PHP - informacje ogólne

Położenie: (nie dotyczy)

Czy działa PHP?

Aby się o tym przekonać wystarczy w treści dokumentu HTML umieścić:
<?php
phpinfo();
?>

Programowanie obiektowe

Tworzymy ogólną klasę „człowiek” (cechy, które składają się na człowieka):
class **Czlowiek** {
    public $imie;
    public $nazwisko;
    public $narodowosc = ‘polska’;
    public $plec;
    public $wiek;
    public $wzrost;
    public $zawod;
}

Teraz tworzymy konkretną instancję tego człowieka, czyli **obiekt**:
$konkretnyCzlowiek = **new Czlowiek()**;
$konkretnyCzlowiek->**imie** = ‘*Wojtek*’;
$konkretnyCzlowiek->**nazwisko** = ‘*Kowalski*’;
$konkretnyCzlowiek->**plec** = ‘*Mężczyzna*’;
$konkretnyCzlowiek->**wiek** = *35*;
$konkretnyCzlowiek->**wzrost** = *178*;
$konkretnyCzlowiek->**zawod** = ’*księgowy*’;

echo $konkretnyCzlowiek->**imie** . ‘ ‘ . $konkretnyCzlowiek->**nazwisko**;

Znak strzałki "->" łączy dany obiekt z metodą lub z atrybutem:
nazwaObiektu->metoda()
lub
nazwaObiektu->atrybut

Znak podwójnej strzałki "=>" łączy klucz tablicy z wartościami tablicy:
$tablica = **array(** x => 5, y => 8, z => 9 **)**;

Użycie zmiennej $this:
public **nazwaFunkcji()** {
    return **$this**->atrybut;
}
$obiektABC->**nazwaFunkcji()**;

Wyjaśnienie: W tym przypadku „*this*” oznacza „*obiektABC*”, bo to on wywołał tę funkcję.

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.
Uwaga: Zmienne mają swój zasięg (z reguły są dostępne w obrębie danego dokumentu. Istnieje kilka metod wysyłania zmiennych do innych dokumentów:
$_GET[‘*nazwaZmiennej*’] – przesyłanie jawne w adresie odnośnika;
$_POST[‘*nazwaZmiennej*’] – przesyłanie półjawne, stosowane w formularzach;
$_SESSION[‘*nazwaZmiennej*’] - przesyłanie niejawne (zmienne przetrzymywane tylko na serwerze PHP; dostęp do nich ma tylko ta przeglądarka, która posiada tzw. identyfikator sesji [PHPSESSID] przetrzymywany w ciasteczku, które jest usuwane przy wylogowaniu lub przy zamknięciu przeglądarki; jeśli ktoś ukradnie ciasteczko / numer – będzie miał dostęp do sesji; identyfikator sesji można także wyświetlić za pomocą *echo session_id()*).

Operatory i funkcje logiczne

and - koniunkcja
&& - to samo, co „and”, ale śliślej 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)
isset() / **!isset()** - czy zmienna istnieje / nie istnieje; jeśli zmienna ma wartość null, także będzie traktowana jako nieistniejąca; zwraca wartość typu *bool*;
empty() / **!empty()** - czy wartość zmiennej jest / nie jest pusta lub równa *false*; zwraca wartość *bool*, ale tylko wtedy, gdy zmienna istnieje (jeśli nie istnieje, niczego nie zwraca);
is_null() / **!is_null()** - sprawdza, czy wartość zmiennej jest / nie jest typu *null*; zwraca wartość typu bool;

Wartość zmiennej	***isset($var)***	***empty($var)***	***is_null($var)***
“” *(pusty string)*	*true*	*true*	*false*
” ” *(spacja)*	*true*	*false*	*false*
FALSE	*true*	*true*	*false*
TRUE	*true*	*false*	*false*
NULL	*false*	*true*	*true*
0 *(jako liczba)*	*true*	*true*	*false*
var **$zmienna** *(bez wartości)*	*false*	*true*	*true*

Instrukcje warunkowe

if *(test-logiczny)*
{
jeśli-prawda;
}
else
{
jeśli-fałsz;
}

To samo, ale za pomocą operatora trójskładnikowego:
test-logiczny ? jeśli-prawda : jeśli-fałsz

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). Więc lepiej „*require*”.
Jeśli użyjemy „*require_once*” - kod zostanie wczytany tylko raz, nawet jeśli programista przez pomyłkę umieścił wczytywanie kodu kilka razy w tym samym dokumencie.

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;

Jeśli chcemy wprowadzić za pomocą „*echo*” wiele linii, nie musimy w każdej ponawiać tej komendy. Wystarczy utworzyć etykietę (np. „*KONIEC*”):
echo<<<KONIEC
tutaj będzie kod HTML
KONIEC;

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*;
lub jako tablica:
$GLOBALS[‘*nazwaZmiennej*’**]** = ‘*wartośćZmiennej*’;
Aby zmienna miała wartość jaką ostatnio jej nadano (przy poprzednim wywołaniu funkcji):
static *$nazwaZmiennej*;

sqrt(9) - pierwiastek z 9;
rand(10, 10+10) - liczba losowa pomiędzy 10 i 20;
void - funkcja nie zwraca żadnego wyniku;

Sesje

Sesja to pojemnik na zmienne, tablica (niejawna dla użytkownika, bo przetrzymywana na serwerze), dostępna wyłącznie dla twórcy (klient posiada wyłącznie PHPSESSID zapisywane w „*cookies*”), przenoszona między stronami (globalna). Jeśli użytkownik ujawni swoje PHPSESSID, każdy będzie miał dostęp do jego sesji. Aby temu zapobiec, można wygenerować sesję zawierającą nasze IP lub napisać system sesji całkowicie od nowa według własnego pomysłu. Domyślnie, sesja rozciąga się na naszą domenę (i podkatalogi); aby utworzyć dwie odmienne sesje jednocześnie, należałoby utworzyć subdomenę.
session_start() - tworzy nową sesję (tablicę globalną) lub włącza się do istniejącej sesji (jedna domena zawiera zazwyczaj jedną sesję); deklaracja ta musi być więc zamieszczona zarówno na stronie, która tworzy sesję (tworzy zmienne globalne), jak i każdej innej stronie, która odczytuje te zmienne;
$_SESSION[„nazwaZmiennej”] = „Treść zmiennej globalnej”** (wymaga wcześniejszego wywołania *session_start()*) - rejestruje / wyrejestrowuje zmienną globalną dla następnych stron. Aby wyrejestrować zmienną globalną, należy użyć **unset($zmienna1, $zmienna2, itd);**, a w przypadku bycia wewnątrz funkcji: **unset($GLOBALS['zmienna']);. Gdy chcemy usunąć pojedynczy element tablicy: **unset($tablica['element1']);**;
session_unset() - niszczy całą sesję, wszystkie jej zmienne; podobny efekt osiągniemy zamykając przeglądarkę internetową;
session_destroy() - niszczy wszystkie dane skojarzone z bieżącą sesją. Nie usuwa żadnych globalnych zmiennych związanych z sesją. Nie usuwa też ciasteczka sesyjnego. Istnieje błąd PHP: po zniszczeniu sesji mogą być problemy z ponownym jej rozpoczęciem (zob. http://bugs.php.net/32330).
setcookie(session_name() ,"",0,"/"); - niszczy ciasteczko sesyjne.
unset($_COOKIE[session_name()]); - czyści ciasteczko sesyjne.

Jak długo trwa czas sesji (po którym następuje wylogowanie)? Określone to jest w pliku *php.ini*:
session.gc_maxlifetime=1440

Odczytywanie danych z formularza

W pliku HTML umieszczamy formularz:

```
<html>
<head>
   <style>
      label
      {display: block;}
   </style>
</head>

<body>
<form action="zamowienie.php" method="POST">
   <fieldset>
      <legend>Kolor laptopa</legend>
      <label><input type="radio" value="zielony" name="kolor" checked>Zielony</label>
      <label><input type="radio" value="czerwony" name="kolor">Czerwony</label>
      <label><input type="radio" value="niebieski" name="kolor">Niebieski</label>
   </fieldset>

   <label for="rozmiarProduktu">Wielkość matrycy:</label>
   <select id="rozmiarProduktu" name="rozmiar">
      <option value="duza">duża</option>
      <option value="srednia" selected>średnia</option>
      <option value="mala">mała</option>
   </select>

   <fieldset>
      <legend>Wyposażenie laptopa</legend>
      <label><input type="checkbox" value="instrukcja" name="instrukcja" disabled
      checked>Instrukcja obsługi</label>
      <label><input type="checkbox" value="drukarka" name="drukarka" disabled
      checked>Drukarka gratis</label>
      <label><input type="checkbox" value="hdd2tb" name="hdd2tb">HDD 2TB</label>
      <label><input type="checkbox" value="ram16gb" name="ram16gb">RAM 16GB</label>
      <label><input type="checkbox" value="inteli7" name="inteli7">Intel Core i7</label>
      <label><input  type="checkbox"  value="matrycamatowa"  name="matrycamatowa">Mato
      wa matryca</label>
      <label><input type="checkbox" value="dvd" name="dvd">Stacja DVD</label>
   </fieldset>
   <input type="submit" value="Zamówienie...">
   <!-- lub może to być <button>Zamówienie</button> -->
</form>
</body>
</html>
```

Uwaga: Element *<input>* powinien posiadać zamknięcie tylko w XHTML. Gdy *<input>* jest typu "*radio*" może wyjątkowo posiadać tę samą wartość "*name*" w kilku instancjach. Domyślną wartością elementu *<button>* jest "*type="submit*".

Dane w formularzu są zapisywane w postaci tablicy. Aby przypisać jakąś zmienną do tablicy (gdy była wysłana metodą POST):
$jakaśZmienna = $_POST['nazwaPolaFormularza'];
lub gdy była wysłana metodą GET:
$jakaśZmienna = $_GET['nazwaPolaFormularza'];

W pliku "*zamowienie.php*" odbieramy więc wybraną wartość z formularza jako:
$wybranyKolor = $_POST['kolor'];
$wybranyRozmiar = $_POST['rozmiar'];
$wybranyHDD2TB = $_POST['hdd2tb'];
$wybranyRAM16GB = $_POST['ram16gb'];
$wybranyInteli7 = $_POST['inteli7'];
$wybranyMatrycaMatowa = $_POST['matrycamatowa'];
$wybranyDVD = $_POST['dvd'];

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 uzyskała 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"];
}
```

Należy pamiętać, że w przypadku, gdy formularz ma być obsłużony przez kod PHP znajdujący się bezpośrednio na tej samej stronie, a nie przez zewnętrzny plik PHP (czyli *de facto* po naciśnięciu przycisku „*Submit*” strona ma być przeładowana), to w polu „*action= ”* nie możemy zostawiać pustego miejsca (wczytanie przez kogoś całej strony do *iframe* i wykorzystanie *$_GET*, który jest domyślny, jeśli nie zdefiniowano inaczej - rodzi możliwość jej zhakowania), nie można także tego atrybutu pominąć (HTML5 tego zabrania). Wydaje się, że najlepszą praktyką w tym przypadku jest użycie *action=”#”* lub *action=”?”* (w tym drugim przypadku nie możemy używać na stronie metody *GET*).

Odczytywanie zawartości folderu

readdir() - zawartość powinna być wczytana do tablicy.

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 php5
:Location /*.html
Use php5
```

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 występowaniem 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 5.0, w pliku php.ini domyślnie wyłączono opcję *registers_globals*. Jeśli jest ona włączona, wszystkie zmienne przesłane za pomoca 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

Należy używać zmiennych z tablic $_, a nie zmiennych globalnych

Używając zmiennej *$_POST['identyfikator']* określamy wyraźnie źródło pochodzenia zmiennej, gdy tym czasem zmienną globalną można wprowadzić różnymi drogami.
Zły kod:
```
<?php
if($foo == 'secret')
{
   echo ('Secret Area');
}
?>
```
Dobry kod:
```
<?php
if($_POST['foo'] == 'secret')
{
   echo ('Secret Area');
}
?>
```

SQL injection

Za pomocą pewnych modyfikacji, można zmienić zapytania w bazie danych, np. zamiast pytania o prawidłowy login i hasło, zadać pytanie jedynie o prawidłowy login (który oczywiście będzie hakerowi znany). W tym celu należy w formularzu logowania wpisać:
Login: **znanyPrawidlowyLogin@domena.pl' --** *(czyli: login, apostrof, spacja, dwa myślniki, spacja)*
Hasło: *dowolne*
Apostrof zamyka zapytanie po loginie, a dwa myślniki w *mySQL* oznaczają początek komentarza, a więc ignorowana jest dalsza część zapytania.

Inny sposób (nie trzeba znać nawet loginu):
Login: *dowolny*
Hasło: **' OR 1=1 --** *(czyli: apostrof, spacja, OR, spacja, 1=1, spacja, dwa myślniki, spacja)*

Jest to zapytanie, którego wynik będzie zawsze prawdziwy, gdyż jeden z członów alternatywy jest tautologią (*1=1*), a więc i cała alternatywa jest prawdziwa.

Jeszcze inny sposób polega na podaniu identyfikatora numerycznego (większość baz ma właśnie taki identyfikator kluczowy):
Login: **‘ OR id=4 --** *(czyli: apostrof, spacja, OR, spacja, id=4, spacja, dwa myślniki, spacja)*
Hasło: *dowolne*

Lekarstwem na taki atak, jest wyeliminowanie podejrzanych wpisów użytkownika w formularzu poprzez przefiltrowanie tych danych za pomocą funkcji „*htmlentities()*”, np.:
$login = htmlentities($login, ENT_QUOTES, „UTF-8”);
Funkcja ta zamienia każdy znacznik HTML, JS i innych języków – na zwykły tekst z encjami. Opcja „*ENT_QUOTES*” robi to samo także w przypadku cudzysłowów, apostrofów, myślników. Czyli zamiast wyrażenia * * będziemy mieli *<div>*, zamiast cudzysłowu - *"*, zamiast apostrofu - *'* .
Dodatkowo, tak oczyszczone dane przepuszczamy jeszcze przez funkcję „**mysqli_real_escape_string($polaczenie, $login)**”.

Tego typu rozwiązania można testować na stronie: *http://demo.testfire.net/login.jsp*
Username: admin' OR '1'='1
Password: *dowolny*

Zmiana wartości zmiennych

W tej metodzie należy najpierw zdobyć identyfikator nawiązanej przez użytkownika sesji. Możemy to zrobić tylko mając dostęp do jego przeglądarki: *Menu Firefox / Więcej narzędzi... / Narzędzia dla twórców witryn / Dane / Ciasteczka: http://adres.strony.pl / PHPSESSID: 1234567890123456*.

Następnie wysyłany podmienioną wartość jakiejś zmiennej występującą w jego skrypcie:
user@host ~ $ **curl -X POST** *http://adres.strony.pl/skrypt.php* **-H ‘Cookie: PHPSESSID=***123456789012345***’ --data ”***nazwaZmiennej=nowaWartość***”**

Uwaga: W tym przypadku „*nazwaZmiennej*” jest podana na stronie w formie jawnej jako atrybut „*name=nazwaZmiennej*” w elemencie *<input>*.

Operacje na bazach mySQL

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

Poniższe polecenia działają do wersji PHP 5.5... a w wersji 7.0 zostały całkowicie usunięte:
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:
~~mysql_query(....);~~
(dotyczy to wszystkich poleceń zaczynających się na „mysql_*”).

Połączenie z *mySQL* (od wersji PHP 5.5 włącznie; tzw. rozszerzenie *mySQLi,* czyli „*mySQL improoved*” - ulepszone... lub za pomocą całkowicie obiektowego PDO):
$link = mysqli_connect('hostname', 'user', 'password', 'dbname') or die("Error " . mysqli_error($link));
$result = mysqli_query($link, "SELECT * FROM City")

Zapytanie:
$zapytanie = “select nazwaKolumny from nazwaTabeli where id<10”;
$rezultat = mysqli_query($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:

*// Wyciąganie wierszy na podstawie ich **numerów** w tablicy (przydatne np. do pętli FOR):*
mysqli_fetch_row($rezultat) or die(mysql_error());
*// Wyciąganie wierszy na podstawie **nazw** ich kolumn:*
mysqli_fetch_assoc($rezultat) or die(mysql_error());
// Wyciąganie wierszy zarówno na podstawie nazw ich kolumn, jak i na podstawie ich numerów w tablicy:
mysqli_fetch_array($rezultat) or die(mysql_error());
mysqli_fetch_object($rezultat) or die(mysql_error());
Przykład:
while($wiersz = mysqli_fetch_array(rezultat))
echo $wiersz[“ID”] $wiersz[“Nazwisko”] $wiersz[“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();

Problemy

Strona jest całkowicie pusta, brak błędów

- Należy sprawdzić, czy nie ma opuszczonych apostrofów lub cudzysłowów.
- Przyczyną braku błędów może być także brak interpretowania kodu PHP (pliki mają rozszerzenie **.html*, a nie **.php*).
- Wyłączenie wyświetlania błędów dokonuje się w pliku "*php.ini*" na serwerze (powinny być wyłączone w wersji produkcyjnej!), natomiast w wersji deweloperskiej powinny być włączone.
- W niektórych przypadkach można włączyć raportowanie błędów z poziomu strony www:

```
<?php
    error_reporting(E_ALL);
    ini_set('display_errors', TRUE);
?>
```

Problem z czyszczeniem zmiennych przy odświeżaniu strony
Zmiany wartości zmiennych (gdy jesteśmy w sesji) występują tylko podczas wysłania ponownego zapytania **Submit** oraz przy **header(location: …)**. Nie występuje przy wciśnięciu w przeglądarce przycisku **Reload** (gdyż wtedy ponownie wczytujemy wartości zmiennych zapisanych w sesji). Przykład:
```
unset($nazwaZmiennej);
unset($_POST['nazwaZmiennej']);
```
Po kliknięciu przycisku *Reload*, ładowane są poprzednie wartości zmiennych.

Ostatnia aktualizacja: 21 marca 2024.