


Date
23/02/2017

The most horrible to use. Don't do this unless you know what you're doing.

Pros:

1. Very easily implemented → It takes very little to implement this, and understand.
2. Does not dirty source → Variables are outputted directly to javascript, so the DOM is not affected.

Cons:

1. Insecure → 
2. Harder to get structured data
3. Tightly couples PHP to your data logic

Example:

<script>

var data = <?php echo json_encode("42");?>;

</script>

Date
24/02/2017

int preg_match(string \$pattern, string \$subject
[, array \$matches])

Perform a regular expression match. It returns 1 if match found, 0 otherwise and FALSE if an error occurred.

Example:

```
if(preg_match("/php/i", "PHP is the") {  
    echo "A match was found";  
} else {  
    echo "A match was not found";  
}
```

Annotations:
- / or \ or #
- i case insensitive

Output:

A match was found

Delimiters can be pretty much anything that is not alpha-numeric. Backslash (\) is not allowed. The most used are generally ~, / and #.

try

number abs(mixed \$number)

Return the absolute value of number.

Example:

```
echo abs(-4.2); // 4.2  
echo abs(5); // 5  
echo abs(-5); // 5
```


Date
24/02/2017

`int printf(string format, mixed args...)`

Output a formatted string. It returns the length of the outputted string.

Example:

```
$xyz = printf("I am %d years %d month %s", 35, 2, 'old');  
echo '<br>'. $xyz;
```

Output:

I am 35 years 2 month old

25

How a web server work, what is mod_rewrite, URL rewriting and pretty-links, RegEx

To understand what mod_rewrite does you first need to understand how a web server works. A web server responds to HTTP requests. An HTTP request at its most basic level looks like this:

```
GET /foo/bar.html HTTP/1.1
```

This is the simple request of a browser to a web server requesting the URL /foo/bar.html from it. It is important to stress that it does not request a file, it requests just some arbitrary URL. The request may also look like this:

```
GET /foo/bar?baz=42 HTTP/1.1
```

This is just as valid a request for a URL, and it has more obviously nothing to do with files.

The web server is an application listening on a port, accepting HTTP requests coming in on that port and returning a response. A web server is entirely free to respond to any request in any way it sees fit/in any way you have configured it to respond. This response is not a file, it's an HTTP response which may or may not have anything to do with physical files on any disk. A web server does not have to be Apache, there are many other web server which are all just programs which run persistently and are attached to a port which respond to HTTP requests. You can write one yourself. This paragraph was intended to divorce you from any notion that URLs directly equal files, which is really important to understand 😊.

Date
24/02/2017

The default configuration of most web servers is to look for a file that matches the URL on the hard disk. If the document root of the server is set to, say, /var/www, it may look whether the file /var/www/foo/bar.html exists and serve it if so. If the file ends in ".php" it will invoke the PHP interpreter and then return the result. All this association is completely configurable; a file does not have to end in ".php" for the web server to run it through the PHP interpreter, and the URL does not have to match any particular file on disk for something to happen.

mod_rewrite is a way to rewrite the internal request handling. When the web server receives a request for the URL /foo/bar, you can rewrite that URL into something else before the web server will look for a file on disk to match it. Simple example:

RewriteEngine On

RewriteRule /foo/bar /foo/baz

This rule says whenever a request matches "/foo/bar", rewrite it to "/foo/baz". The request will then be handled as if /foo/baz had been requested instead. This can be used for various effects. For Example:

RewriteRule (.*) \$1.html