

Formatowanie kodu PHP

Włodzimierz Gajda
<http://www.gajdaw.pl>

22 listopada 2005
ver. 0.3

Streszczenie

Wśród bogactwa skryptów PHP dostępnych w sieci znajdziemy najprzeróżniejsze przykłady. Od galerii fotografii, przez fora dyskusyjne po rozbudowane aplikacje klientów pocztowych. Jeśli chcemy dołączyć do grona twórców, których skrypty cieszą się powodzeniem powinniśmy ułatwić odbiorcom analizę naszych skryptów. Jednym z kroków w tym kierunku jest stosowanie jasnych reguł formatowania kodu.

Spis treści

1. Formatowanie kodu	2
2. Zagadnienia ogólne	2
2.1. Format pliku	2
2.2. Długość linii kodu	3
2.3. Tabulatory kontra spacje	3
2.4. Średnik i wiele instrukcji w jednym wierszu	4
2.5. Puste linie w kodzie skryptu	4
3. Format kodu PHP	5
3.1. Otwarcie kodu PHP	5
3.2. Komentarze	5
3.3. Operatory	5
3.3.1. Operatory dwuargumentowe	5
3.3.2. Operatory jednoargumentowe	6
3.3.3. Operatory specjalne	6
3.3.4. Nawiasy grupujące i operator ?:	9
3.4. Instrukcje sterujące	10
3.5. Dołączanie kodu	15
3.6. Wywołanie funkcji	15
3.7. Definicja funkcji	15
3.8. Definicja klasy	16

4. Nazewnictwo	16
4.1. Stałe	16
4.2. Zmienne globalne	16
4.3. Klasy	16
4.4. Funkcje i metody	17
4.5. Pliki	17
4.6. Przykładowe adresy URL	17
5. Spójność formatu	18
6. Problemy z życia wzięte	20
7. Podsumowanie	22

1. Formatowanie kodu

Reguły formatowania kodu źródłowego programów wpływają na czytelność i niejednokrotnie pozwalają uniknąć błędów programistycznych. W poszukiwaniu jasnych zasad, jakie powinniśmy stosować pisząc skrypty w języku PHP, należy w pierwszej kolejności odwiedzić strony projektu PEAR. Na stronie internetowej pod adresem <http://pear.php.net/manual/en/standards.php> znajdziemy dosyć dokładny opis wymagań nakładanych na kod rozpowszechniany w ramach projektu PEAR. Zasady te są oparte w znacznej mierze na standardzie ustanowionym przez książkę p.t. „*Język ANSI C*” napisaną przez Briana Kernighana oraz Denisa Ritchiego.

Ponieważ opis formatu zawarty we wspomnianym dokumencie nie jest kompletny (tj. nie porusza wszystkich zagadnień dotyczących programowania w języku PHP), musimy się posłużyć dodatkowymi źródłami. Dobrym rozwiązaniem jest analiza kilku wybranych aplikacji napisanych w PHP i cieszących się uznaniem środowiska programistów PHP. Aplikacjami takimi są między innymi `phpBB`, `SquirrelMail`, `Tiki-wiki`, `Mambo` czy `typo3`. Wprowadzie format wymienionych programów jest bardzo rozbieżny i niespójny, jednakże analiza wybranych fragmentów w połączeniu ze standardami kodowania PEAR pozwoli na rozstrzygnięcie wszystkich wątpliwych kwestii.

2. Zagadnienia ogólne

2.1. Format pliku

Podstawowe wymagania dotyczą formatu pliku. Skrypty sformatowane zgodnie z zaleceniami projektu PEAR powinny być zakodowane jako `iso-8859-1`. Oznacza to, że polskie znaki diakrytyczne należy zapisywać w postaci `\xb1`, na przykład w napisie „*żółć*”:

```
echo "\xbf\xfb\xfb\xe6";
```

Ponadto znaki złamania wiersza należy zapisywać w formacie `u*ix`, czyli jako znaki LF o kodzie ASCII 10 (znaki `\n`).

Margines	Courier New 10pt	Courier New 12pt	Courier New 14pt
0,0 cm	99	82	70
0,5 cm	94	78	67
1,0 cm	89	74	64
1,5 cm	85	70	60
2,0 cm	80	66	57
2,5 cm	75	62	53

Tabela 1. Zależność liczby znaków wiersza od wielkości kroju czcionki oraz marginesów

2.2. Długość linii kodu

Kto wie, czy nie najtrudniejszym problemem do rozwiązania, jest ustalenie zasad łamania wiersza. Świadczy o tym chociażby fakt, że główne pakiety zawarte w bibliotece PEAR (z PEAR oraz DB na czele) stosują bardzo często wiersze o monstrualnej długości sięgającej nawet kilkuset znaków! Ten sam zarzut dotyczy zresztą wielu innych pakietów i aplikacji, w tym wszystkich wymienionych: `phpBB`, `SquirrelMail`, `Tiki-wiki`, `Mambo` i `typo3`.

Długość wiersza nie powinna przekraczać 80 znaków. Przemawiają za tym dwa powody. Po pierwsze analiza wiersza, który nie mieści się cały na ekranie jest utrudniona. Po drugie, jeśli zechcemy kod o długich liniach wydrukować, efekt będzie godny pożałowania!

Tabela 1 przedstawia liczby znaków w wierszu, jakie zmieszczą się na wydruku. Ze wzrostem wielkości kroju czcionki oraz marginesów liczba ta oczywiście maleje. Stosując czcionkę wielkości 10 punktów i margines mniejszy niż 2,5 cm możemy na stronie zmieścić od 80 niemalże do 100 znaków.

Dodajmy na marginesie, że do wydruku kodu programów komputerowych stosujemy czcionkę *nieproporcjonalną*, czyli taką, w której każdy znak ma stałą szerokość, na przykład `Courier New`. Użycie czcionki proporcjonalnej `Arial`, `Verdana` czy `Times New Roman` (niestety taka sytuacja ma miejsce we wspomnianej książce „*Język ANSI C*”) powoduje, że wcięcia kodu oraz białe znaki są niezauważalne, zaś cały kod wygląda po prostu źle.

Dane zawarte w tabeli 1 zostały uzyskane przez przeanalizowanie wydruków dokumentów z tabeli 2.

2.3. Tabulatory kontra spacje

Do wcinania kodu należy stosować znaki spacji. Tabulatory są znakami, zależnymi od bieżącego kroju czcionki. Wcięcia wykonane tabulatorem mogą ulec zmianie w przypadku zmiany wielkości czcionki. Dowodzi tego ostatni z dokumentów zawartych w tabeli 2.

Standardy kodowania biblioteki PEAR zalecają stosowanie czterech spacji do wcinania kodu.

lp.	Nazwa pliku	Marginesy
1.	miarka-kodu-0-0-cm.zip	0,0 cm
2.	miarka-kodu-0-5-cm.zip	0,5 cm
3.	miarka-kodu-1-0-cm.zip	1,0 cm
4.	miarka-kodu-1-5-cm.zip	1,5 cm
5.	miarka-kodu-2-0-cm.zip	2,0 cm
6.	miarka-kodu-2-5-cm.zip	2,5 cm
7.	miarka-kodu-tabulacja.zip	Tabulatory

Tabela 2. Wydruki testowe kodu

2.4. Średnik i wiele instrukcji w jednym wierszu

Średnik kończący instrukcje formatujemy tak jak w zwykłym tekście, przyklejając do poprzedzającego go napisu:

```
echo 'ala';  
$i++;  
$a = $b - 1;
```

Podobnie postępujemy, gdy średnik pojawi się wewnątrz bardziej złożonej instrukcji, na przykład `for`:

```
for ($i = 0; $i < $ile; $i++) {  
    ;  
}
```

Nie zapisujemy kilku instrukcji w jednej linijce:

PRZYKŁAD NIEZALECANY

```
$a = 5; $b = 3;
```

Powyższe uwagi możemy również z powodzeniem zastosować do komentarzy. Nie należy umieszczać komentarzy w tej samej linii, co komentowany kod. Zamiast:

PRZYKŁAD NIEZALECANY

```
$a = $b + $c; // suma liczb
```

piszmy:

```
// suma liczb  
$a = $b + $c;
```

2.5. Puste linie w kodzie skryptu

W skrypcie dopuszczalne są puste linie zwiększające czytelność. Wewnątrz metod, funkcji oraz instrukcji sterujących stosujemy wedle uznania pojedyncze puste linie. Oddzielając większe partie kodu, na przykład klasy, czy zmienne globalne od stałych możemy stosować podwójne puste linie.

Nie stosujemy potrójnych pustych linii ani dłuższych przerw w kodzie skryptu.

3. Format kodu PHP

3.1. Otwarcie kodu PHP

Kod PHP otwieramy stosując wyłącznie znaczniki `<?php` oraz `?>`. Dzięki temu skrypty będą bardziej niezależne od konfiguracji PHP. Znaczniki `<?` otwiera kod PHP tylko wówczas, gdy opcja `short_open_tags` ma wartość `On`. Ponadto stosowanie skróconych znaczników powoduje konflikt z językiem XML.

Po znacznikach `<?php` oraz `?>` nie stosujemy wcięcia.

Rezygnacja z opcji `short_open_tag` wymusza także rezygnację ze skróconego zapisu instrukcji `echo`: `<?= $zm; ?>` .

3.2. Komentarze

PHP zezwala na stosowanie komentarzy znanych z języka C, z C++ oraz powłoki u*ixowej `bash`:

```
/* komentarz w~C */
// komentarz w~C++
# komentarz w~shellu
```

Standardy kodowania PEAR odradzają stosowanie komentarzy rozpoczynanych znakiem `#`.

Bez względu na wybrany rodzaj komentarzy, pamiętajmy o tym, by tekst komentarza był wcięty podobnie, jak komentowane instrukcje:

```
for ($i = 0; $i < $ile; $i++) {
    // Czy $i jest parzyste?
    if ($i % 2 == 0) {
        /*
         * $i jest parzyste
         */
        echo $i;
        ...
    }
}
```

3.3. Operatory

Operatory języka PHP możemy podzielić na trzy grupy: operatory dwuargumentowe, operatory jednoargumentowe oraz operatory specjalne.

3.3.1. Operatory dwuargumentowe

Operatorami dwuargumentowymi są między innymi: instrukcja przypisania, operatory arytmetyczne `+`, `-`, `*`, `/` oraz operatory logiczne `>`, `<`, `>=`, `<=`. Otaczamy je z każdej strony co najmniej jedną spacją:

```
$a = 3;  
$b = $a * 5;  
$c = $a + $b;  
$d = $c <= 666;
```

W przypadku serii podobnych instrukcji, stosujemy większą liczbę spacji by otrzymać kod wyrównany pionowo:

```
$plk           = file('a.txt');  
$liczba_wierszy = count($plk);  
$wynik         = array();
```

Pełne zestawienie operatorów, które otaczamy znakami spacji jest zawarte w tabeli 3. Zwróćmy uwagę, że język PHP pozwala na stosowanie operatorów logicznych znanych z języka Pascal: **and**, **or** oraz **xor**. Najlepiej na wstępie zdecydować się na stosowanie jednego rodzaju operatorów. Korzystamy z zestawu **&&**, **||** oraz **^** lub **and**, **or** i **xor**. W większości aplikacji i bibliotek pisanych w PHP przeważa użycie operatorów języka C.

3.3.2. Operatory jednoargumentowe

Drugą grupę, operatory jednoargumentowe, formatujemy „przyklejając” do operandu. Na przykład:

```
$i++;  
$bialy = !$czarny;  
$plk   = @file_get_contents('dane.txt');  
$ile   = (int)$wiek;
```

Pełna lista operatorów jednoargumentowych jest zawarta w tabeli 4.

3.3.3. Operatory specjalne

Trzecią grupę operatorów stanowią operatory specjalne. Należą do nich nawiasy kwadratowe **[i]** dotyczące tablic, operator do tworzenia obiektów **new**, oraz przecinek umożliwiający wykonanie wielu instrukcji. Warto w tym miejscu omówić także formatowanie odwołania do pól i metod obiektów i klas **->** oraz **::**, strzałkę **=>** występującą w definicji tablic oraz instrukcji **foreach** jak również referencje **&**. Formalnie, elementy te nie są operatorami, ale ich format jest zbliżony do formatu operatorów.

Po operatorze **new** umieszczamy jedną spację. Format ten różni się od formatu pozostałych operatorów jednoargumentowych (tabela 4). Jednakże ze spacji tej nie możemy zrezygnować, gdyż otrzymamy błąd:

PRZYKŁAD NIEPOPRAWNY

```
$obj = newTStudent();
```

Operator	Działanie	Przykład
=	Podstawienie	\$a = 5;
+	Suma	\$c = \$a + \$b;
-	Różnica	\$c = \$a - \$b;
*	Iloczyn	\$c = \$a * \$b;
/	Iloraz	\$c = \$a / \$b;
%	Reszta z dzielenia	\$a = \$b % \$c;
.	Konkatenacja napisów	\$n = 'a' . 'b';
>>	Przesunięcie bitowe	\$a = 32 >> 2;
<<	Przesunięcie bitowe	\$a = 1 << 4;
<	Porównanie	\$a < \$b
>	Porównanie	\$a > \$b
<=	Porównanie	\$a <= \$b
>=	Porównanie	\$a >= \$b
==	Równość	\$a == \$b
!=	Różność	\$a != \$b
===	Identyczność	\$a === \$b
!==	Nie identyczność	\$a !== \$b
&&	Koniunkcja logiczna	\$a && \$b
	Alternatywa logiczna	\$a \$b
&	Koniunkcja bitowa	\$a & \$b
	Alternatywa bitowa	\$a \$b
^	Różnica symetryczna	\$a ^ \$b
.=	Konkatenacja napisów	\$a .= 'uff';
+=	Zwiększenie wartości zmiennej	\$a += 5;
-=	Zmniejszenie wartości zmiennej	\$a -= 10;
*=	Pomnożenie wartości zmiennej	\$a *= 2;
/=	Podzielenie wartości zmiennej	\$a /= 100;
%=	Reszta z dzielenia	\$a %= 15;
&=	Koniunkcja bitowa	\$a &= 7;
=	Alternatywa bitowa	\$a = 127;
^=	Różnica symetryczna	\$a ^= 1;
<<=	Przesunięcie bitowe	\$a <<= 2;
>>=	Przesunięcie bitowe	\$a >>= 2;

Tabela 3. Formatowanie operatorów dwuargumentowych

Operator	Działanie	Przykład
!	Negacja logiczna	<code>\$a = !\$b;</code>
~	Negacja bitowa	<code>\$a = ~\$b;</code>
++	Inkrementacja	<code>\$a++;</code> <code>++\$a;</code>
--	Dekrementacja	<code>\$a--;</code> <code>--\$a;</code>
(int)	Rzutowanie na typ int	<code>\$a = (int)\$b;</code>
(float)	Rzutowanie na typ float	<code>\$a = (float)\$b;</code>
(string)	Rzutowanie na typ string	<code>\$a = (string)\$b;</code>
(array)	Rzutowanie na typ array	<code>\$a = (array)\$b;</code>
(object)	Rzutowanie na typ object	<code>\$a = (object)\$b;</code>
@	Komunikaty diagnostyczne	<code>\$a = @file('a.txt');</code>

Tabela 4. Formatowanie operatorów jednoargumentowych

Nawiasy kwadratowe, stosowane przy tablicach formatujemy niemalże tak, jak sformatowalibyśmy nawiasy w zwykłym tekście: nawiasy są przyklejone do objętego tekstu:

```
$a[2] = $a[$i * 2 - 1];
```

Jedyną różnicą jest to, że po nazwie tablicy nie umieszczamy spacji.

Podobnie postępujemy z operatorem `,` (przecinek). Jego format jest identyczny jak format przecinków w zwykłym tekście. Przecinek „przyklejamy” do poprzedzającego go tekstu, zaś po przecinku umieszczamy pojedynczą spację:

```
for ($i = 0, $ile = count($tab); $i < $ile; $i++) {
    ...
}
```

Zwróćmy uwagę, że powyższy zapis będzie prowadził do długich wierszy. Zawsze możemy go zastąpić oddzielnymi instrukcjami:

```
$ile = count($tab)
for ($i = 0; $i < $ile; $i++) {
    ...
}
```

Operatory dostępu do pól i metod obiektów oraz statyczne wywołanie metod formatujemy nieco ciśniej od operatorów dwuargumentowych: nie stosujemy spacji dookoła znaków `->` oraz `::`. Oto przykładowe użycie:

```
$obj->getName(10);
PEAR::triggerError('Fatal error');
```


Operator	Działanie	Format
<code>new</code>	Tworzenie instancji klasy	Inaczej niż pozostałe operatory jednoargumentowe: jest spacja po operaterze!
<code>[]</code>	Odwołania do elementów tablic	Tradycyjne formatowanie nawiasów w tekście, ale bez spacji po nazwie tablicy.
<code>,</code>	Wykonanie kilku instrukcji	Tradycyjne formatowanie przecinków w tekście
<code>-></code>	Dostęp do składowej obiektu	Inaczej niż operatory dwuargumentowe: bez spacji.
<code>::</code>	Statyczne wywołanie metody	Inaczej niż operatory dwuargumentowe: bez spacji.
<code>=></code>	Definicja tablicy	Tak jak operatory dwuargumentowe (spacje otaczające).
<code>=></code>	Pętla <code>foreach</code>	Tak jak operatory dwuargumentowe (spacje otaczające).
<code>&</code>	Referencja	Tak jak operatory jednoargumentowe (bez spacji przed operandem).

Tabela 5. Formatowanie pozostałych operatorów i konstrukcji językowych zbliżonych do operatorów

Strzałka występująca w instrukcji `foreach` oraz w definicji tablic podlega tym samym zasadom, co operatory dwuargumentowe:

```
$a = array('one' => 'adin');
foreach ($t as $k => $v) {
```

Zaś operator `&` do tworzenia referencji formatujemy tak, jak operatory jednoargumentowe:

```
$a = &$b;
$a = &new MyClass();

function &getMe($a, &$b)
{
}
```

Pełne zestawienie operatorów specjalnych jest zawarte w tabeli 5.

3.3.4. Nawiasy grupujące i operator `?:`

Omawiając operatory należy wspomnieć o nawiasach występujących w wyrażeniach. Nawiasy formatujemy przyklejając je do objętych nimi zapisów:

(wyrażenie)

na przykład:

```
$a = (3 + 4) * 5;
```

lub

```
$a = ((3 + 2) / (5 - 1)) * (((3 + 4) * 5) + 11);
```

Ostatnim, niewymienionym dotychczas operatorem, jest operator selekcji `?:`. Pełni on rolę tę samą, co instrukcja `if`:

```
$c = $a > $b ? $a : $b;
```

Znaki `?` oraz `:` otaczamy pojedynczymi spacjami, zaś w przypadku otrzymania dłuższych linii lepiej użyć instrukcji `if`.

3.4. Instrukcje sterujące

Instrukcje sterujące języka PHP mogą być zapisywane na dwa sposoby. Pierwszy sposób wywodzi się z języka C i charakteryzuje się użyciem nawiasów klamrowych, na przykład:

```
if ($a >= 0) {  
    echo 'Liczba nieujemna';  
} else {  
    echo 'Liczba ujemna';  
}
```

Drugi sposób — nazywany notacją alternatywną — różni się od pierwszego tym, że w miejscu otwierającej klamry pojawia się dwukropek, zaś zamykająca klamra jest zastąpiona jednym z napisów: `endif`, `endwhile`, `endfor`, `endforeach` lub `endswitch`. Na przykład:

PRZYKŁAD NIEZALECANY

```
$i = 1;  
while ($i <= 10):  
    echo $i;  
    $i++;  
endwhile;
```

Na wstępie rezygnujemy z alternatywnego zapisu instrukcji sterujących i stosujemy wyłącznie notację języka C.

Drugą zasadą jest stosowanie nawiasów klamrowych nawet wówczas, gdy są one zbędne. Zamiast:

PRZYKŁAD NIEZALECANY

```
if ($a < 0)  
    $a = -$a;
```

piszmy:

```
if () {  
    ;  
}  
  
if () {  
    ;  
} else {  
    ;  
}
```

Listing 1. Formatowanie instrukcji `if`

```
if ($a < 0) {  
    $a = -$a;  
}
```

Do formatowania instrukcji sterujących stosujemy zasady zawarte w książce Kernighana i Ritchiego (w żargonie format ten bywa nazywany Kernighan-Ritche). Klamra otwierająca instrukcji sterującej znajduje się w tej samej linii, co początek instrukcji, zaś zamykająca — w tej samej kolumnie co pierwszy znak instrukcji:

```
if () {  
    ;  
}  
  
while () {  
    ;  
}  
  
foreach () {  
    ;  
}
```

Dodatkowo, nawiasy obejmujące warunek instrukcji, są otoczone z każdej strony przez pojedynczą spację zaś po klamrze kończącej całą instrukcję nie umieszczamy średnika.

Format instrukcji `if` jest przedstawiony na listingu 1. Wszystkie listingi dotyczące instrukcji sterujących zawierają znak średnik wskazujący wcięcie zagnieżdżonych instrukcji.

W przypadku skomplikowanych, wieloczęściowych instrukcji `if` należy wybrać jeden z dwóch dostępnych wariantów. Pierwszy stosuje dwa słowa kluczowe `if` oraz `else`, zaś drugi — pojedyncze słowo kluczowe `ifelse`. Szczegóły przedstawia listing 2.

Wszystkie pętle dostępne w języku PHP są przedstawione na listingu 3.

```
//wariant stosujący jedno słowo kluczowe
if () {
    ;
} elseif () {
    ;
} elseif () {
    ;
} else {
    ;
}

//wariant stosujący dwa słowa kluczowe
if () {
    ;
} else if () {
    ;
} else if () {
    ;
} else {
    ;
}
```

Listing 2. Dwa warianty złożonej instrukcji if

```
while () {  
    ;  
}  
  
do {  
    ;  
} while ();  
  
for (; ; ) {  
    ;  
}  
  
for ($i = 0; $i < $ile; $i++) {  
    ;  
}  
  
foreach ($t as $v) {  
    ;  
}  
  
foreach ($t as $k => $v) {  
    ;  
}
```

Listing 3. Format pętli while, do-while, for oraz foreach

```
switch () {  
  case 1:  
    ;  
    break;  
  
  case 2:  
    ;  
    break;  
  
  default:  
    ;  
    break;  
}
```

Listing 4. Format instrukcji `switch`

Natomiast listing 4 przedstawia dość kontrowersyjną instrukcję `switch`. Pomimo zaleceń zawartych w standardach kodowania `PEAR`, wszystkie pakiety `PEAR` stosują wcięcie w odniesieniu do słów `case`:

PRZYKŁAD NIEZALECANY

```
switch () {  
  case 1:  
    break;  
  
  case 2:  
    break;  
  
  default:  
    break;  
}
```

Dodajmy jeszcze, że format instrukcji `declare` jest identyczny jak format instrukcji `while`:

```
declare () {  
  ;  
}
```

oraz, że średnik następujący po instrukcjach `break` oraz `continue` jest do nich przyklejony:

```
break;  
continue;
```

Ostatnia uwaga dotyczy instrukcji `return`. Nie stosujemy nawiasów otaczających zwracaną wartość. Zamiast:

PRZYKŁAD NIEZALECANY

```
return(true);
```

piszmy:

```
return true;
```

3.5. Dołączanie kodu

Kod dołączamy do skryptu stosując instrukcje `include`, `include_once` oraz `require_once`. Pamiętajmy, że są to instrukcje a nie funkcje. Zatem nie stosujemy nawiasów otaczających nazwę dołączanego pliku:

```
require_once 'user.class.php';  
include_once 'a.inc.php';
```

W przypadku wielokrotnego dołączania jednego pliku do skryptu stosujemy instrukcję `include`:

```
include 'tabela.inc.php';
```

3.6. Wywołanie funkcji

Nie umieszczamy odstępów pomiędzy nazwą funkcji, nawiasem otwierającym i pierwszym parametrem. Wewnątrz listy parametrów stosujemy reguły takie, jak w tekście. Po znaku przecinka umieszczamy spację, zaś sam przecinek „przyklejamy” do poprzedniego argumentu:

```
$zm = myFun($a, $b, $c);
```

3.7. Definicja funkcji

Parametry występujące w definicji funkcji formatujemy identycznie jak w wywołaniu. Klamry otaczające ciało funkcji umieszczamy w oddzielnych liniach w pierwszej kolumnie kodu:

```
function myFun($a, $b, $c)  
{  
    return $a + $b + $c;  
}
```

3.8. Definicja klasy

Klasy formatujemy stosując konwencję zbliżoną do definicji funkcji:

```
class A
{
    ...
}
```

Nawiasy klamrowe, otaczające treść klasy umieszczamy w osobnych liniach w tej samej kolumnie, co litera c słowa kluczowego `class`.

4. Nazewnictwo

4.1. Stałe

Nazwy stałych piszemy dużymi literami. Stosujemy znak podkreślenia do oddzielania wyrazów. Do nazwy stałej należy dołączać przedrostek określający pakiet, w którym stała została zdefiniowana, na przykład:

```
CHESS_CLIENT_PORT
TEXT_WORD_SEPARATORS
```

W podanych przykładach wyrazy `CHESS` oraz `TEXT` stanowią nazwy pakietów.

Jedynym wyjątkiem są stałe `true`, `false` oraz `null`, których nazwy piszemy małymi literami.

4.2. Zmienne globalne

Nazwy zmiennych globalnych rozpoczynamy prefiksem `_NAZWAPAKIETU_`, na przykład:

```
$_CHESS_username
$_TEXT_active
```

Pozwoli to na uniknięcie zdublowania nazwy zmiennej w poszczególnych pakietach.

4.3. Klasy

Należy nadawać jasne nazwy klasom. Unikamy przy tym skrótów. Nazwa klasy powinna rozpoczynać się dużą literą. W celu zawarcia informacji o hierarchii klas stosujemy znak podkreślenia:

```
Log
Net_Finger
HTML_Upload_Error
```

Składowe prywatne poprzedzamy prefiksem `_`:

```
var $_nazwa_pliku;
```


4.4. Funkcje i metody

Funkcje i metody nazywamy stosując zasadę „*wielbłądzich garbów*” (*ang. camel caps*). Nazwę funkcji rozpoczynamy zawsze małą literą, zaś każdy kolejny człon (tj. wyraz, skrót lub fragment wyrazu) nazwy rozpoczynamy z dużej litery. Ewentualnie, nazwę funkcji poprzedzamy prefiksem pakietu:

```
connect()
getData()
buildSomeWildget()
XML_RPC_serializeData()
```

4.5. Pliki

Pliki zawierające klasy nazywamy nazwą klasy z rozszerzeniem `.class.php`, na przykład:

```
Smarty.class.php
```

Ewentualnie pomijamy człon `.class` w nazwie pliku:

```
DB.php
```

Każdą klasę zapisujemy w osobnym pliku lub grupujemy w jednym pliku kilka klas współpracujących ze sobą.

Plikom zawierającym stałe, zmienne globalne oraz funkcje nadajemy rozszerzenie `.inc.php` ewentualnie pomijając człon `.inc`:

```
walidacja.inc.php
function.eval.php
```

4.6. Przykładowe adresy URL

Zgodnie z dokumentem RFC 2602, poprawnymi przykładowymi adresami URL są adresy zawierające jedną z następujących nazw domenowych:

```
example.com
example.org
example.net
```

na przykład:

```
http://www.example.org/skrypt.php?imie=Jan&nazwisko=Nowak
email:ktos@example.net
```

5. Spójność formatu

Wymagania wymienione powyżej są w znacznej mierze oparte na wielokrotnie wymienionym dokumencie zatytułowanym „*Standardy kodowania biblioteki PEAR*”. Zagadnienia nieporuszone przez standardy (m.in. kilka operatorów, puste wiersze, wcinanie komentarzy) oparłem na własnych przyzwyczajeniach oraz na kodach aplikacji wymienionych we wstępie.

Zwróćmy uwagę na pewien ważny aspekt formatowania kodu, a mianowicie jego spójność. Może się zdarzyć tak, że części zasad nie akceptujemy i stosujemy własne rozwiązania. Na przykład formatowanie referencji wolelibyśmy zapisywać w postaci:

```
$a =& $b;
```

W takiej sytuacji należy konsekwentnie stosować jeden z zapisów. Bałaganiarstwem jest pisanie:

```
$a =& $b;
```

```
...
```

```
$a = &$b;
```

A bałaganiarstwo nigdy nie popłaca.

Szczególną uwagę należy zwrócić na następujące zagadnienia:

- otwarcie kodu PHP,
- wcięcia,
- puste linie,
- komentarze,
- instrukcje wyjścia `echo` oraz `print`,
- stosowanie napisów,
- odwołania do liter napisów,
- operator `?:`,
- operatory Pascala-podobne `or`, `and` i `xor`,
- złożoną instrukcję `if`,
- długie wyrażenia,
- długie nagłówki funkcji,
- oraz długie wywołania funkcji.

Jeśli z jakichkolwiek powodów zdecydujemy się na krótkie otwarcie PHP oraz wcięcia przy użyciu tabulatorów, wówczas w całym skrypcie (czy też we wszystkich skryptach w przypadku większych projektów) należy konsekwentnie stosować wybrane rozwiązanie.

Szczególnym przypadkiem są instrukcje `echo` oraz `print`. Składnia języka PHP zezwala na następujące siedem rozwiązań:

```
echo 'a';
echo'a';
echo('a');
print 'a';
print'a';
print('a');
echo 'a', 'b', 'c';
```

Moim zdaniem należy się zdecydować i stosować wyłącznie jedno z nich. Ja osobiście jestem przyzwyczajony do instrukcji:

```
echo 'a';
```

Podobnie rzecz się ma w odniesieniu do napisów. Przykładowe rozwiązania w tym względzie mogą wyglądać:

```
echo ' tekst ';
echo " tekst ";

require_once 'a.inc.php';
require_once "a.inc.php";

echo ' tekst ' . $a . ' TEKST ' . $b;
echo " tekst $a TEKST $b";
```

Najczęstszym rozwiązaniem w przeanalizowanych przeze mnie kodach jest stosowanie apostrofów zawsze tam, gdzie cudzysłów nie jest konieczny, na przykład:

```
echo ' tekst ';
require_once 'a.inc.php';
```

Ponadto stosując rozwiązanie zawierające konkatencję napisów:

```
echo ' tekst ' . $a . ' TEKST ' . $b;
```

mamy możliwość skrócenia długości wierszy.

Patrząc na formatowanie kodu nieco szerzej, bezpiecznym wyjściem jest stosowanie zasady zawartej w specyfikacji języka HTML. Zasada ta odnosi się do programów generujących i analizujących kod HTML i brzmi: *„Generujemy kod jak najbardziej rygorystycznie zgodny z pewnymi (np. powyższymi) wytycznymi, zaś analizując cudze programy stosujemy jak najbardziej liberalne podejście.”*

```
var $security_settings = array(
    'PHP_HANDLING'      => false,
    'IF_FUNCS'          => array(
        'array',
        'list',
        'isset',
        'empty',
        'count',
        'sizeof',
        'in_array',
        'is_array',
        'true',
        'false',
        'null'
    ),
    'INCLUDE_ANY'        => false,
    'PHP_TAGS'           => false,
    'MODIFIER_FUNCS'     => array('count'),
    'ALLOW_CONSTANTS'   => false
);
```

Listing 5. Pole `$security_settings` po przeformatowaniu

6. Problemy z życia wzięte

Głównym problemem, jaki napotkamy przy formatowaniu rzeczywistych będzie długość linii. W wielu sytuacjach możliwe jest przeformatowanie kodu tak by wiersze jego miały długość nieprzekraczającą 80 i kod był ciągle czytelny. Jednakże są i takie przypadki, w których będziemy bezsilni.

Analizując plik `Smarty.class.php` w wersji 1.516 (dystrybucja 2.6.10 pakietu `Smarty`) w linii 229 znajdziemy definicję pola `$security_settings`. Kod ten możemy znacznie lepiej sformatować stosując do długich wywołań funkcji konwencję zbliżoną do instrukcji sterujących:

```
nazwaFunkcji(
    ...
    ...
    ...
);
```

Listing 5 przedstawia skrócony zapis definicji pola `$security_settings`.

Podobnie postępujemy z nagłówkiem funkcji `register_object()` z linii 716. Nagłówek ten po zastosowaniu powyższego rozwiązania przyjmie postać:

```
function register_object(  
    $object,  
    &$object_impl,  
    $allowed = array(),  
    $smarty_args = true,  
    $block_methods = array()  
) {  
    ...  
}
```

Natomiast instrukcja `return` z linii 950 `call_user_func_array()` może zostać zapisana jako:

```
return call_user_func_array(  
    $this->cache_handler_func,  
    array(  
        'clear',  
        &$this,  
        &$dummy,  
        $tpl_file,  
        $cache_id,  
        $compile_id,  
        $exp_time  
    )  
);
```

Analogicznie, wywołanie `smarty_core_write_compiled_include()` z linii 1421 (oryginalna linia ma niemalże 200 znaków długości) formatujemy:

```
smarty_core_write_compiled_include(  
    array_merge(  
        $this->_cache_include_info,  
        array(  
            'compiled_content' => $_compiled_content,  
            'resource_name'    => $resource_name  
        )  
    ),  
    $this  
);
```

Długie warunki instrukcji `if` czy `while` możemy zapisywać stosując konwencję:

```
if (  
    ...  
) {  
    ...  
} else {  
    ...  
}
```

Linia 1694 po zastosowaniu powyższego rozwiązania przyjęłaby postać:

```
if (  
    ($string{0} == "" || $string{0} == '')  
    && $string{strlen($string)-1} == $string{0}  
) {  
    return substr($string, 1, -1);  
} else {  
    return $string;  
}
```

7. Podsumowanie

Trud włożony w przyzwyczajenie się do zasad formatowania kodu PHP opisanych w dokumentacji biblioteki **PEAR** z pewnością się opłaci. Na format kodu warto zwrócić uwagę od początku nauki języka, wtedy unikniemy konieczności przestawiania się z jednego formatu na inny.

Jeśli którąkolwiek z zasad uważamy za zbędną, lub chcemy stosować własne wypracowane rozwiązanie, pamiętajmy o zachowaniu spójności i konsekwencji.

Nazewnictwo w programach komputerowych stanowiło temat rozprawy doktorskiej Charles Simonyi p.t. „*Meta-Programming: A Software Production Method*”. Praca ta ustanowiła szeroko rozpowszechniony standard nazewnictwa nazywany notacją węgierską. Znajdziemy ją na stronach MSDN (dokładny adres jest podany w tabeli 6).

Szerszy opis adresów URL, w tym wspomnianych `example.org`, zawarto w dokumencie RFC 2606 p.t. „*Reserved Top Level DNS Names*”.

Badając formatowanie kodu w językach, opartych o składnię języka C, warto odwiedzić witrynę programu `astyle`. Program `astyle` służy do automatycznego przeformatowania kodu programu w języku C. z dokumentacji dowiemy się o różnych stylach formatowania programów w języku C.

Zawartość	Adres
Standardy kodowania	http://pear.php.net/manual/en/standards.php
Notacja węgierska	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvsngen/html/HungaNotat.asp
RFC 2606	http://www.ietf.org/rfc/rfc2606.txt
astyle	http://sourceforge.net/projects/astyle/

Tabela 6. Adresy w internecie