

Czy działa PHP?

Aby się o tym przekonać wystarczy w treści dokumentu XHTML umieścić:

```
<?php  
phpinfo();  
?>
```

Typy zmiennych

integer - liczby całkowite

double - liczby rzeczywiste

boolean - logiczne (TRUE lub FALSE)

string - ciągi znaków

array - tablice

object - zdefiniowana klasa

null - tylko jedna wartość (np. `$mojaZmienna = null;`)

Zmienna zachowuje swoją wartość w podwójnych cudzysłowach [„”], w pojedynczych traktowana jest jak zwykły tekst. W PHP nie można nadawać zmiennym nazw zaczynających się od cyfr.

Operatory logiczne

and - koniunkcja

&& - to samo, co „and”, ale ścisłej wiąże argumenty

! - zaprzeczenie, np. `!$zmienna`

!= - różny

== - równy

=== - identyczny (argumenty muszą być tych samych typów)

or - lub

|| - to samo, co „or”, ale ściślej wiąże argumenty

xor - albo (tylko jeden argument może być prawdziwy)

Instrukcje warunkowe

```
if (test-logiczny)
{
    jeśli-prawda;
}
else
{
    jeśli-falsz;
}
```

To samo, ale za pomocą operatora trójkrotnikowego:

test-logiczny ? jeśli-prawda : jeśli-falsz

Jeśli często występuje zagnieżdżone „else”, można zastosować „elseif”:

```
if ($day == 5)
{
    print „Dzisiaj jest piątek”;
}
elseif ($day == 4)
{
    print „Dzisiaj jest czwartek”;
}
elseif ($day == 3)
itd.
```

```
switch ($day)
{
    case 5:
        instrukcja1;
        instrukcja2;
        break;
    default:
        instrukcja;
}
```

```
while ($liczba > 10)
{
    instrukcja;
    $liczba = $liczba + 1;
}
```

```
do
{
    print „Aktualna liczba: $liczba”;
    $liczba = $liczba + 1;
}
while ($liczba < 10);
```

```
for (wyrażeniePoczątkowe; warunekZakończenia;
    wyrażenieKońcowe)
{
    instrukcja;
}
```

```
for ($x = 1, $y = 1, $z = 1; $y = 15, $z = 7; $komunikat = „Osiągnięto cel”;)
{
    $x = $x+1; $y = $y+2; $z = $z+1;
}
```

Definiowanie stałych

define(nazwaStałej, wartośćStałej);

Dołączanie plików

```
include('/ścieżka/plik.inc')
require('/ścieżka/plik.inc')
```

Od PHP5 nie ma już **istotnej** różnicy między użyciem „require” (w przypadku błędu zatrzymuje wykonywanie kodu) a „include” (w przypadku błędu ostrzega).

Ciągi znaków

\ - znak sterujący, np.:

```
$zmienna = 'Brown\'s Bicycle';
$zmienna = 'c:\\Windows\\system\\';
echo „To jest jakieś”, „zdanie”;
```

`strlen(„To jest jakieś zdanie”)` - zwraca liczbę znaków w ciągu;

Funkcje

```
function nazwaFunkcji ($argument1, $argument2)
{
    instrukcja1;
    instrukcja2;
    return ($wartość);
}
```

Funkcje mogą wywoływać się nawzajem. Zmienne w funkcjach mają tylko zasięg lokalny (są widoczne tylko w obrębie danej funkcji). Aby zmienne w funkcji miały zasięg globalny należy wewnątrz funkcji:

global *\$nazwaZmiennej*;

Aby zmienna miała wartość jaką ostatnio jej nadano (przy poprzednim wywołaniu funkcji):

static *\$nazwaZmiennej*;

sqrt(9) - pierwiastek z 9;

rand(10, 10+10) - liczba losowa pomiędzy 10 i 20;

void - funkcja nie zwraca żadnego wyniku;

Sesje

session_register() / **session_unregister()** - rejestruje / wyrejestrowuje zmienną globalną dla następnych stron, ale skrypty nie będą działać w środowisku z wyłączoną dyrektywą `register_globals` (czyli domyślnie od wersji PHP 4.2). W przyszłości w ogóle zostaną one usunięte. W zamian należy korzystać z **\$_SESSION[„nazwaZmiennej”]** = „Treść zmiennej globalnej” (wymaga wcześniejszego wywołania `session_start()`). Aby wyrejestrować zmienną globalną, należy użyć **unset(\$zmienna1, \$zmienna2, itd)**; a w przypadku bycia wewnątrz funkcji: **unset(\$GLOBALS[„zmienna”])**; Gdy chcemy usunąć pojedynczy element tablicy: **unset(\$tablica[„element1”])**;

session_destroy() - niszczy wszystkie dane skojarzone z bieżącą sesją. Nie usuwa żadnych globalnych zmiennych związanych z sesją. Nie usuwa też ciasteczka sesyjnego. Istnieje błąd PHP: po zniszczeniu sesji mogą być problemy z ponownym jej rozpoczęciem (zob. <http://bugs.php.net/32330>).

setcookie(session_name() ,„”,0,„/”); - niszczy ciasteczko sesyjne.

unset(\$_COOKIE[session_name()]); - czyści ciasteczko sesyjne.

Odczytywanie danych z formularza

Dane w formularzu są zapisywane w postaci tablicy. Aby przypisać jakąś zmienną do tablicy:

\$jakaśZmienna = \$_POST[„nazwaPolaFormularza”]; (dla PHP > 4.1)

lub:

\$HTTP_POST_VARS[„nazwaPolaFormularza”]; (dla PHP < 4.1)

lub po prostu:

\$nazwaPolaFormularza (jeśli nie działa, należy w pliku `php.ini` włączyć opcję: `register_globals = On`; od wersji 4.2 ta możliwość jest domyślnie wyłączona i nie zalecana)

Jeśli dane z formularza będzie chciała odczytać funkcja to napotkamy na problem. Funkcja widzi tylko swoje własne lokalne zmienne, a nie widzi zmiennych globalnych, tj. nazw pól formularza. Aby uzyskać dostęp do tych zmiennych należy użyć instrukcji `global`:

```
$a = 23;
```

```
function aaa(){
    global $a;
    echo $a;
}
```

Inny sposób to użycie tablicy `$GLOBALS`:

```

$a = 1;
$b = 2;
function Sum () {
    $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}

```

Interpretowanie plików z rozszerzeniem *.html jako *.php:

W pliku /etc/apache2/conf/commonapache2.conf musi znaleźć się linia:

```
AddType application/x-httpd-php .php .html
```

Każdy użytkownik może również włączyć sobie samodzielnie tę opcję poprzez umieszczenie w katalogu ze swoimi stronami pliku „.htaccess” z wpisem w środku:

```
:Location /nazwaFolderu/*.html
```

```
Use php4
```

```
:Location /*.html
```

```
Use php4
```

Buforowanie danych

Buforowanie można włączyć na 3 sposoby:

- w pliku *php.ini* wpisując: **output_buffering=On**
- w pliku *.htaccess*
- na stronie ze skryptem wpisując **ob_start()**; a kończąc wpisem **ob_end_flush()**; (kończy buforowanie).

Brak buforowania może spowodować błędy typu:

```
Warning: Cannot modify header information - headers already sent by (output started at plik.php:10) in plik.php on line 74
```

Komunikat ten może być spowodowany występowanie spacji przed “<?php” lub po “?>”. Może także być spowodowany występowaniem przed skryptem innego skryptu, który używa np. polecenia “*print*” lub “*echo*” i wysłał już informacje do przeglądarki. Rozwiązaniem problemu jest wtedy wpisanie na samej górze linii:

```
<?php ob_start(); ?>
```

Różne zmienne

\$PHP_SELF - nazwa bieżącego pliku;

\$HTTP_REFERER - adres strony, z której użytkownik wszedł na stronę ze skryptem;

\$REMOTE_ADDR - adres IP zdalnego użytkownika przeglądającego stronę;

\$REMOTE_PORT - port na maszynie zdalnego użytkownika używany do komunikacji z serwerem www;

\$REMOTE_HOST - nazwa hosta, z którego klienta otworzył połączenie.

np. Ustawienie w cookie adresu strony:

```
<?php setcookie('adresCelu','http://$HTTP_HOST$PHP_SELF'); ?>
```

Zmienne predefiniowane i tablice superglobalne

Od wersji PHP 4.2 zastąpiły tablice *\$HTTP_*_VARS*, które ze względu na kompatybilność są jeszcze obecne. Różnica między tablicami superglobalnymi a tymi używanymi wcześniej jest taka, że tablice superglobalne dostępne są w dowolnym miejscu kodu, bez potrzeby użycia instrukcji *global* przed zmienną.

Od wersji 5.0, w pliku *php.ini* domyślnie wyłączono opcję *registers_globals*. Jeśli jest ona włączona, wszystkie zmienne przesłane za pomocą metody POST są dostępne w tablicy *\$GLOBALS*. Gdy jest wyłączona, trzeba się do nich odwoływać poprzez *\$_GET*, *\$_POST*, *\$_FILES*, *\$_COOKIE*, *\$_SESSION*.

\$_SERVER - zawiera informacje o nagłówkach (header, *REMOTE_ADDR*, *HTTP_USER_AGENT*, *HTTP_REFERER*, *HTTP_HOST*), ścieżkach i lokacji skryptów; informacje te są tworzone przez serwer PHP.

\$_POST - przetrzymuje zmienne np. z pól formularza, czyli wszystko, co zostało przesłane za pomocą metody

POST, jest przetrzymywane w tej tablicy. Aby się do tych zmiennych odwołać, należy: `$_POST['nazwaPolaFormularza']`

`$_COOKIE` - odczytuje zmienne z ciasteczek.

Luki w bezpieczeństwie

Używać zmiennych z tablic `$_`, a nie zmiennych globalnych

Używając zmiennej `$_POST['identyfikator']` określamy wyraźnie źródło pochodzenia zmiennej, gdy tymczasem zmienną globalną można wprowadzić różnymi drogami.

Zły kod:

```
<?php
if($foo == 'secret')
{
    echo ('Secret Area');
}
?>
```

Dobry kod:

```
<?php
if($_POST['foo'] == 'secret')
{
    echo ('Secret Area');
}
?>
```

Operacje na bazach MySQL

Poprzedzenie polecenia znakiem "@" sprawia, że nie jest zwracana informacja o błędzie (bezpieczniejsze).

Połączenie z MySQL:

`@mysql_connect($hostname, $user, $password) or die(„Połączenie z bazą danych nie powiodło się”);`

Wybór bazy danych:

`@mysql_select_db($database) or die(„Brak dostępu do bazy danych”);`

Zapytanie:

`$zapytanie = "select nazwaKolumny from nazwaTabeli where id<10";`

`$rezultat = mysql_query($zapytanie) or die(mysql_error());`

Wynikiem zapytania nie jest jednak ciąg danych, lecz wartość true, false lub identyfikator instrukcji. Ciąg danych umieszczany jest w buforze i czeka na pobranie:

`mysql_fetch_row(rezultat) or die(mysql_error());`

`mysql_fetch_object(rezultat) or die(mysql_error());`

`mysql_fetch_array(rezultat) or die(mysql_error());`

Przykład:

```
while($row = mysql_fetch_array(rezultat))
echo $row["ID"] $row["Nazwisko"] $row["Imie"];
```

Wielokrotne pobieranie danych z buforu:

`mysql_data_seek($rezultat, 0);`

Informacje o błędach:

`mysql_query($zapytanie) or die(„Treść komunikatu w przypadku złego zapytania”);`

lub

`if (!mysql_select_db($bazaDanych))`

```
{
    print(mysql_error());
}
```

Data i czas:

\$aktualnaData= date ("Y-m-d");

\$czasUniksowy=time();

Ostatnia aktualizacja: 13 lipiec 2010.