Day 1

PHP Programming Basics For Web

Course Content

PHP Programming Language

Handle Web Request and Response

Handle forms

Ajax and JSON

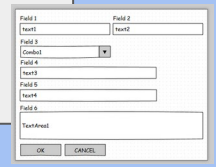
Working With Database (MySQL)

PHP Programming (Basics)

Web Concepts

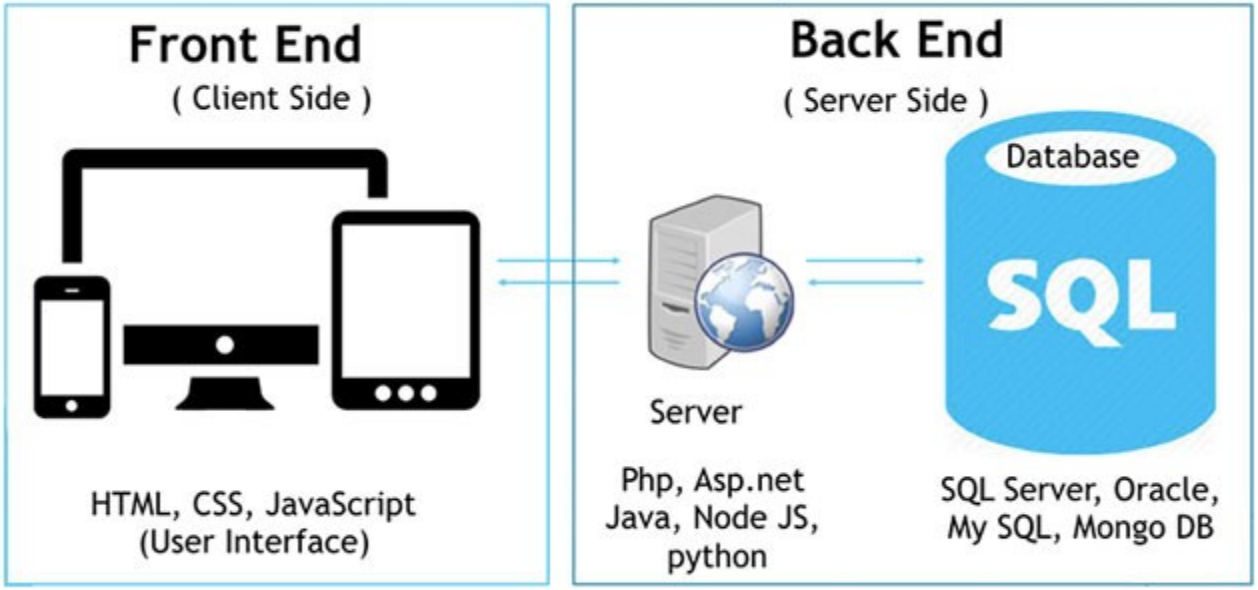
Ajax/API

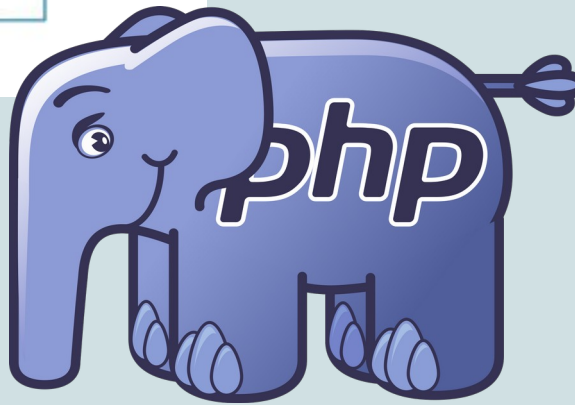
Handling Forms

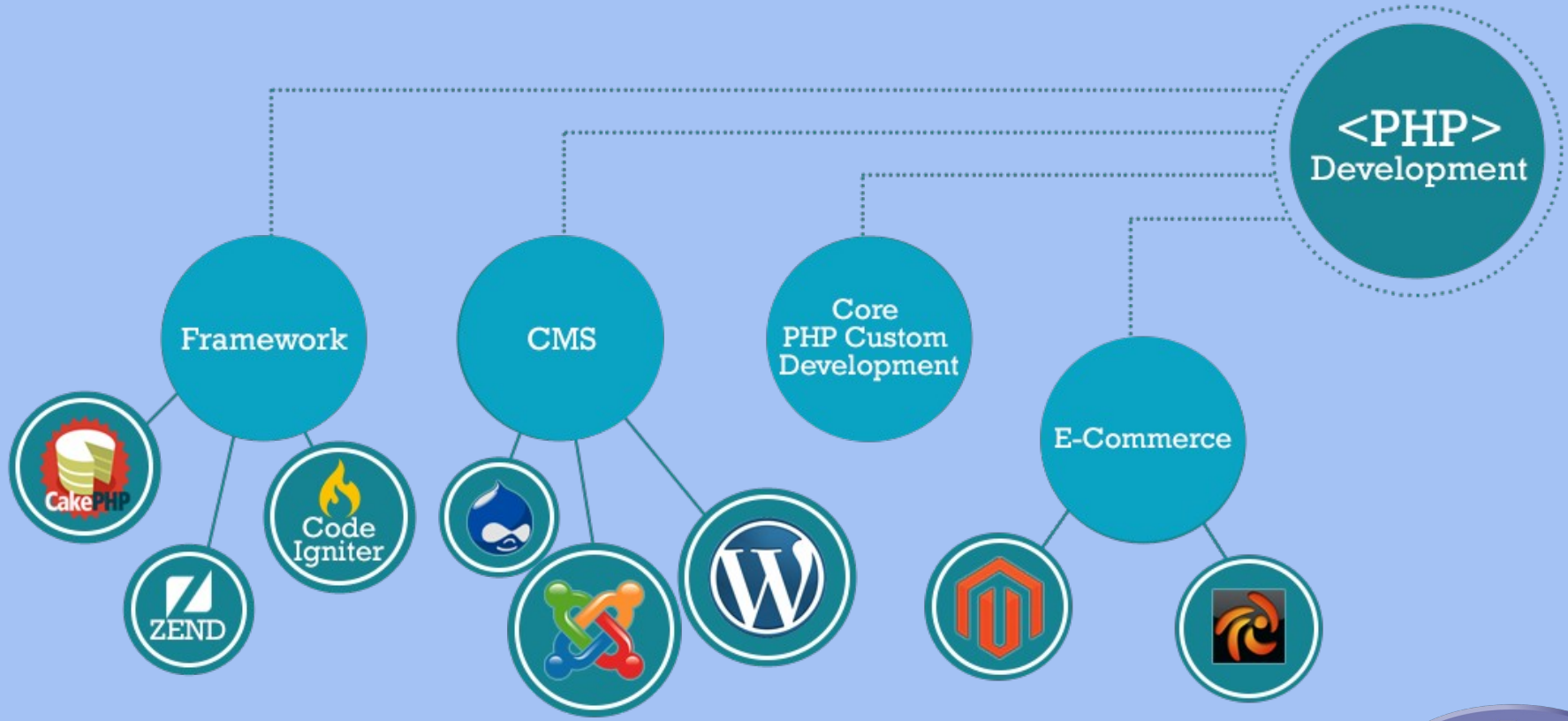


Working with Database

Full Stack Web Development






<PHP>
Development
Framework
CMS
Core
PHP Custom
Development
E-Commerce
CakePHP
Code
Igniter
ZEND
php

PHP Programming Language

- Variables, strings, arrays, and operators in PHP
- PHP in web applications
- Control structures in PHP
- Functions in PHP
- The PHP filesystem

PHP Programming Language: intro

PHP is a recursive acronym for "PHP: Hypertext Preprocessor".

PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking,

It is integrated with a number of popular databases,MySQL

PHP Syntax is C-Like.

PHP is a very loosely typed language

PHP Tags

Canonical PHP tags: **<?php PHP code goes here ?>**

Short or short-open tags: **<? PHP code goes here ?>**

Files ext .php

```
1  <html>
2     <head>
3        <title>Hello World</title>
4     </head>
5     <body>
6        <?php echo "Hello, UI!";?>
7     </body>
8  </html>
```

PHP Comments

Single line comment using //

```
// comment code
```

Multiline comment using /* */

```
<?php
   /*
     This is a comment block
   */
```

PHP Basic Syntax

PHP is quite a simple language with roots in C and Perl

- **Semicolons ;** Statements are expressions terminated by semicolons

```
$name = “track”;
```

- **The $ symbol** $variable = 123;
- **PHP is case sensitive**
- **PHP is whitespace insensitive**
- **The {} Braces make blocks**

PHP Variables

a variable is a value that can change, depending on conditions or on information passed to the program

```php
<?php
$a = 1;
$b = 2;
$c = $a + $b;
echo $c; // 3
```

Variable Naming

begins with a **letter** or underscore _ character.

can consist of numbers, letters, underscores but you cannot use characters like + , - , % , (,) . &

Variables Data Types

integers – are whole numbers

doubles – are floating-point

booleans – true or false.

NULL – has one value: NULL.

strings – are sequences of characters,

arrays – are named and indexed collections of other values.

objects – are instances of class

resources – are special variables

Constants

constant value **cannot change** during the execution of the script, constant identifiers are always **uppercase**

Only scalar data (boolean, integer, float and string) can be contained in constants.

```
<?php
   define("MINSIZE", 50);

   echo MINSIZE;
   echo constant("MINSIZE"); // same thing as the previous line
?>
```

Magic constants

Magic constant	Description
`__LINE__`	The current line number of the file.
`__FILE__`	The full path and filename of the file. If used inside an `include`, the name of the included file is returned. Some operating systems allow aliases for directories, called *symbolic links*; in `__FILE__` these are always changed to the actual directories.
`__DIR__`	The directory of the file. (Added in PHP 5.3.0.) If used inside an `include`, the directory of the included file is returned. This is equivalent to *`dirname`*`(__FILE__)`. This directory name does not have a trailing slash unless it is the root directory.
`__FUNCTION__`	The function name. (Added in PHP 4.3.0.) As of PHP 5, returns the function name as it was declared (case-sensitive). In PHP 4, its value is always lowercase.
`__CLASS__`	The class name. (Added in PHP 4.3.0.) As of PHP 5, returns the class name as it was declared (case-sensitive). In PHP 4, its value is always lowercased.
`__METHOD__`	The class method name. (Added in PHP 5.0.0.) The method name is returned as it was declared (case-sensitive).
`__NAMESPACE__`	The name of the current namespace. (Added in PHP 5.3.0.) This constant is defined at compile time (case-sensitive).

Operators

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Operators

Arithmetic Operators

-, +, *, /,%, ++, --, **

Comparison Operators

==,!=, >=,<=, >, <, <>, ===, !=

Logical (or Relational) Operators

&&, and, ||, or, !, xor

Assignment Operators

=, +=, -=, *=, /=, %=, .=

Operator precedence

Operator	Type
**	Arithmetic
++, --	Increasing/decreasing
!	Logical
*, /, %	Arithmetic
+, -	Arithmetic
<, <=, >, >=	Comparison
==, !=, ===, !==	Comparison
&&	Logical
||	Logical
=, +=, -=, *=, /=, %=, **=	Assignment

Control Structures

Conditionals

if, else, elseif, switch...case

Loops/iterations

while, do...while, for, foreach,

Working With Strings

- Single quote ‘Hello World‘ and “Hello **$world**” double quotes
  - **Escape characters**: such as **\n, \t, \”, \\ ...**
  - **Variable expanding**: **$** with string will be evaluated
- concatenate strings with [.]
- **PHP has a built-in function:**

• **strlen**: the number of characters

• **trim**: removing all the blank spaces from left and right

Working With Strings

- **strtoupper** and **strtolower:** upper or lower case respectively

- **str_replace**: replaces all occurrences of a given string by the replacement string.

- **substr**: This function extracts the string contained between the positions specified by parameters, with the first character being at position 0.

- **strpos**: This function shows the position of the first occurrence of the given string. It returns false if the string cannot be found.

- **explode**: convert string to array

Arrays

is a data structure that stores one or more similar type of values can be access with key called **index**

Numeric array − An array with a numeric index.

Associative array − An array with strings as index.

Multidimensional array − An array containing one or more arrays and values are accessed using multiple indices

Arrays

is a data structure that stores one or more similar type of values can be access with key called **index**

Numeric array − An array with a numeric index.

```
$array = [1,2,3,4,5,6];
```

Associative array − An array with strings as index.

```
$array = [“name” => “ITI”, “intake” => 39];
```

Multidimensional array − An array containing one or more arrays and values are accessed using multiple indices

```
$array = [[],[]];
```

Arrays

Initialize array: *[] or array()*

Populating arrays: Arrays are not immutable *print_r($array), unset($array[0]), $array[0] = “value”;*

Accessing arrays: print_r($array), echo $array[0]

The empty **and** isset **functions and more:**

* **Search** *in_array('needle', $array); and array_search()*
* **Sort** *sort(), rsort()*
* **Others:** array_keys(), array_values(), count(), array_merge(), explode()

Functions

A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

Creating a PHP Function:- function hello(){}

Calling a PHP Function:- hello();

File Inclusion

There are two PHP functions which can be used to included one PHP file into another PHP file.

- ***The include("path/to/file.php") Function***

- ***The require("path/to/file.php") Function***

helps in creating functions, headers, footers, or elements that can be **reused** on **multiple pages**. This will help developers to make it easy to change the layout of complete website with minimal effort. If there is any change required then instead of changing thousand of files just change included file.

File Inclusion

No file or wrong path

require is similar to **include** , except that it will produce a fatal level error on failure. When the

require fails, it will halt the script. When the include fails, it will not halt the script and only emit warning.

Files I/O (Filesystem)

The list of functions extends to over 80 different ones

Direct Reading ***file_get_contents('/path/to/file.ext');***

Direct Writing ***file_put_contents('path_to_file.ext', $data);***

More ,stdout readfile(“filename”);

Or as pointers and stream

1.fopen(‘path’, mode),2.fread(pointer,size),

3. filesize(pointer), 4. fclose(pointer)

JSON

Decoding a JSON string

The **json_decode()** function takes a JSON-encoded string as its first parameter and parses it into a PHP variable.

```php
// Returns an object (The top level item in the JSON string is a JSON dictionary)
$json_string = '{"name": "Jeff", "age": 20, "active": true, "colors": ["red", "blue"]}';
$object = json_decode($json_string);
printf('Hello %s, You are %s years old.', $object->name, $object->age);
#> Hello Jeff, You are 20 years old.

// Returns an array (The top level item in the JSON string is a JSON array)
$json_string = '["Jeff", 20, true, ["red", "blue"]]';
$array = json_decode($json_string);
printf('Hello %s, You are %s years old.', $array[0], $array[1]);
```

JSON

Encoding a JSON string

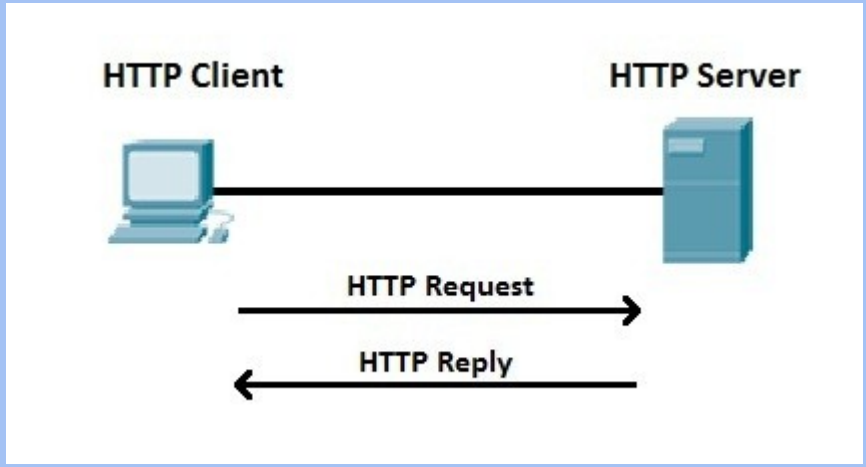
The **json_encode** function will convert a PHP array (or, since PHP 5.4, an object which implements the JsonSerializable interface) to a JSON-encoded string. It returns a JSON-encoded string on success or FALSE on failure.

```php
$array = [
    'name' => 'Jeff',
    'age' => 20,
    'active' => true,
    'colors' => ['red', 'blue'],
    'values' => [0=>'foo', 3=>'bar'],
];
```

```php
echo json_encode($array);
```

Break;

The HTTP protocol



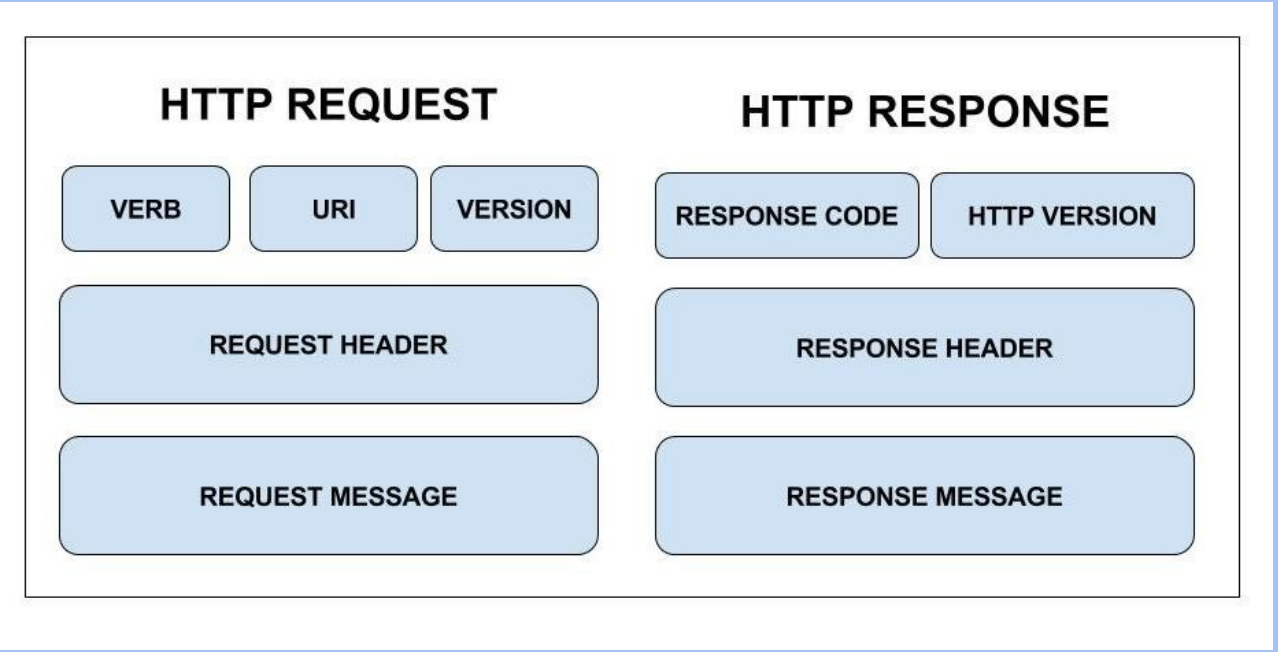
The HTTP protocol

HTTP stands for **HyperText Transfer Protocol**. As any other protocol, the goal is to allow **two** entities or nodes to **communicate** with each other. In order to achieve this, the **messages** need to be **formatted** in a way that they both understand, and the entities must follow some **pre-established rules**.

HTTP is **stateless**; that is, it treats each request **independently**,

read more

HTTP(Request/Response)Message



HTTP(Request/Response)Message

URL: The destination of message

http://localhsot/index.php

http://server.com/index.php

the URL can contain extra parameters, known as a query string.

http://server.com/index.php**?name=UI**

HTTP(Request/Response)Message

The HTTP method/Verb:

Is the verb of the message It identifies what kind of action the sender wants to perform with this message.

- **GET**: most common example is asking for a web page
- **POST**: the sender can ask the receiver to update his profile name.

There are other methods, such as **PUT**, **DELETE**, or **OPTION**, but they are less used in **web development (Regular or MVC)**, although they play a crucial role in **REST APIs**

Body

The body of the message **contains** the **content** of the message itself; for example, if the user requested a web page, the body of the response would consist of the HTML code that represents this page.

The body can contain text in any format; it can be an **HTML text** that represents **a web page**, **plain text**, the content of an **image**, **JSON**, and so on.

Headers

The metadata that the receiver needs in order to **understand** the **content** of the **message**

Accept: text/html

Cookie: name=Ui

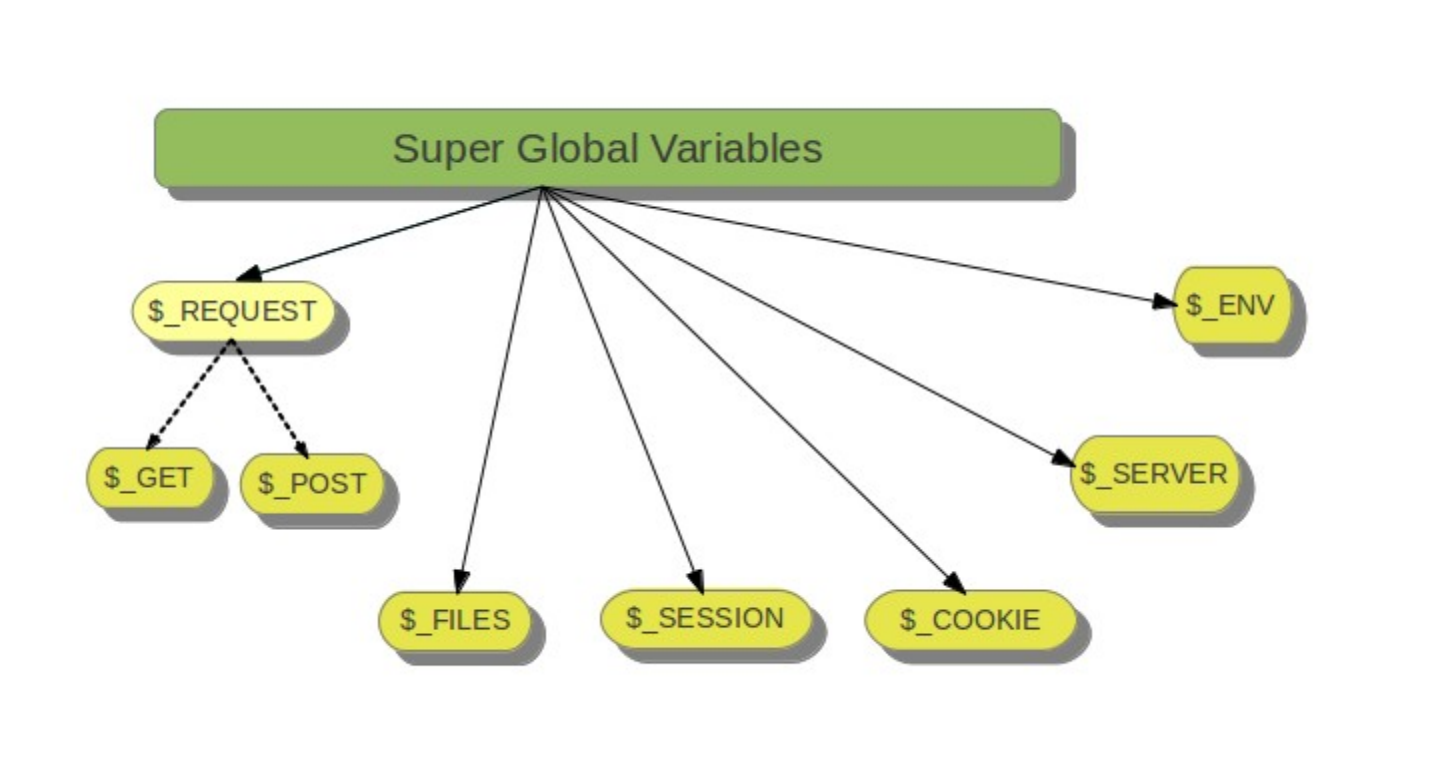
The status code

It identifies the **status** of the **request** with a **numeric code** so that browsers and other tools know how to react. For example visit page does not exists server replies 404

Common status codes are:

- **200**: The request was successful
- **401**: Unauthorized; the user does not have permission
- **404**: Page not found
- **500**: Internal server error;

Server Provide Data with Super Global



$_GET (GET Data from User)

The GET method sends the encoded user information appended to the URL requested. The page and the encoded information are separated by the ? character.

http://localhost/page.php?city=alexandria&color=blue

$_GET[“city”] // alexandria

$_GET[“color”] // blue

$_POST (GET Data from User Again)

The POST method transfers information in request body.

```php
<?php

$name = "anonymous";
if ($_POST['fname']) {
    $name = $_POST['fname'];
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
</head>
<body>
    <form method="POST" action="form.php">
        <label for="fname">First Name:</label>
        <input type="text" name="fname">
        <input type="submit" name="send" />
    </form>
    <h1>Hello <?php echo $name ?></h1>
</body>
</html>
```

Sharing Data Between pages ($_cookies)

Cookies are text files stored on the client computer

Name - The name or key of the cookie.

Value -The value to store in the cookie.

expire - A Unix timestamp representing when the cookie should expire. If set to zero the cookie will expire at

the end of the session. If set to a number less than the current Unix timestamp the cookie will expire

immediately.

Path, domain, secure, httponly

Sharing Data Between pages ($_cookies)

Set a cookie

```
setcookie("key", "value", time() + 86400, "/");
```

Read a cookie

```
echo $_COOKIE['user'];
```

Read a cookie

```
setcookie('user', '', time() - 3600, '/');
```

Sharing Data Between pages ($_sessions)

Data stored in server memory and make share data accessible across the various pages of an entire website

```php
<?php
// Starting the session
session_start();

// Storing the value in session
$_SESSION['id'] = 342;

// conditional usage of session values that may have been set in a previous session
if(!isset($_SESSION["login"])) {
    echo "Please login first";
    exit;
}
// now you can use the login safely
$user = $_SESSION["login"];

// Getting a value from the session data, or with default value,
//     using the Null Coalescing operator in PHP 7
$name = $_SESSION['name'] ?? 'Anonymous';
```

Sharing Data Between pages ($_session)

Start Session

Set session name value

Check session is setted

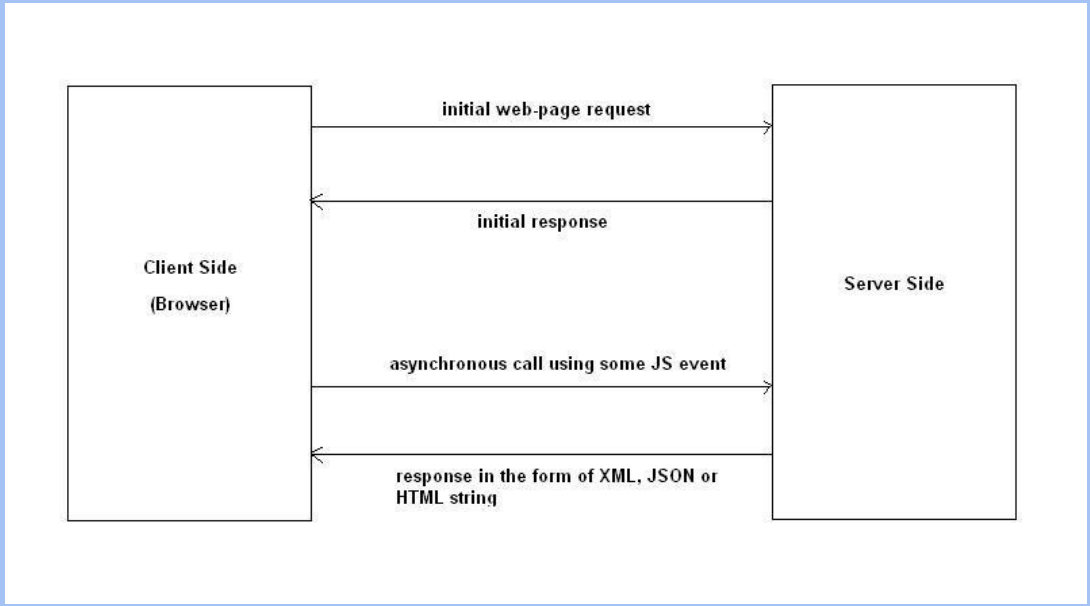
Get session values

Destroy session


We followed your API documentation to the letter! There must be a bug in the API!
No way
Our code is exactly as it should be according to the documentation... The API is badly implemented!
Check your code again and fix it
No ! You guys should check your API and fix it! Our code works as it should!
No ! YOU check YOUR code and fix it!
No, it's your problem
No, it's YOURS!
YOURS!!
NO, YOURS!!!
CommitStrip.com

AJAX

AJAX stands for Asynchronous JavaScript and XML, and it allows you to fetch content from the back-end server asynchronously, **without a page refresh.**



AJAX

Let’s quickly go through the usual AJAX flow:

1. First, the user opens a web page as usual with a synchronous request.
2. Next, the user clicks on a DOM element—usually a button or link—that initiates an asynchronous request to the back-end server. The end user won’t notice this since the call is made asynchronously and doesn’t refresh the browser. However, you can spot these AJAX calls using a tool like Firebug.
3. In response to the AJAX request, the server may return XML, JSON, or HTML string data.
4. The response data is parsed using JavaScript.
5. Finally, the parsed data is updated in the web page's DOM.

AJAX

Using Vanilla JS (See Demo)

Using JQuery (See Demo, and Documentations)