HTML form has two required attributes: action and method. The action attribute specifies the script to which the form data is passed. Method may be either GET or POST.

The information from the html forms is available in associative arrays called \$\_POST or \$\_GET (depending on the method used in the form)

In both methods you can also use \$\_REQUEST array.

### Example with POST

Main.html

```
<body>
<form action="prove.php" method="POST">
Name: <input type="text" name="name"/><br />
Surname: <input type="text" name="surname" /><br />
<input type="submit" value="Send">
</form>
</body>
```

prove.php

```
$name = $_POST['name'];
$surname = $_POST['surname'];
```

Or

```
$name = $_REQUEST['name'];
```

When the user fills in this form and hits the submit button, the prove.php page is called.

## READ FORMS: GET

### Example with GET

If method GET is used, then the variables are then displayed in the address bar.

```
Main.html  <body>
            <form action="prove.php" method="GET">
            Name: <input type="text" name="name"/><br />
            Surname: <input type="text" name="surname" /><br />
            <input type="submit" value="Send">
            </form>
            </body>
```

```
prove.php  $name = $_GET['name'];
            $surname = $_GET['surname'];
```

Because the data is contained in the address bar, variables cannot only be passed through HTML forms but also through HTML links

```
<a href="prove.php?name=Marta">link</a>
```

## READ FORMS: TEST DATA

We need to test data from a form before it is used, to confirm that it exists and has a valid value

**isset(\$variable)** → Returns true if the variable exists and has been assigned a value other than null.

**is\_null(\$variable)** → can be used to test whether a variable is set to null

**empty(\$variable)** → Returns true if the variable has an empty value— such as null, 0, false, or an empty string. It also returns true if the variable does not exist.

## TEST EXAMPLE

### index.html FILE

```
<form action="hello.php" method="POST">  
Name: <input type="text" name="name"/><br />  
<input type="submit" name="button" value="Send">  
</form>
```

### hello.php FILE

```
<?php  
    $name=$_POST['name'];  
    echo "Hello, $name";  
?>
```

Copy the code to your editor and run the application from the index.html file

## TEST EXAMPLE

- **NORMAL USE:** Insert your name in the input test and click Send

Hello, John

- **EMPTY NAME:** leave the name blank and click Send

Hello,

- **GO TO HELLO.PHP:** now go directly to the second web page by typing the URL, without going through index.html

Notice: Undefined index name....

## EXERCISE

Use `isset($variable)` `is_null($variable)` `empty($variable)` functions to solve previous problems.

(For example)

```
if(empty($name)) { echo "You cannot leave the name blank"; }
```

## SECURITY CONCERNS CROSS-SITE SCRIPTING (XSS) ATTACKS

XSS occurs when an attacker is capable of injecting a script, often Javascript, into the output of a web application

Example: we have this form

```
<form action="post.php" method="post">  
  <input type="text" name="comment" value="">  
  <input type="submit" name="submit" value="Submit">  
</form>
```

And post.php

```
echo $_POST['comment'];
```

We can type in the form this comment:

```
<script>alert("hacked")</script>
```

This is just an alert, but it could be a dangerous code!!

## SECURITY CONCERNS: PREVENT XSS ATTACK

Three measures: Data validation, Data sanitization and Output escaping.

### 1. DATA VALIDATION: Check correct data type.

#### Example1: Validates a phone

```
// phone number USA:
$phone = '1-909-466-4344';
if (preg_match('/^((1-)?\d{3})-\d{3}-\d{4}/', $phone))
{
    echo "Valid $phone";
} else {
    echo "Invalid $phone";
}
```

More about php patterns:

[https://www.w3schools.com/php/php\\_regex.asp](https://www.w3schools.com/php/php_regex.asp)

#### Example2: Validates an email address

```
if(!filter_var($_POST['email'], FILTER_VALIDATE_EMAIL))
echo "Invalid email address";
```

All the options here:

<https://www.php.net/manual/es/filter.filters.validate.php>



## SECURITY CONCERNS: PREVENT XSS ATTACK

Example3: PHP has several useful functions for determining the type of a variable. These functions can be seen in the table

Name	Description
is_array()	True if variable is an array.
is_bool()	True if variable is a bool.
is_callable()	True if variable can be called as a function.
is_float(), is_double(), is_real()	True if variable is a float.
is_int(), is_integer(), is_long()	True if variable is an integer.
is_null()	True if variable is set to null.
is_numeric()	True if variable is a number or numeric string.
is_scalar()	True if variable is an int, float, string, or bool.
is_object()	True if variable is an object.
is_resource()	True if variable is a resource.
is_string()	True if variable is a string.

## SECURITY CONCERNS: PREVENT XSS ATTACK

### Example 4: Input the correct data

When possible, use HTML5 labels to specify exactly the type of value you want to collect in input, and also the pattern label to specify its format

Month (1-12): `<input type="number" name="month" min="1" max="12"/>`

`<input type="text" pattern="[0-9]{8}[A-Z]{1}" name="dni" />`

8 numbers    1 capital letter



## SECURITY CONCERNS: PREVENT XSS ATTACK

2. DATA SANITIZATION: manipulate data to ensure it is secure.

Imagine we want a String

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

This function strips HTML and PHP tags from a string.

3. OUTPUT ESCAPING: Don't write this

```
echo "You have search: " . $_GET["query"];
```

Use always htmlspecialchars: when user-supplied data is output to the web page as text, the htmlspecialchars function should be used.

```
echo "You have search: " .  
htmlspecialchars($_GET["query"], ENT_QUOTES, 'UTF-8'));
```

Converts special characters to HTML entities

## **USING GET OR POST?**

**Security concerns: it is always better to use the POST method than GET, because with GET the parameters are passed through the URL and are more exposed to an attack**

**As a rule, you should only use GET forms if, when the form is submitted, nothing on the server changes—such as when you're requesting a list of search results. Because the search terms are in the URL, the user can bookmark the search results page and get back to it without having to type in the search term again.**

**But if, after submitting the form, a file is deleted, or a database is updated, or a record is inserted, you should use POST.**

**The primary reason for this is that if a user bookmarks the page (or presses back in their browser) it won't trigger the form submission again and potentially create a duplicate record**

## INPUT TYPE ELEMENTS

Access to different types of form input elements

### ☐ INPUT ELEMENTS

- ☐ Text
- ☐ Radio
- ☐ Checkbox
- ☐ Button
- ☐ File
- ☐ Hidden
- ☐ Password
- ☐ Submit / Reset

### ☐ SELECT

- ☐ Simple / multiple

### ☐ TEXTAREA

## INPUT TYPE ELEMENTS: TEXT

```
<label for="fname">First name</label><br>  
<input type="text" name="fname"><br>
```

First Name

```
<?PHP  
    $fname = $_REQUEST['fname'];  
    echo $fname;  
?>
```

## INPUT TYPE ELEMENTS: PASSWORD

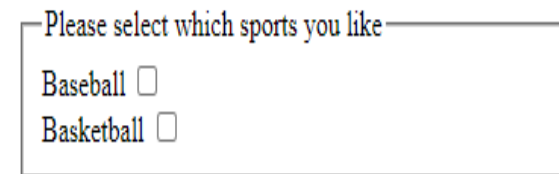
```
<input type="password" name="ssn">
```

```
<?PHP  
    $ssn = $_REQUEST['ssn'];  
    echo $ssn;  
?>
```

## INPUT TYPE ELEMENTS: CHECKBOX

Often, checkboxes are used in groups, so you can use the fieldset. However, you can check all the options.

```
<fieldset>
<legend>Please select which sports you like</legend>
<label>
Baseball
<input type="checkbox" name="baseball"><br>
</label>
<label>
Basketball
<input type="checkbox" name="basketball"><br>
</label>
</fieldset>
```



If you check the baseball checkbox, you can do this:

```
echo $_POST['baseball'];
```

However, if you uncheck it, the `$_POST['baseball']` does not exist. So verify first.

```
if isset($_POST['baseball']) echo $_POST['baseball'];
```

## INPUT TYPE ELEMENTS: RADIO

A radio group is defined by giving the same name to every radio button in the group. Only one radio button in a group can be selected at the same time.

```
<input type="radio" id="html" name="fav_language" value="HTML">  
<label for="html">HTML</label><br>  
<input type="radio" id="css" name="fav_language" value="CSS">  
<label for="css">CSS</label><br>  
<input type="radio" id="javascript" name="fav_language" value="JavaScript">  
<label for="javascript">JavaScript</label>
```


The value attribute defines the unique value associated with each radio button

☐ HTML  
☒ CSS  
☐ JavaScript



## Simple and Multiple SELECT

```
<label for="cars">Choose a car:</label>
<select id="cars" name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

Choose a car: 

Use the multiple attribute to allow the user to select more than one value.  
The name is an array:

```
<select id="cars" name="cars[]" size="4" multiple>
```

```
<?PHP
  $cars = $_REQUEST['cars'];
  foreach ($cars as $car)
    echo "$car<BR>\n";
?>
```

Choose a car: 

## MORE INFORMATION

This URL describes all the different HTML form elements.

[https://www.w3schools.com/html/html\\_form\\_elements.asp](https://www.w3schools.com/html/html_form_elements.asp)

This URL describes the different types for the HTML <input> element.

[https://www.w3schools.com/html/html\\_form\\_input\\_types.asp](https://www.w3schools.com/html/html_form_input_types.asp)