Kontrola sesji w PHP

- HTTP jest protokołem **bezstanowym** (ang. *stateless*) – nie utrzymuje stanu między dwoma transakcjami.
- Kontrola sesji służy do śledzenia użytkownika podczas jednej sesji na stronie internetowej (logowanie, koszyk zakupowy itp.).
- Mechanizm sesji w PHP jest zaimplementowany przez zaszyfrowany, losowy identyfikator sesji (**SID**). Może on być przechowywany w komputerze klienta w postaci pliku cookie lub przekazywany przez adres strony.
- Na serwerze są przechowywane zmienne sesji, czyli takie, które są powiązane z określoną sesją danego użytkownika i przez cały czas jej trwania można do nich uzyskać dostęp.

Ciasteczka

- Ciasteczko (ang. *cookie*) to niewielki zestaw informacji, które są zapisywane na komputerze klienta najczęściej w formie pliki tekstowego na dysku twardym.
- Aby utworzyć cookie na komputerze użytkownika wystarczy wysłać następujący nagłówek HTTP:

  Set-Cookie: NAZWA=WARTOSC; [expires=DATA;] [path=SCIEZKA;] [domain=NAZWA_DOMENY;] [secure]
- **Ciasteczko powinno być przesłane przed innymi nagłówkami.**

Ciasteczka c.d.

- **nazwa=wartość** – Wartość ta jest jedynym wymaganym atrybutem przy wysyłaniu ciasteczka. Składa się z dowolnych znaków z wyjątkiem średników, przecinków, białych spacji i ukośników (/). Jeśli zajdzie potrzeba ich użycia, najczęściej koduje się je w formacie odpowiednim dla URL (%XX), gdzie XX to kod ASCI znaku (np. %2F to zakodowana postać ukośnika, a %20 – spacji).
- **expires=data** – Atrybut expires informuje przeglądarkę o dacie wygaśnięcia danego ciasteczka. Zostanie ono usunięte z dysku, gdy jego data ważności zostanie przekroczona. Jeśli nie podano daty wygaśnięcia, to ciasteczko zostanie skanowane na koniec danej sesji.
- **path** i **domain** – Określają dla jakiej ścieżki i dla jakiej domeny ważne jest dane ciasteczko.
- **secure** – Ten parametr nie posiada wartości. Jeśli zostanie podany, to ciasteczko będzie widoczne (wysłane) tylko wtedy gdy połączenie będzie szyfrowane

Ciasteczka c.d

Właściwości ciasteczek:

- Ciasteczka o tej samej nazwie ale o innych ścieżkach będą nadpisywane.
- W celu skasowania należy wysłać ciasteczko o takiej samej nazwie i czasie wygaśnięcia z minioną datą.
- Możliwe jest wysyłanie kilku ciasteczek w jednym nagłówku (poprzez kilka atrybutów Set-Cookie).
- Istnieją limity przy zapisywaniu ciasteczek na dysku (po ich przekroczeniu przeglądarka zwykle usuwa starsze ciasteczka).
  - maksymalna liczba ciasteczek: 300.
  - maksymalna wielkość ciasteczka: 4 kilobajty.
  - maksymalna liczba ciasteczek z jednego serwera lub z jednej ścieżki: 20.
- Należy uważać z obsługą ciasteczek jeśli występuje również serwer proxy.

Tworzenie ciasteczek w PHP

- Ciasteczka z poziomu PHP mogą zostać ustawione (stworzone) za pomocą funkcji **setcookie**:

  bool **setcookie** (string $name [, string $value [, int $expire = 0 [, string $path [, string $domain [, bool $secure = false [, bool $httponly = false]]]]]])

  gdzie httponly – ciasteczka będą dostępne tylko dla protokołu http. Żadne języki skryptowe nie mają do nich dostępu.
- Kiedy do serwera przekazywane jest żądanie GET, przeglądarka w pierwszej kolejności **szuka ciasteczka przechowywanego lokalnie** i jeśli znajdzie odpowiednie to przekazuje jego wartość do serwera.

Cookie i sesje

- PHP domyślnie stosuje ciasteczka w sesji. Gdy to możliwe, w ciasteczku będzie przechowywany identyfikator sesji.
- Można również dołączyć identyfikator sesji do adresu URL. Przez ustawienie na serwerze dyrektywy session. use_trans_sid ta metoda staje się metodą domyślną, ale nie jest do zalecane bo obniża poziom bezpieczeństwa.
- Identyfikator sesji przechowywany jest w stałej **SID**. Można go przekazywać dalej w linkach – dodając go na końcu łącza:

```
<A HREF="link.php?<?php echo (SID); ?>">
```

Etapy sesji

Sesja składa się z 4 podstawowych etapów:

1. Rozpoczynanie sesji.
2. Zgłaszanie zmiennych sesji.
3. Korzystanie ze zmiennych sesji.
4. Usuwanie zmiennych i niszczenie sesji.

Rozpoczynanie sesji można zrealizować na 2 sposoby:

- Używając funkcji **session_start();**

  Sprawdza ona istnienie identyfikatora sesji i jeśli on istnieje funkcja pobierze zmienne sesji by można było z nich korzystać. Jeśli identyfikator nie istnieje to zostanie utworzony nowy wraz z tablicą sesji $_SESSION.

  Funkcja session_start() powinna być wywoływana na początku każdego skryptu korzystającego z sesji.
- Konfiguracja PHP (za pomocą opcji **session.autostart**) by automatycznie rozpoczynał sesji, gdy ktoś odwiedza daną stronę. **Wówczas jednak nie można przechowywać obiektów w sesji!**

Zgłaszanie zmiennych sesji

- Zmienne sesji są przechowywane w tablicy superglobalnej $_SESSION;
- Aby utworzyć zmienną sesji należy ustawić w tej tablicy żądany element np.
  $_SESSION[‘nick’] = 5;
- Utworzona zmienna będzie dostępna do końca sesji, chyba że zostanie wcześniej usunięte ręcznie.
- Sesja może również wygasnąć automatycznie poprzez działanie dyrektywy **session.gc_maxlifetime**.

Stosowanie i niszczenie zmiennych sesji

- Aby uzyskać dostęp do zmiennej sesyjnej korzystamy znowu z tablicy $_SESSION np.

  ```
  $war1 = $_SESSION['nick'];
  ```

- Aby bezpośrednio usunąć zmienną sesji wykorzystujemy funkcję **unset** np.

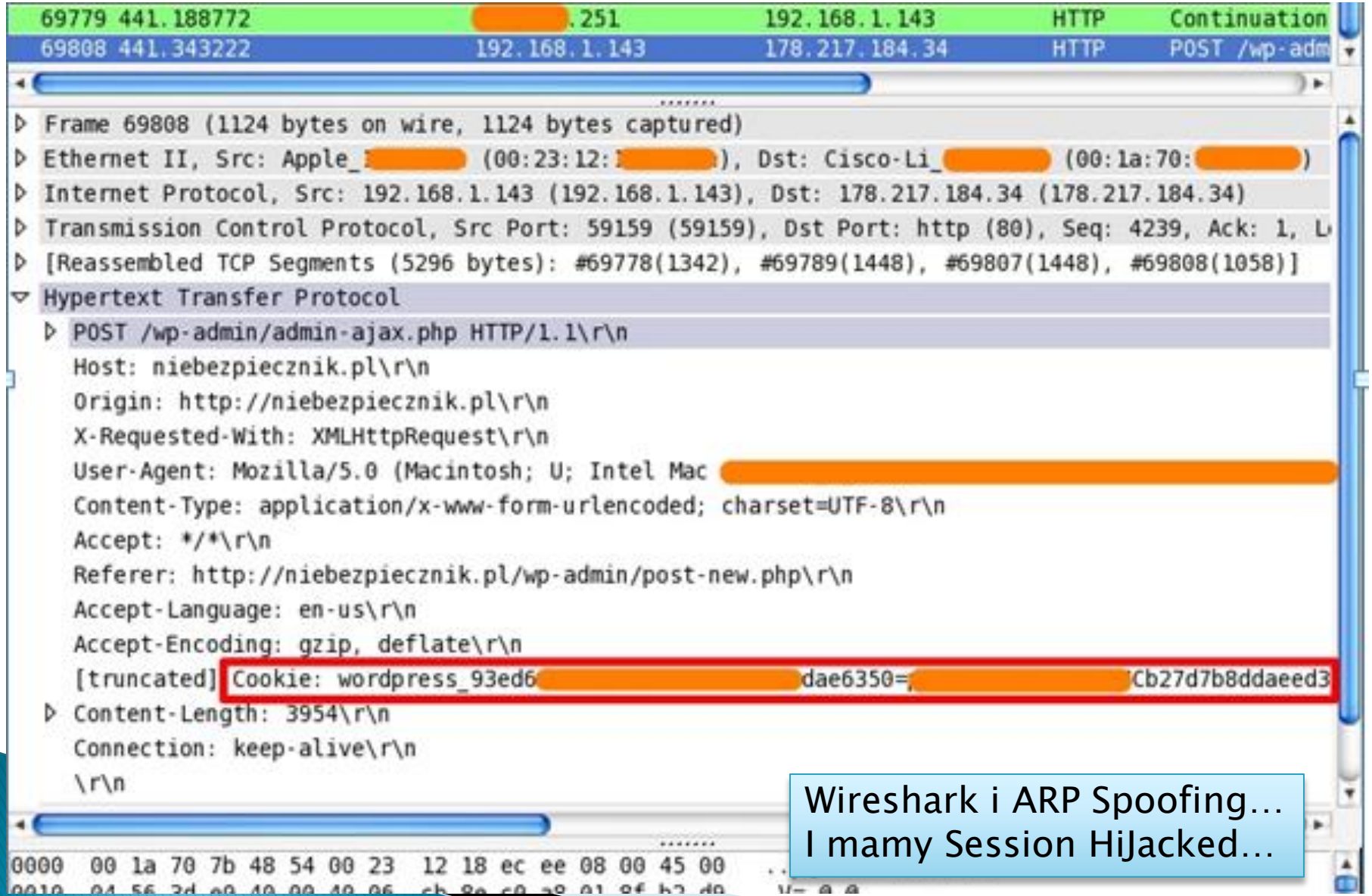
  ```
  unset($_SESSION['nick']);
  ```

- Aby usunąć wszystkie zmienne sesji należy wykonać polecenie:

  ```
  $_SESSION=array();
  ```

- Gdy wykonano wszystkie działania w sesji należy usunąć wszystkie zmienne po czym wywołać `session_destroy()` by usunąć identyfikator sesji.

Sesje... czy bezpieczne? Zależy...



Wireshark i ARP Spoofing...
I mamy Session HiJacked...

Losowanie liczb

- **int rand (void)** – losuje liczbę całkowitą z zakresu [0, getrandmax()].
- **int rand (int $min , int $max)** – losuje liczbę całkowitą z zakresu [$min, $max].

Od wersji PHP 5.2 nie trzeba specjalnie inicjalizować generatora liczb losowych – dokonuje się to automatycznie.

Tablice

- Tablica – zmienna przechowująca zbiór lub sekwencję wartości. Tablice przechowujące inne tablice to tablice wielowymiarowe.

W PHP są 2 typy tablic:

- Tablice indeksowane numerycznie
- Tablice asocjacyjne (indeks to np. ciąg znaków)

Tablice numeryczne domyślnie mają indeksy (klucze) zaczynające się od 0, ale może to być łatwo zmienione.

Tablice indeksowaną numerycznie można utworzyć na kilka sposobów. Jeśli chcemy utworzyć tablicę zawierającą określone wartości może skorzystać z konstrukcji array().

Przykład:

```
$owoce = array('Jabłko', 'Gruszka', 'Banan');
```

Inicjowanie tablic

- Można inicjować tablicę inną tablicą np.

$owoce = array('Jabłko', 'Gruszka', 'Banan');

$owoce_kopia = $owoce;

- Jeśli chcemy stworzyć tablicę zawierającą jakąś sekwencję wygodnie jest skorzystać z funkcji range(). Przykład:

$liczby = range(1, 10);

$nieparzyste = range(1, 10, 2);

$litery = range ('A', 'Z');

- Można inicjować tablice bezpośrednio z pliku czy bazy danych. Ale o tym kiedy indziej…
- Można tworzyć elementy i całe tablice „w locie" np.

$owoce = array('Jabłko', 'Gruszka', 'Banan'); //indeks od zera

$owoce[3] = „Brzoskwinia"; //Idę jeść…

Oczywiście elementu tablicy wartość odczytujemy podając nazwę tablicy i indeks w nawiasach prostokątne.

Można też tak:

$owoce[] = "Morela"; //Wtedy zostanie **wzięty największy dotąd użyty indeks** i powiększony o jeden (w naszym przypadku będzie to 3 + 1 = 4)

Dobieranie się do… elementów tablicy…

- Pętla for?

```
for ($i = 1: $i<count($owoce); $i++)
   echo $owoce[$i]." ";
```

- Pętla foreach?

```
foreach (wyrażenie_tablicowe as $wartość)
   instrukcja;
foreach (wyrażenie_tablicowe as $klucz => $wartość)
   instrukcja;
```

Przykład:

```
foreach ($owoce as $owc)
      echo „Wartość: $owc\n";

foreach ($owoce as $klucz => $owc)
      echo „Indeks: $klucz; Wartość: $owc\n";
```

A może by tak tablica asocjacyjna?

- Tablice asocjacyjne definiujemy używając operatora =>. Przykład:

```
$owoce = array('Jabłko' => 2, 'Gruszka' => 3.5, 'Banan' => 1.7);
```

- Wówczas aby dostać się do wartości trzeciego elementu (sprawdzić ile kosztują banany) należy napisać:

```
echo($owoce['Banan']);
```

- Aby przejrzeć elementy tablicy asocjacyjnej można skorzystać z pętli **foreach()** albo używając funkcji **each()** oraz **list().**

Przykład:

```
while ($element = each($owoce)){
    echo $element['key'];       //lub $element[0];
    echo $element['value'];     //lub $element[1];
}
```

Each zwraca bieżący element tablicy (a raczej parę klucz wartość jako tablicę) i ustawia wskaźnik bieżącego elementu tablicy na następny jej element. Mała uwaga: Wskaźnik bieżącego elementu tablicy zostaje na jej końcu, więc trzeba wywołać funkcję **reset($wyrażenie_tablicowe)**, aby go przestawić na początek.

```
while(list($owoc, $cena) = each($owoce))
    echo „$owoc - $cena \n";
```

Poniższy kod daje ten sam efekt, ale funkcja list() została użyta by rezultat działania funkcji each() zamienić na dwie logicznie brzmiące zmienne.

Operatory tablicowe- powtórka

Operator	Nazwa	Przykład	Rezultat
+	Unia	$x + $y	Zwraca tablicę zawierającą wszystkie elementy tablic $x i $y
==	Równość	$x == $y	Zwraca true jeśli $x i $y mają te same pary kluczy i wartości
===	Identyczność	$x === $y	Zwraca true jeśli $x i $y mają te same pary kluczy i wartości, ułożone w tej samej kolejności
!=	Nierówność	$x != $y	Zwraca true jeśli $x i $y nie są sobie równe
<>	Nierówność	$x <> $y	Zwraca true jeśli $x i $y nie są sobie równe
!=	Nieidentyczność	$x !== $y	Zwraca true jeśli $x i $y nie są

Uwaga dla Unii: Jeśli elementy z $y mają takie same klucze jak niektóre elementy już obecne w $x to nie zostaną one dodane (nadpisane).

Tablice wielowymiarowe

```
$owoce = array ( array('Jb', 'Jabłko', 2),
                 array('Gr', 'Gruszka', 3.5),
                 array('Bn', 'Banan', 1.7));
```

Albo tak:

```
$owoce = array ( array('Kod' => 'Jb', 'Nazwa' => 'Jabłko', 'Cena' => 2),
                 array('Kod' => 'Gr', 'Nazwa' => 'Gruszka', 'Cena' => 3.5),
                 array('Kod' => 'Bn', 'Nazwa' => 'Banan', 'Cena' => 1.7));
```

Albo tak:

```
$owoce = array(array()); //nie wymagane ale dobrze mieć
$owoce[0][1] = "Bn";
$owoce[0][2] = "Banan";
$owoce[0][3] = 1.7;
```

- Oczywiście nie ma ograniczenia na liczbę wymiarów…

Kto wymyśli rozsądny przykład tablicy 10cio wymiarowej?

Sortowanie tablic

1. Funkcja sort().

```
$owoce = array('Jabłko', 'Morela', 'Gruszka', 'Banan');
sort($owoce);
```

I już mamy tablicę posortowaną leksykograficznie. Funkcja sort zwraca uwagę na wielkość liter: 'A' < 'Z' < 'a'

- Można też sortować liczby w tablicy:

```
$owoce = array(100, 2, 100, 6);
sort($owoce);
```

Ale sort ma też drugi parametr opcjonalny, który może przyjmować następujące wartości:

- **SORT_REGULAR** – domyślne zachowanie
- **SORT_NUMERIC** – porównaj elementy numerycznie
- **SORT_STRING** – porównaj elementy jak łańcuchy znaków
  SORT_LOCALE_STRING – porównaj elementy jak łańcuchy znaków wykorzystując systemowe ustawienia języka i porównywania znaków (locale).

- Po co ten parametr? Jak porównujemy np. 13 i 3 to pod względem numerycznym 3 < 13, a pod względem leksykograficznym 13 < 3.

Sortowanie tablic 2

Niestety **sort()** nie zachowuje przyporządkowania do konkretnych kluczy tj.:

```
$owoce[5] = "Bn";
$owoce[7] = 1.7;
$owoce[8] = "Banan";
sort($owoce);
```

Da w wyniku uporządkowanie:

0 – Banan; 1 – Bn; 2 – 1.7

Dlatego do sortowania tablic asocjacyjnych (lub tam, gdzie nam zależy na zachowaniu przyporządkowania kluczy) należy stosować dwie inne funkcje:

- **asort($wyrażenie_tablicowe)** – sortuje według wartości elementów
- **ksort($wyrażenie_tablicowe)** – sortuje według kluczy

Wszystkie przedstawione funkcje sortują elementy tablic jednowymiarowych rosnąco. Ale mają swoje odpowiedniki zawierające literę **r** w nazwie, sortujące w porządku malejącym. Są to:

- **rsort($wyrażenie_tablicowe)**
- **arsort($wyrażenie_tablicowe)**
- **krsort($wyrażenie_tablicowe)**

Ich działanie jest analogiczne do przedstawionych wcześniej braci.

Sortowanie tablic wielowymiarowych

- Aby posortować tablice wielowymiarowe należy zdefiniować funkcję porównującą. Funkcja przyjmuje 2 parametry (dwa elementy, które będziemy porównywać ze sobą) i powinna zwrócić:
  - Wartość 0 jeśli te elementy są równe.
  - Wartość ujemną jeśli pierwszy element jest mniejszy od drugiego.
  - Wartość dodatnią jeśli pierwszy element jest większy od drugiego.

  Przykładowa funkcja porównująca sortująca po wartościach drugiego wymiaru tablicy:

```
function porownaj($x, $y) {
 if($x[1] == $y[1])
    return 0;
 else if ($x[1] < $y[1])
    return -1;
else
    return 1;
}
```

- Następnie wywołujemy funkcję **usort($wyrażenie_tablicowe, 'funkcja_porównująca') ;**

Własne kryteria sortowania c.d.

- Oczywiście funkcja **usort()** może służyć do sortowania tablic jednowymiarowych o skomplikowanej strukturze wedle kryteriów podanych przez użytkownika.
- Literka u w funkcji **usort()** odnosi się do słowa użytkownik – ponieważ to użytkownik określa funkcję porównującą.
- Funkcje asort() i ksort() również mają swoje odpowiedniki umożliwiające podanie kryteriów sortowania przez użytkownika w postaci funkcji **uasort()** i **uksort().**
- Należy zauważyć, że bardzo łatwo można przerobić funkcję porównującą tak, aby dane były uporządkowane w odwrotnym porządku (malejąco).
  Pytanie do Sali: **Co należy zrobić?**

Funkcje i operacje na tablicach

- Funkcja **shuffle($wyrażenie_tablicowe)**

Ustawia losowo kolejność elementów w tablicy.

- Funkcja **array_reverse($wyrażenie_tablicowe)**

Zwraca kopię tablicy o tych samych elementach co tablica podana jako parametr, ale w odwrotnej kolejności.

- Ładowanie danych z pliku do tablicy

```
$owoce = file("$_SERVER['DOCUMENT_ROOT']/zamowienia.txt");
for($i = 0; i < count($owoce); i++)
   $linia = explode("\t", $owoce[$i]);
```

Funkcja **file()** ładuje cały plik do tablicy. Każdy wiersz pliku staje się jednym elementem tablicy. Funkcja **explode(string separator, string ciąg, [int limit]),** dzieli wiersz na poszczególne pola (przy założeniu, że wartości w pliku są rozdzielone znakami tabulatora).

Inne operacje na tablicach

- Poruszanie się wewnątrz tablicy.

Każda tablica posiada wewnętrzny wskaźnik pokazujący aktualny element tablicy. Gdy zostaje stworzona nowa tablica, wskaźnik wskazuje na pierwszy element tablicy.

- Funkcja **current($nazwa_tablicy)** – zwraca element, na który aktualnie wskazuje wewnętrzny wskaźnik.
- **each($nazwa_tablicy)** – pobiera aktualny element i przesuwa wskaźnik o jeden do przodu.
- **next($nazwa_tablicy)** – najpierw przesuwany jest wskaźnik, a potem zwracana jest wartość aktualnego (następnego) elementu.
- **reset($nazwa_tablicy)** – ustawia wskaźnik na pierwszy element tablicy.
- **end($nazwa_tablicy)** – przesuwa wskaźnik na koniec tablicy.
- **prev($nazwa_tablicy)** – przesuwa wskaźnik o jeden element wstecz, po czym zwraca aktualny element.

Inne operacje na tablicach c.d.

- Funkcja użytkownika wykonywana dla każdego elementu tablicy

bool array_walk(array tablica, string funkcja, [mixed dane_użytkownia]);

Pierwszy parametr to tablica do przetworzenia, drugi to funkcja zdefiniowana przez użytkownika, która zostanie zastosowana do każdego elementu tablicy, a trzeci to ewentualne parametry przekazywane do funkcji użytkownika.

Funkcja array_walk pobiera indeks i wartość przechowywaną pod danym indeksem w tablicy a następnie wywołuje funkcję użytkownika w następujący sposób:

funkcja(wartość, klucz, dane_użytkownika);

Przykład:

```
function test_zmiana (&$wartosc, $klucz, $prefiks){
    $wartosc  = "$prefiks: $wartosc";
}
array_walk ($owoce, 'test_zmiana', 'owoc');
```

- array_walk_recursive działa w podobny sposób, ale od razu przetwarza **wszystkie** elementy tablicy

Inne operacje na tablicach c.d.

- Liczenie elementów tablicy
  - Funkcje **count($nazwa_tablicy)** i **sizeof($nazwa_tablicy)** zwracają liczbę elementów tablicy. Jeśli jako parametr zostanie podana tablica pusta to funkcja zwróci wartość zero.
  - **array_count_values($nazwa_tablicy)** – oblicza ile niepowtarzalnych wartości występuje w danej tablicy, oraz zwraca tablicę asocjacyjną zawierającą tabelę częstości występowania każdej wartości.
- Konwersja tablic na zmienne skalarne

Funkcja extract($wyrażenie_tablicowe, [int typ_ekstrakcji] ,[string przedrostek]) pobiera tablicę i tworzy zmienne skalarne o nazwach jak klucze w tablicy.

Przykład:

$owoce = array('Jabłko' => 2, 'Gruszka' => 3.5, 'Banan' => 1.7);

extract($owoce);

Stworzy 3 zmienne o nazwach $Jabłko, $Gruszka, $Banan.

Typ_ekstrakcji – określa sposób postępowania podczas kolizji nazwa (gdy zmienna o danej nazwie już istnieje). Można zdefiniować **przedrostek**, który zostanie dodany do nazwy zmiennej, gdy wystąpi kolizja.

Operacje na stringach

- Usuwanie białych znaków
  - string **trim**(string $lanuch [, string $lista_znakow]) – usuwa białe znaki (pionowe i poziome znaki końca wiersza [\n, \r], tabulacji [\t, \x0B], znak końca ciągu [\0], spacje) z początku i końca ciągu. Można podać własną listę znaków do usunięcia.
  - string **rtrim**(string $lanuch [, string $lista_znakow]) – jw. Ale usuwa tylko z prawej strony ciągu.
  - string **ltrim** (string $lanuch [, string $lista_znakow]) – jw. Ale usuwa tylko z lewej strony ciągu.
- Formatowanie ciągów do wyświetlenia
  - string **nl2br**(string $lancuch [,bool $is_xhtml=true]) – zamienia wszystkie znaki końca linii na znacznik <br />.
  - int **print** (string $arg) – działa tak jak echo, ale dodatkowo zwraca wartość 1 zawsze. **Więc czego używać?! I Dlaczego?!**

Zaawansowane formatowanie stringów

- int **printf** (string $format [, mixed $args [, mixed $...]]) – wyświetla sformatowany ciąg na ekranie.
- string **sprintf** (string $format [, mixed $args [, mixed $...]]) – zwraca sformatowany ciąg (nie wyświetla go).
- int **vprintf** (string $format , array $args) – jak printf, ale zamiast zmiennej liczby argumentów przyjmuje tablice argumentów.
- string **vsprintf** (string $format , array $args) jak sprintf, ale zamiast zmiennej liczby argumentów przyjmuje tablice argumentów.

Pierwszy argument obu funkcji to format w jakim dany ciąg ma być wyświetlony, a pozostałe to lista zmiennych.

Przykład:

```
printf(„Mój ulubiony owoc to $s”, $owoc);
```

Co jest tożsame z:

```
echo „Mój ulubiony owoc to $owoc”;
```

Ale funkcja printf pokazuje pazurki gdy chcemy wyświetlić wartość np. zawsze z dwoma miejscami po przecinku.

```
$cena = 3.4;
printf(„Cena wynosi $.2f”, $cena); //wyświetli 3,40
```

Jak modyfikować format

Każdy ciąg konwersji (%.2f) zbudowany jest według następującego schematu:

$[‘znak_liczby’][‘znak_dopełnienia’][-][szerokość][.dokładność]typ

Wszystkie ciągi konwersji zaczynają się od symbolu %. Aby faktycznie go wyświetlić należy go podwoić %%.

- **Znak_liczby** wymusza dodanie znaku + do każdej liczby dodatniej. Domyślnie tylko – jest dodawany do liczb ujemnych.
- **Znak_dopełnienia** będzie **rozszerzał** zmienną **do określonej szerokości** (domyślnie jest to znak spacji, ale mogą to być np. zera wiodące). Każdy inny znak dopełnienia niż 0 i spacje trzeba poprzedzić znakiem apostrofu.
- **Symbol** – oznacza **wyrównanie do lewej strony**. Domyślnie dane wyrównywane są do prawej strony.
- **Szerokość** – określa ile **minimum miejsca** w znakach należy **zarezerwować** dla zmiennej.
- **Dokładność** – musi się rozpoczynać od znaku **kropki.** Określa ona żądaną **liczbę miejsc po przecinku** do wyświetlenia. Dla ciągu znaków ten parametr określa tzw. odcięcie – ustawia maksymalny limit liczby znaków.

Typy respektowane w formacie

Typ	Opis
b	Argument traktowany jest jako integer i wyświetlany jako liczba binarna.
c	Argument traktowany jest jako integer i wyświetlany jako znak
d	Argument traktowany jest jako integer i wyświetlany jako liczba dziesiętna ze znakiem
e	Argument traktowany jest jako liczba zapisana w notacji inżynierskiej (1.2e+2)
E	J.W. ale stosując wielkie litery (1.2E+2)
u	Argument traktowany jest jako integer i wyświetlany jako liczba dziesiętna bez znaku
f	Argument traktowany jest jako float i wyświeltany jako liczba rzeczywista (z uwzględnieniem systemowych zasad locale)
F	Argument traktowany jest jako float i wyświeltany jako liczba rzeczywista (bez uwzględnianiania systemowych zasad locale)
g	Krótka forma dla e lub f (w zależności od postaci argumentu)
G	Krótka forma dla E lub F (w zależności od postaci argumentu)
o	Argument traktowany jest jako integer i wyświetlany jako liczba ósemkowa
s	Argument traktowany jest i wyświetlany jako string
x	Argument traktowany jest jako integer i wyświetlany jako liczba szesnastkowa (używając małych liter).
X	Argument traktowany jest jako integer i wyświetlany jako liczba szesnastkowa (używając wielkich liter).

Zmiana wielkości liter w ciągu

- string **strtoupper**(string $lancuch) – zwraca ciąg zamieniony na wielkie litery.
- string **strtolower**(string $lancuch) – zwraca ciąg zamieniony na małe litery.
- string **ucfirst**(string $lancuch) – Pierwsza litera ciągu jest zmieniana na wielką.
- string **ucwords**(string $lancuch) – Pierwsze litery każdego wyrazu są zamienia na wielkie.

Wszystkie funkcje działają tylko dla liter alfabetu wyznaczonych przez aktualne ustawienie locale. Przykładowo w angielskim locale polskie ogonki nie są traktowane jako litery alfabetu.

Stringi i baza danych…

- string **addslashes** (string $str) – zwraca ciąg, w którym wszystkie znaki aportforu (‘), cudzysłowia (”), backslasha (\), NUL (wartość NULL) poprzedzone będą dodatkowym znakiem ukośnika (backslasha).
- string **addcslashes** (string $str , string $charlist) – j.w. ale można zdefiniować własny zestaw znaków jakie należy „uciekać”.
- string **stripslashes** (string $str) – usuwa wprowadzone przez funkcję addslashes dodatkowe znaki ukośników.
- string **stripcslashes** (string $str) – j.w. ale pomija znaki specjalnie jak \n, \r, \t, reprezentację szesnastkową i ósemkową.
- int **get_magic_quotes_gpc** (void) – zwraca 0 jeśli mechanizm „magic_quotes_gpc” jest wyłączony, oraz 1 w przeciwnym wypadku.

Gdy jest włączony mechanizm magic_quotes_gpc **automatycznie** dokonuje on „ucieczki” od niebezpiecznych znaków pochodzących ze źródła GET, POST lub COOKIE. Mechanizm ten ma tatus *DEPRECATED* od php 5.3.0, więc dobrze jest badać jego stan przed zastosowaniem add lub stripslashes.

Inne sposoby radzenia sobie z takimi sytuacjami dla baz danych zostaną przedstawione później…

Łączenie i rozdzielanie ciągów

- array **explode** (string $separator , string $lancuch [, int $limit]) – zwraca tablicę, w której każdy element to podciąg danego łańcucha, rozbitego używając znak separatora.
- string **implode** ([string $klej], array $czesci) – odwraca działanie funkcji explode. Parametr klej oznacza ciąg, który zostanie wykorzystany przy łączeniu elementów (domyślnie to pusty ciąg)
- **join()** – alias dla funkcji implode.
- string **strtok** (string $ciag , string $separator) – rozbija ciąg na podciągi w zależności od separatora. Jednakże funkcja ta pobiera te podciągi jeden po drugim. Tylko raz należy wywołać funkcję w tej formie.
- string **strtok** (string $separator) – dalsze wywołania strtok powinny przyjmować wyłącznie separator. Wówczas funkcja modyfikuje wewnętrzny wskaźnik tablicy. Gdyby została wywołana funkcja w formie **strtok** (string $ciag , string $separator) to wskaźnik wróciłby na początek i znowu otrzymalibyśmy pierwszy element ciągu.

Podciągi

- **string substr (string $string , int $start [, int $length])** – pozwala na dostęp do podciągu zawartego między punktem startu i końca (który jest określony jako liczba znaków do pobrania od punktu startowego).
- Jeśli **start** ma wartość większą od zera to podciąg zaczynać się będzie od tego właśnie miejsca.
- Jeśli **start** jest mniejszy od zera, to start wyznacza miejsce licząc od końca ciągu.
- Jeśli wartość **start** jest mniejsza lub równa długości ciągu, funkcja substr zwraca false.
- Jeśli **length** jest większa od zera – zostanie zwrócony podciąg o długości co najwyżej równej wartości tego parametru.
- Jeśli **length** jest mniejsza od zera to tyle znaków zostanie pominiętych od końca ciągu.
- Jeśli **length** jest równe 0 to funkcja substr zwraca false.
- Jeśli **lenght** zostanie pominięte – podciąg będzie miał długość licząc od punktu startowego do końca bazowego ciągu.

Porównywanie podciągów

- int **strcmp** (string $str1 , string $str2) –
  - Zwraca < 0 jeśli *str1* jest mniejsze niż *str2*;
  - > 0 jeśli *str1* jest większe niż *str2*
  - 0 jeśli są one równe.
  - Nie uwzględnia ona wielkości liter
- int **strcasecmp** (string $str1 , string $str2)
  - Jak wyżej tyle, że uwzględnia wielkość liter.
- int **strnatcmp** (string $str1 , string $str2)
- int **strnatcasecmp** (string $str1 , string $str2)

Obie powyższe funkcje posiadają bardziej naturalny algorytm porównywania:

strcmp da : 1, 10, 12, 2

strnatcmp da :1, 2, 10, 12

- int **strlen** (string $string) – zwraca długość podanego ciągu lub 0 jeśli jest to ciąg pusty.

Odnajdywanie pozycji i zamiana podciągów

- int **strpos** (string $stóg , mixed $igła [, int $offset = 0]) – działa podobnie jak strstr(), ale zwraca pozycję pierwszego wystąpienia igły w stogu. $offset definiuje od której pozycji w ciągu należy szukać igły, ale zwrócona wartość jest nadal w relacji do początku całego ciągu.
- Jeśli funkcja nie znajdzie niczego zwraca **FALSE.**

  Pierwszy znak ciągu jest na pozycji zero!
- int **strrpos** (string $stóg , string $igła [, int $offset = 0]) – zwraca pozycję **ostatniego** wystąpienia igły w stogu.
- mixed **str_replace** (mixed $co, mixed $naco , mixed $gdzie [, int &$ile_zamian]) – funkcja zmienia wszystkie wystąpienia znaków **$co** na znaki **$naco**, przeszukując ciąg **$gdzie.** Dodatkowo można określić **po ilu zamianach** funkcja **zakończy** dalsze przeszukiwanie. Funkcja zwraca nowy ciąg po dokonaniu zamian.

  Można do tej funkcji przekazać tablice znaków do zastąpienia i tablice znaków zamian.

Zamiana podciągów

- mixed **substr_replace** (mixed $gdzie , string $naco , int $start [, int $długość]) – funkcja zamienia część ciągu **$gdzie**, na podciąg **$naco**, rozpoczynając od pozycji **$start** i ewentualnie definiując określoną długość.
- Jeśli **$start** jest wartością ujemną, to początek zamiany liczy się od końca ciągu.
- Jeśli **$długość = 0**, ciąg $naco zostanie wstawiony do ciągu bazowego bez zmiany jego zawartości.
- Jeśli **$długość < 0** – reprezentuje liczbę znaków licząc od końca na których należy skończyć zamianę.
- Jeśli długość nie jest podana to zostaje zastąpiony ciąg od **$start** do jego końca.

Przykład:

```
substr_replace('abcd', 'x', 0, -4); //xabcd
substr_replace('abcd','x',0,0); //xabcd
substr_replace('abcd', 'x', 1, -3); //axbcd
substr_replace('abcd', 'x', 1, 0); //axbcd
```

Znajdywanie ciągów w ciągach

Aby odnaleźć ciąg wewnątrz innego ciągu można zastosować jedną z funkcji

- string **strstr** (string $stóg , mixed $igła [, bool $przed_igłą = false]) – szukamy wystąpień igły w stogu. Funkcja zwraca podciąg stogu, rozpoczynający się **od pierwszego wystąpienia igła do końca** ciągu bazowego. Jeśli **$przed_igłą = true** zostanie zwrócony pociąg od pierwszego elementu do igła. Funkcja **zwraca uwagę na wielkość znaków**.
- strchr() – alias do funkcji strstr();
- stristr() – jak strstr() tylko nie przywiązuje wagi do wielkości znaków.
- strrchr() – alias do funkcji stristr();