Instrukcja 2

1. Zmienne w PHP

a) Pojęcie literału - literał to wartość, którą programista na stałe przypisuje do jakiejś zmiennej w kodzie programu; nie jest stałą - stała to identyfikator!

```
$a = 1234; // 1234
$b = "tekst" // tekst
$imiona = Array('Marcin', 'Daniel', 'Magda', 'Paulina'); // 'Marcin', 'Daniel', 'Magda', 'Paulina'
$z = $a / 4; // 4 (ustalona wartość)
```

b) Dostępne typy danych

Cztery typy skalarne:
• boolean
• integer
• float (floating-point number, aka double)
• string
Dwa typy przetworzeniowe:
• array
• object
Dwa typy specjalne:
• resource
• NULL
Pseudo-typy:
• mixed
• numbers
• callback

c) Przykład poprawnych i niepoprawnych zmiennych w PHP

```
<?php
$var = 'Bob';
$Var = 'Joe';
echo "$var, $Var";      // "Bob, Joe"

$4site = 'not yet';     // niepoprawnie: nazwa zmiennej rozpoczyna się od liczby
$_4site = 'not yet';    // poprawnie: rozpoczyna się od podrkeślenia
$täyte = 'mansikka';    // poprawnie: 'ä' to znak ASCII (rozszerzony) 228.
?>
```

d) Przykład referencji zmiennych:

```
<?php
$foo = 'Bob';              // Przypisuje'Bob' do $foo
$bar = &$foo;              // Referencja $foo do $bar.
$bar = "My name is $bar";  // Podmienienie $bar
echo $bar;
echo $foo;                 // $foo również została podmieniona
?>
```

e) Podczas pisania skryptów PHP wykorzystuje się przeważnie wiele zmiennych. Jednym z mankamentów języka jest możliwość wykorzystania tzw. pustej zmiennej, która np. została zadeklarowana (tj. otrzymała odpowiednią wartość) w dalszej części kodu. PHP posiada pewne funkcje ułatwiające prześledzenie zachowania się wskazanych zmiennych, wyświetlenia ich typu oraz zawartości, dzięki czemu łatwiej można zlokalizować i naprawić nasz skrypt (tzw. funkcje dbugujące):

var_dump(mixed $wyrazenie1, …., mixed $wyrazenieN) – funkcja pełni rolę narzędzia dla programisty PHP; wyświetla informacje dotyczące podanego wyrażenia (jego typ oraz wartość);

Przykład:
$a = 42; $b = „Treść”;
var_dump($a, $b);

mixed print_r(mixed $wyrazenie, bool $return = false) - czytelne dla człowieka informacje dotyczące zmiennej; pod pierwszy parametr $wyrazenie podstawiamy zmienną, którą chcemy wyświetlić; jeżeli ustawimy parametr $return na true (domyślnie false) to funkcja zwróci nam swój wynik (czyli można go będzie zapisać do zmiennej). Przykład:
$var1 = 15; $var2 = „Treść”;
print_r($var1);
$var3 = print_r($var2, true);
echo $var3;

mixed var_export(mixed $wyrazenie, bool $return = false) – funkcja parsuje i zwraca w postaci łańcucha znakowego informacje o zmiennej; zmienną podaje się jako $wyrazenie, opcjonalnie można podać drugi parametr $return (wartość true) – wtedy wynik funkcji zostanie zwrócony w postaci zmiennej znakowej (którą można zapisać do nowej zmiennej). Przykład:
$var1 = 15; $var2 = „Treść”;
var_export($var1);
$var3 = var_export($var2, true);
echo $var3;

void unset(mixed $zmienna1,..., mixed $zmiennaN) – funkcja niszczy wskazane zmienne. Nie usuwa ich trwale z programu, ale ich wartość od czasu wykonania nadaje wskazanym zmiennym wartość NULL. (funkcja nic nie zwraca; wskazuje na to wartość void przed nazwą funkcji)

bool isset (mixed $zmienna1,..., mixed $zmiennaN) – sprawdza, czy wskazane zmienne istnieją i czy nie mają wartości NULL; jeżeli istnieją i posiadają jakieś wartości inne od NULL funkcja zwraca true; w przeciwnym wypadku false.

bool empty(mixed $zmienna) – sprawdza czy zmienna istnieje i czy posiada jakąkolwiek wartość; jeżeli istnieje i posiada wartość funkcja zwraca wynik true; w przeciwnym wypadku false. Przykład użycia nieprzypisanej zmiennej:

```php
<?php
// Niustawiona oraz nieprzypisana zmienna; na wyjściu NULL
var_dump($unset_var);

wyjście 'false'
echo($unset_bool ? "true\n" : "false\n");

// wyjście 'string(3) "abc"'
$unset_str .= 'abc';
var_dump($unset_str);

// wyjście 'int(25)'
$unset_int += 25; // 0 + 25 => 25
var_dump($unset_int);

// wyjście 'float(1.25)'
$unset_float += 1.25;
var_dump($unset_float);
?>
```

f) Innym aspektem jest zasięg używanych zmiennych. Przykładowo zmienne deklarowane w głównej części znaczniki <?php ?> są tzw. zmiennymi globalnymi – każda funkcja, bądź inny dołączony plik PHP, może je odczytać i/lub zmodyfikować. Z kolei zmienne deklarowane i używane w funkcjach „żyją” tylko w nich i zostają „zapomniane” wraz z klamrą zamykającą daną funkcję. Ponadto funkcje „nie widzą” zmiennych globalnych w ciele funkcji chyba, że zostaną w nich zadeklarowane jako globalne (przykład poniżej):

```php
<?php
$a = 1; /* zasięg globalny */

function test()
{
    echo $a; /* zasięg lokalny */
}

test();

$b = 2, $c;

function Sum()
{
    global $a, $b;

    $b = $a + $b;
    $GLOBALS['c'] = $GLOBALS['a'] + $GLOBALS['b'];
}

Sum();
echo $b + " " + $c;


?>
```

g) Zmienne statyczne pozwalają na zachowanie swojej wartości przez cały okres „życia”

wykonywanego skryptu. Celem ich zastosowania są oczywiście funkcje, w których wartość zmiennej domyślnie przechowywana jest tylko w czasie jej wykonywania. Poniższą sytuację ilustruje przykład:

```php
<?php
function test()
{
        static $int = 1+2;          // wyrażenie - błąd
    static $int = sqrt(121);  // to również jest wyrażenie
    static $a = 0;       //dobrze - deklaracja
    echo $a;
    $a++;
}
echo test();
echo test();
echo test();
?>
```

h) Tablice w programowaniu spełniają bardzo ważne funkcje. O ile do pewnego momentu każdemu początkującemu programiście wystarczy definicja kilku/kilkudziesięciu zmiennych, o tyle w dalszej swojej pracy zacznie doskwierać mu brak typu, do którego mógłby dynamicznie dodawać/modyfikować/usuwać dane. Tutaj z pomocą pojawiają się właśnie tablice. Typ sam w sobie może przypominać listę – mamy oto jej nazwę, np. $list, w której to, pod kolejnymi indeksami (kluczami, czyli jednoznacznymi identyfikatorami; inaczej klucz to po prostu nazwa naszej zmiennej na całej liście) posiadać będziemy przechowywane wartości.

Lista[0] = „1”;
Lista[1] = „2”;
Lista[2] = „3”;

0	1	2
(„1”	„2”	„3”)

LISTA['a'] = 1;
LISTA['b'] = 2;
LISTA['c'] = 3;
LISTA['tmp'] = -100;

'a'	'b'	'c'	'tmp'
(1	2	3	-100)

Powyższe przykłady starają się zilustrować jak wygląda zmienna tablicowa. Kluczami są odpowiednio 0,1,2 w pierwszym przypadku oraz 'a','b','c','tmp' w drugim. Wartości z kolei zostały przypisane operatorem przypisania.
Oczywiście to tylko proste przykłady tablic. Tablice w PHP pozwalają, w przeciwieństwie do np. C, C# czy JSP, a podobnie do chociażby JavaScript, na mieszanie typów przechowywanych danych. Oznacza to, że pod jednym kluczem może znaleźć się integer, a pod drugim string, by pod jeszcze następnym bool. Oczywiście dobrym nawykiem jest ujednolicanie typów w tablicy (np. do wolnego w pewnych przypadkach, ale uniwersalnego, typu łańcuchów znakowych). Indeksy tablic również mogą być mieszane – w jednej tablicy mogą być one zarówno znakowe jak i liczbowe. Należy przy

tym pamiętać, że domyślnie klucze są typu liczbowego. Jeżeli np. dodamy nową wartość do tablicy w taki sposób:

$list[] = „test”;

to przypisana ona zostanie do tablicy z indeksem liczbowym; liczba ta będzie równa wartości poprzedniego indeksu liczbowego powiększonego o jeden (czyli jeżeli ostatni indeks to 3, nowym będzie 4; jeżeli 0 to będzie 1; jeżeli żaden indeks liczbowy nie został znaleziony to nowy będzie miał wartość 0).
Tablice w opisywanym języku, podobnie jak w pozostałych językach (z małymi wyjątkami) można zagnieżdżać. Oznacza to, że przykładowo pod indeksem 0 można, zamiast wartości podstawowego typu, umieścić kolejną tablicę. Wygląda to tak:

$lista[0] = array(1, 2, „test”);
echo $lista[0][2]; //wyświetli „test”

Oczywiście zagnieżdżanie można wykonać dla kolejnego indeksu już zagnieżdżonej tablicy. Za każdym razem przy odwoływaniu się do wartości takiego zagnieżdżenia dokładamy kolejny kwadratowy nawias z kluczem, po którym odwołujemy się do pożądanej przez nas wartości.

Przydatne funkcje tablicowe (wybrane):

array array(mixed $value1,..., mixed $valueN) – funkcja array() tworzy tablicę z podanych jako parametry wartości, po czym zwraca tą tablicę (wartość można przypisać do zmiennej). Podczas tworzenia można nadać pożądane wartości indeksów poprzez formułę: nazwa_klucza => wartość, przykładowo

array ('test' => „to jest test”, 42 => „pod indeksem 42”, 'x' => 12)
array (1, 2, „test”, 'xan' => 42, true, 15 => „aaaa”)

array array_values(array $tablicaWejsciowa) – funkcja zwraca tablicę ze wszystkimi wartościami tablicy wejściowej, posortowanej wedle kluczy numerycznych; jeżeli w tablicy znajdują się wartości puste (NULL) to automatycznie są one usuwane, a numery indeksów przemienione na nowo.

array array_keys(array $wejscie, mixed $wartosc = NULL, bool $restrykcja = false) – funkcja zwraca wszystkie klucze lub klucze, pod którymi zapisana jest wartość zmiennej określona pod $wartosc. Parametr $restrykcja, gdy ustawiony jest na true, wymusza porównywanie jako ===, a nie ==

int array_push(array &$tablica, mixed $zmienna1,..., $zmiennaN) – funkcja dodaje do podanej tablicy $tablica jedną bądź więcej zmiennych (1...N) na koniec tablicy; funkcja zwraca liczbę elementów w tablicy (uwzględniając nowe elementy).

mixed array_pop(array &$tablica) – funkcja usuwa ostatni element z tablicy podanej jako parametr i zwraca ostatni istniejący element w tej tablicy; jeżeli nie pozostał już ani jeden element lub podana zmienna nie jest tablicą, zwrócona zostanie wartość NULL.

array array_slice(array $tablica, inf $przesuniecie, int $dlugosc = NULL, bool $zachowaj_klucze = false) – funkcja wycina fragment podanej tablicy $tablica i zwraca go. Zmienna $przesuniecie określa od którego elementu ma nastąpić wykrojenie tablicy; jeżeli jest to liczba pozytywna to cięcie rozpoczyna się od danego elementu; jeżeli liczba negatywna to startowy element

wyszukiwany jest od końca (np. -2 oznaczać będzie element drugi od końca). Zmienna $dlugosc jest opcjonalna – mówi ile elementów ma zostać wykrojonych; liczba ujemna tego parametru mówi, na którym elemencie od końca ma zostać zakończone wykrawane (np. -2 oznaczać będzie, że mają zostać pominięte 2 końcowe elementy). Zmienna $zachowaj_klucze, jeżeli ustawiona zostanie na true, spowoduje że po wykrojeniu tablica zwrócona będzie posiadać wartości kluczy identyczne do oryginalnej tablicy; domyślnie nowa tablica ma przeindeksowane wartości.

i) Działania na łańcuchach znakowych

Zapisy liczbowe dla ludzi są mało czytelne. Przeważnie chcemy, aby informacje kierowane do nas były w postaci tekstowej; jest to jeden z kilku preferowanych przez nas sposobów komunikacji. PHP zapisuje zmienne znakowe jako ciąg bajtowy, na każdy znak tegoż ciągu przeznaczając jeden bajt. Oznacza to, iż maksymalnie można zapiać 256 znaków (standard ASCII). Stąd brak natywnego wsparcia Unicode. Nie znaczy to, że poprzez skrypty PHP nie można wydrukować znaków w standardzie UTF. Tekst zostanie wyświetlony w formacie, w którym zakodowany został plik tekstowy z kodem. Niestety wszelkie operacje na tekście poprzez funkcje (zwiększanie/ zmniejszanie znaków, zamiana znaków, skracanie/poszerzanie łańcucha znakowego itd.) innego niż angielski (w którego skład wejdą np. polskie, arabskie, chińskie litery i/lub znaki specjalne) mogą zakończyć się niepożądanym efektem.
Łańcuch znakowy przez PHP traktowany jest jak tablica zmiennych – każdy znak (bajt) zapisywany jest jako kolejny element tablicy. Oznacza to, że słowo „osa” będzie zapisana w tablicy trójelementowej, gdzie pod indeksem 0 będzie litera 'o', pod 1 – 's', a pod 2 – 'a'.
Znaki tekstowe są zamienne na liczby. Każdy znak z tablicy ASCII posiada swój odpowiednik liczbowy, np. litera 'A' ma odpowiednik liczby dziesiętnej 65, szesnastkowej 41. Przykładowo echo „\x41” wyświetli nam właśnie znak 'A'.

Funkcje łańcuchów znakowych.

array explode(string $separator, string $ciag, int $limit) – funkcja rozbija podany $ciag (łańcuch znakowy) na tablicę ciągów znakowych; poszczególne łańcuchy rozbijane są po napotkaniu frazy separującej ($separator). Dodatkowym, opcjonalnym parametrem jest $limit; jeżeli go ustawimy (podamy mu wartość dodatnią bądź ujemną) to odpowiednio: przy dodatniej wartości określimy ile elementów ma posiadać docelowa tablica; przy wartości ujemnej – ile ostatnich ciągów znakowych ma nie zostać w niej zapisane; przy zerowej wartości – parametr przyjmuje wartość 1.

string implode(string $spoiwo, array $czesci) – funkcja scalająca tablicę $czesci w pojedynczy łańcuch znakowy. Opcjonalny jest tutaj parametr $spoiwo – domyślnie jest jego brak (zaraz po pierwszym elemencie dostawiany jest drugi, bez żadnej przerwy); gdy podamy np. znak ';' to elementy będą oddzielone od siebie średnikiem

string sprintf(string $format, mixed $argument1,..., $argumentN) – zwraca sformatowany ciąg znaków. Zmienna to standardowy ciąg znakowy, który chcemy wyświetlić. Możemy go wzbogacić dodatkowe zmienne, które chcemy sformatować i wyświetlić. Zmienną taką poprzedzamy znakiem %, po czym podajemy jakiego typu ona jest; może to być integer (d), string (s), float (f), zmiennoprzecinkowa z notacją naukową (e), w postaci szesnastkowej (x, X). Dodatkowo można wzbogacić formatowanie o np. ilość zarezerwowanych miejsc znakowych na daną zmienną, precyzję liczb zmiennoprzecinkowych (dla liczb zmiennoprzecinkowych) znaki, którymi zostanie uzupełnione puste miejsce podczas formatowania (np. gdy chcemy wyświetlić liczbę, a zarezerwowaliśmy dla niej 3 znaki to można sprawić, aby funkcja sprintf dodała w puste miejsca np. liczbę 0). W celu dokładnego zapoznania się działaniem funkcji proszę zajrzeć do dokumentacji

int ord (string $znak) – funkcja zwraca dziesiętną wartość podanego znaku w zmiennej $znak

string chr(int $liczba) – zwraca znak reprezentowany przez podaną liczbę dziesiętną

int strrpos(string $lancuch, string $szukany_ciag, int $przesuniecie = 0) – przeszukuje podany łańcuch znakowy w zmiennej $lancuch na potencjalne występowanie szukanej frazy podanej w $szukany_ciag; opcjonalnym parametrem $przesuniecie możemy wskazać, od którego znaku chcemy zacząć przeszukiwanie (pozytywna wartość) lub funkcja przeszukuje ciąg od zerowego elementu do wskazanego znaku, licząc od końca (ujemna wartość)

string substr(string $lancuch, int $start, int $dlugosc) – funkcja ze zmiennej $lancuch zwraca nowy ciąg znaków będący fragmentem oryginału; zwracany fragment rozpoczyna się od znaku pod numerem $start; można wyciąg tylko określoną ilość znaków (parametr dodatkowy $dlugosc).

Łańcuchy znakowe posiadają wiele przydatnych funkcji. Wszystkie są opisane (wraz z przykładami użycia) na stronie - http://www.php.net/manual/en/ref.strings.php .

Przykłady.

```
$a = 15;
$b = 15/7;
$c = 'Standardowy ciąg znakowy';
$d = "Zmienna formatowana z liczbą $a";

printf('Wyświetl %1$s, <br /> później %2$1.4f oraz %2$e, <br /> jeszcze %3s, a na koniec %4$d', $d, $b, $c, $a);

echo sprintf('Wyświetl %1$s, <br /> później %2$1.4f oraz %2$e, <br /> jeszcze %3s, a na koniec %4$d', $d, $b, $c, $a);

print($c . " " . $d . „<br />”);

print $c . " " . $d . „<br />”;

$e = $c;
$e .= $d;

print <<<TEST
\n $e
TEST;

print <<<'TEST2'
\n $e
TEST2;

$e[1] = „A”;
print $e;
```

• łańcuch znakowy (string) to ciąg znaków, a znak to jeden bajt
• natywnie PHP wspiera standard ASCII (256 znaków), nie Unicode
• zmienna typu string może osiągnąć wielkość 2 GB
Brak wsparcia standardu w PHO dotyczy jedynie pewnych funkcji (przykładowo porównujących

typy znakowe). Praktycznie znaki zapisywane są w tablicy bajtowej bez żadnego kodowania. Dopiero programista decyduje o wyborze preferowanej strony kodowej.

j) parsowanie zmiennych

Jednym z celów tworzenia, modyfikowania bądź zmiany wartości poszczególnych zmiennych jest najczęściej wyświetlenie ich wartości użytkownikowi naszej aplikacji (strony internetowej). Stąd bardzo ważna jest wiedza jak poprawnie wyświetlić wartości poszczególnych zmiennych wraz z pozostałym tekstem na stronie WWW.

```php
<?php
$juices = array("apple", "orange", "koolaid1" => "purple");

echo "He drank some $juices[0] juice.".PHP_EOL;
echo "He drank some $juices[1] juice.".PHP_EOL;
echo "He drank some juice made of $juice[0]s.".PHP_EOL;
echo "He drank some $juices[koolaid1] juice.".PHP_EOL;

class people {
    public $john = "John Smith";
    public $jane = "Jane Smith";
    public $robert = "Robert Paulsen";

    public $smith = "Smith";
}

$people = new people();

echo "$people->john drank some $juices[0] juice.".PHP_EOL;
echo "$people->john then said hello to $people->jane.".PHP_EOL;
echo "$people->john's wife greeted $people->robert.".PHP_EOL;
echo "$people->robert greeted the two $people->smiths.";
?>
```

Inny sposób parsowania:

```php
<?php
class foo {
    var $bar = 'I am bar.';
}

$foo = new foo();
$bar = 'bar';
$baz = array('foo', 'bar', 'baz', 'quux');
echo "{$foo->$bar}\n";
echo "{$foo->$baz[1]}\n";
?>
//I am bar.
//I am bar.
```

Powyższy przykład działa dla wersji PHP co najmniej 5.0.

2. Zmienne środowiskowe PHP.

Zmiennymi środowiskowymi nazywamy zmienne predefiniowane w języku PHP. Najczęściej są zgrupowane w odpowiedniej tabeli, w której indeksem jest nazwa zmiennej jednostkowej, tj. pojedynczej zmiennej zadeklarowanej wcześniej w kodzie skryptu. Większość predefiniowanych zmiennych (poza małymi wyjątkami) to zmienne superglobalne – zawsze mamy do nich dostęp. Poniżej zostaną przedstawione te najczęściej używane.

a) $GLOBALS - tablica zawiera wszystkie zmienne o zasięgu globalnym (tablica asocjacyjna zawierająca wszystkie zmienne globalne danej aplikacji/skryptu).

```php
<?php
function test() {
    $foo = "zmienna lokalna";

    echo '$foo o globalnym zasięgu: ' . $GLOBALS["foo"] . "\n";
    echo '$foo o lokalnym zasięgu: ' . $foo . "\n";
}

$foo = "Zmienna globalna";
test();
?>
```

b) $_SERVER — informacje na temat serwera i środowiska operacyjnego; przykładowe nazwy zmiennych serwerowych i ich zawartość:

• ’PHP_SELF’ - nazwa wykonywanego skryptu, np. http://example.com/test.php/foo.bar zwróci /test.php/foo.bar
• ’SERVER_ADDR’ - IP serwera, na którym wykonywany jest skrypt
• ’HTTP_CLIENT_IP’ - IP klienta, który wywołał skrypt
• ’HTTP_X_FORWARDED_FOR’ - jeżeli klient jest za tzw. maskaradą, zwraca adres lokalny urządzenia (np. 192.168.1.1)
• ’QUERY_STRING’ - łańcuch znakowy zapytania (jeżeli istnieje)
• ’HTTP_ACCEPT_CHARSET’ - pobiera z nagłówka kodowanie pliku skryptu
• ’HTTP_USER_AGENT’ - zawiera identyfikator przeglądarki oraz systemu, np. Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586)
• ’REMOTE_ADDR’ - IP pod którym dany skrypt jest wykonany
• ’REMOTE_PORT’ - port, na którym następuje komunikacja z użytkownikiem
• ’SCRIPT_FILENAME’ - absolutna ścieżka do wykonywanego skryptu
• ’REQUEST_URI’ - Uniform Resource Identifier, którego następstwem jest dostęp do wykonywanego skryptu
• ’SERVER_SIGNATURE’ - podpis wersji serwera oraz nazwa hosta

użycie: echo $_SERVER['REMOTE_ADDR'];

c) $_GET — zmienne z HTTP GET; przesyłane są poprzez adres WWW, np.

http://www.example.com/index.php?imie=Jacek&nazwisko=Kropka&wiek=16

w powyższym adresie mamy trzy zmienne typu GET – imię, nazwisko oraz wiek wraz z odpowiednimi wartościami.

d) $_POST — zmienne z HTTP POST; przeważnie są to wartości z pól formularza (gdy formularz posiada atrybut method='post'), a nazwami zmiennych (indeksami w tablicy) są nazwy poszczególnych elementów formularza

e) $_FILES — tablica zawiera elementy przesłane metodą HTTP POST; przesyłanie plików będzie omawiane w późniejszym terminie

f) $_REQUEST — tablica asocjacyjna; zawiera w sobie tablice $_GET, $_POST oraz $_COOKIE (HTTP Request)

g) $_SESSION — zmienne sesyjne; tablica ta zawiera wszystkie zmienne ustawiane podczas trwania sesji (tablica czyszczona jest wraz z zakończeniem sesji!). Sesja zostanie omówiona w 4 punkcie.

h) $_ENV — zmienne środowiskowe (environment)

i) $_COOKIE — HTTP Cookies; tablica zawiera ustawiane przez programistę ciastka na określony czas (po którym są kasowane)

3. Obsługa ciastek.

Pliki cookies spełniają bardzo pożyteczną rolę – pozwalają na zachowanie pewnych ustawień dotyczących przeglądanej strony (przykładowo koloru czcionki, jej wielkość czy też pewnych opcji konfiguracyjnych) dzięki czemu użytkownik nie musi za każdym razem ich zmieniać (chyba, że chce). Ciasteczka zapisują na na dysku lokalnym użytkownika strony w katalogu tymczasowym przeglądarki (tzw. cache). Oczywiście na dzień dzisiejszy istnieje obawa, iż dane z tego typu plików mogą zostać wykorzystane np. do wyświetlania nam reklam kontekstowych bądź, w skrajnych wypadkach, część danych z tychże plików może zostać wykorzystana przez inne witryny (jednak, aby do tego doszło strona musiałaby co najmniej spreparować adres trony oraz „domyślić” się nazwy zmiennej, pod którą zachowane zostało ciastko). Jak więc można wywnioskować pliki cookies są bardziej przydatne niż szkodliwe.
Przykładowe ustawienie ciasteczka:

```php
<?php
if (isset($_COOKIE['count'])) {
    $count = $_COOKIE['count'] + 1;
} else {
    $count = 1;
}
setcookie('count', $count, time()+3600);
?>
```

Przekazywanie zawartości ciastek pomiedzy skryptami

//skrypt1.php
$nazwa = „Nasza nazwa”;

```
setcookie(„ciastko”, „czekoladowe”, time() + 1800);
setcookie(„nazwa”, $nazwa);
header(„Location: skrypt2.php”);

//skrypt2.php
echo $_COOKIE['ciastko'];
echo „ „  . $_COOKIE['nazwa'];
setcookie(„ciastko”, „”, time() - 1800);
```

INFORMACJA: Proszę zapoznać się ze szczegółami użytkowania funkcji setcookie() (http://www.php.net/manual/en/function.setcookie.php) oraz funkcji header() (http://pl1.php.net/manual/en/function.header.php), w których zawarty jest sposób korzystania z ciastek.

4. Obsługa sesji.

Obsługa sesji obecna jest niemal każdej stronie WWW, która oferuje użytkownikowi możliwość personalizacji, zapisywania/pobierania danych, ich przetwarzania itp. Najlepszym przykładem tego typu serwisów są strony banków lub fora dyskusyjne. O ile w drugim wypadku sesja jedynie ułatwia przeglądanie strony (służy przeważnie do zapisania poświadczeń w celu eliminacji żądania loginu i hasła przy każdorazowym przeładowaniu strony), o tyle na stronach banku sesja jest krytycznym elementem - podczas trwania takiej sesji możliwe jest dokonywanie płatności, przegląd stanu konta bądź wprowadzenie zmian na tymże koncie.
To co odróżnia obie sesje to na pewno czas ich trwania – w przypadku tej bankowej czas życia musi być ściśle określony i w miarę krótki (przeważnie 5 minut); zabezpiecza to przed nieautoryzowanym dostępem do informacji osób trzecich (przykładowo użytkownik zamknął zakładkę ze stroną banku, ale nie wylogował się; potencjalny przestępca może otworzyć stronę i się pod niego podszyć), dane sesji z bankiem powinny być ponaddo odpowiednio zabezpieczone (najlepiej zakodowane).
Czym jest sesja? To forma autoryzacji użytkownika na stronie WWW. W przeciwieństwie do ciastek (zapisywane na komputerze klienta), dane sesyjne zapisuje się po stronie serwera. Podczas startu sesji nadany zostaje unikalny numer (SESSION ID, w skrócie SID), pod którym na serwerze gromadzone są jej dane. Rozwiązanie to zdaje się być o tyle bezpieczniejsze, że nikt nie ma dostępu do zapisywanych danych, przez co zmiana ich wartości staje się niemal niemożliwa (w przeciwieństwie do ciasteczek, które można znaleźć, a ich dane próbować podmienić).

Funkcje obsługi sesji:
bool session_start(void) – funkcja tworzy lub kontynuuje sesję użytkownika.

string session_id(string $id) – funkcja zwraca aktualny numer id aktywnej sesji w postaci ciągu znakowego. Jeżeli podamy jakikolwiek parametr pod zmienną $id to numer identyfikacyjny sesji zostanie zmieniony na właśnie podany.

bool regenerate_session_id(bool $delete_old_session = false) – funkcja zmienia id sesji na nowy (generuje ciąg znaków). Jeżeli zmienimy parametr na true wtedy nastapi skasowanie starej sesji (jej pliku)

void session_unset(void) – funkcja czyści wszystkie zmienne sesji, jakie zostały ustawione

bool session_destroy(void) – funkcja niszczy sesję oraz wszystkie zmienne jakie były do niej dowiązane.

Przykład ustawienia zmiennych sesyjnych:
$_SESSION['nowa_zmienna'] = „Jej wartość”;
$_SESSION['integer'] = 12;

Pobranie wartości zmiennych sesyjnych:

echo $_SESSION['nowa_zmienna'];

WAŻNE! Funkcje sesyjne należy wywoływać na samym początku pliku ze skryptem. Jeżeli tworzymy stronę HTML ze wstawkami PHP kod z funkcjami sesyjnymi MUSI zostać umieszczony PRZED znacznikiem <HTML>. W przeciwnym wypadku sesja może nie działać prawidłowo.

5. Parsowanie danych z formularzy HTML w PHP.

Najbardziej powszechnym elementem zbierania informacji od użytkownika stron WWW są formularze. Używane niemal wszędzie – na blogach, na forach, w sklepach internetowych, w polach kontaktowych.... Nawet jeżeli mamy tzw. koszyk w wirtualnym sklepie, który nie posiada jawnych pól formularza, dosyć często tworzony jest jako formularz (z ukrytymi polami).
W formularzu HTML wystarczy w polu 'action' wprowadzić nazwę skryptu PHP, którym chcemy sparsować dane. Każda wartość z takie formularza zostanie automatycznie zapisana w tablicy $_REQUEST (oraz $_POST/$_GET – w zależności od wybranej metody przesyłania danych), w której będzie znajdować się pod nazwą indeksu tożsamą z nazwą wpisaną pod atrybutem 'name'.

Przykład użycia:

```
<!-- PLIK FORMULARZA! -->
<form action="foo.php" method="post">
    Name:  <input type="text" name="username" /><br />
    Email: <input type="text" name="email" /><br />
    <input type="submit" name="submit" value="Submit me!" />
</form>
<!-- PLIK FORMULARZA! -->

//PLIK SKRYPTU

<?php
// od PHP 4.1.0

   echo $_POST['username'];
   echo $_REQUEST['username'];

   import_request_variables('p', 'p_');
   echo $p_username;

// od PHP 5.0.0, można wyłączyć poprzez dyrektywę register_long_arrays

   echo $HTTP_POST_VARS['username'];

// Dostępne gdy dyrektywa PHP register_globals = on. Od
// PHP 4.2.0 domyślnie register_globals = off.
// Używanie tej formy nie jest zalecane!.

   echo $username;
?>
```

Zadania do wykonania:

1. Proszę stworzyć prostą stronę logowania; jej obowiązkowymi składnikami powinien być formularz logowania (login + hasło); po kliknięciu na „zaloguj” strona powinna przesłać metodą post obie wartości do formularza sprawdzającego ich poprawność (na chwile obecną stałe w kodzie). Jeżeli będą identyczne to na stronie powinien pojawić się komunikat „Zalogowane” bądź odpowiedni komunikat o błędzie. Dobrze byłoby oczywiście stworzyć lub wykorzystać tworzony w poprzednim semestrze szablon prostej strony (w celu późniejszego dodawania funkcjonalności).
2. Proszę dodać do poprzedniej strony obsługę ciastek; po pierwsze proszę umożliwić użytkownikowi zmianę niektórych elementów strony (wedle życzenia; wszystko za pomocą plików CSS). Następnie dodać przycisk zapisu tychże ustawień; jeżeli użytkownik postanowi je zapisać to mają zachować się na okres 5 minut; proszę sprawdzić czy wszystko działa poprawnie
3. Proszę dodać obsługę sesji; sesja ma umożliwić użytkownikowi komfortową pracę ze stroną (tj. zapamiętywać czy się zalogował, czy nie). Aby sprawdzić poprawność działania proszę dodać odpowiednio kilka podstron. Przy przełączaniu się pomiędzy nimi poświadczenia (tj. tekst powitalny danego użytkownika) powinien stale wyświetlać jego login!
4. Proszę sprawdzić zawartości poszczególnych zmiennych globalnych; Następnie proszę utworzyć podstronę, na której użytkownik będzie mógł się zobaczyć informację o systemie serwerowym oraz o aktualnie używanej przeglądarce (jak widzi go serwer). Dla różnych przeglądarek proszę przygotować różne teksty (np. „Google Chrome działa na silniku Webkit”, „Firefox używa silnika Gecko” itp.)
5. Napisać licznik wprowadzonych znaków w formularzu przez użytkownika. Ponadto każdą znalezioną pustą spację proszę zamieniać na znak _