Damian Daszkiewicz

PHP dla zielonych

Podstawy programowania w języku PHP

Damian Daszkiewicz

PHP dla zielonych

Wydawnictwo Escape Magazine
http://www.escapemag.pl

PHP dla zielonych

Damian Daszkiewicz

Skład i łamanie:

Patrycja Kierzkowska

Korekta:

Anna Matusewicz

Wydanie pierwsze, Jędrzejów 2007

ISBN: 978-83-60320-82-2

darmowy fragment

Wydawnictwo Publikacji Elektronicznych Escape Magazine

ul. Spokojna 14

28-300 Jędrzejów

http://www.escapemag.pl

Wstęp

Cieszę się, że zakupiłeś tę publikację. Starałem się pisać jak najprostszym językiem i umieszczać dużo zrzutów ekranowych, bo uważam, że jeden obraz potrafi więcej wyjaśnić niż nudny i długi opis. Podczas pisania tej publikacji założyłem, że masz jakieś niewielkie doświadczenie związane z programowaniem. Jeśli nigdy wcześniej nie programowałeś, to się nie martw, bo omawiane przykłady są dość proste i dość dobrze opisane.

Wszystkie przykłady możesz ściągnąć ze strony: http://php.escapemag.pl/przyklady.zip

Nie będziesz musiał tracić czasu na ich wklepywanie ;-)

W razie drobnych problemów możesz do mnie napisać email: d.daszkiewicz@escpoland.pl Jako drobny problem mam na myśli wytłumaczenie jakiegoś fragmentu ebooka, którego do końca nie zrozumiałeś. E-mail nie służy do proszenia o przerabianie skomplikowanych skryptów.

Damian Daszkiewicz

Elementarne podstawy języka znaczników HTML

HTML, to język skryptowy osadzany w dokumentach HTML, więc pisząc skrypty PHP, należy znać choć troszkę HTML. W ebooku poświęcę odrobinę miejsca na wyjaśnienie elementarnych podstaw HTML-a. Jeśli nie znasz tego języka, a po przeczytaniu tego krótkiego wprowadzenia poczujesz pewien niedosyt, to mogę polecić bardzo dobry i darmowy kurs autorstwa Pawła Wimmera: http://webmaster.helion.pl/kurshtml/

Czym jest HTML?

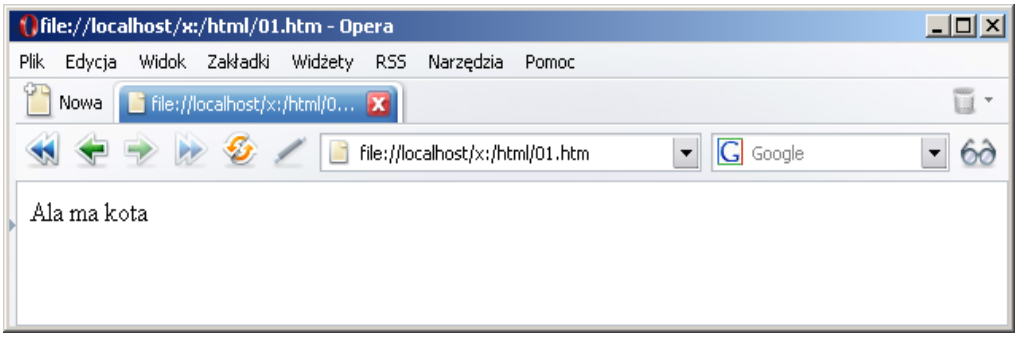
HTML (ang. HyperText Markup Language), to język składający się ze znaczników (ang. tags) stosowany do pisania stron www.

Podstawy języka HTML

Do pisania stron w czystym języku HTML wystarczy zwykły windowsowy Notatnik, bo dokumenty HTML są zapisywane w formacie tekstowym. Oczywiście Notatnik nie jest najwygodniejszym programem. Istnieją specjalne edytory tekstu, dostosowane do tworzenia stron, zawierające różne przydatne narzędzia np. znany Pajączek, ale te programy są przeznaczone dla osób zaawansowanych, natomiast osoby początkujące, mogłyby się przerazić ilością posiadanych funkcji.

Aby stworzyć najprostszą stronę można napisać w notatniku jakiś tekst np. `Ala ma kota`

Teraz należy zapisać tak stworzony dokument, nadając mu jakąś nazwę i rozszerzenie **htm** lub **html**. Po zapisaniu otwieramy plik w swojej ulubionej przeglądarce internetowej i zobaczysz mniej więcej coś takiego:



Można powiedzieć, że to jest pierwsza najprostsza strona internetowa. Jeśli próbowałeś samemu napisać trochę więcej tekstu, to pewnie zauważyłeś, że taki monotonny tekst, bez pogrubień, różnych rodzajów czcionek jest dość niewygodny do czytania. Druga wada to kłopoty z polskimi literami (zamiast nich mogą się pojawić krzaczki). O polskich literach napiszę nieco później.

Często dane partie tekstu należy np. pogrubić. Dokonujemy tego, umieszczając TAG (zwany też: znacznik) przed tekstem, który chcemy pogrubić i TAG w miejscu, gdzie ma się kończyć ten pogrubiony tekst.

Gdy przeglądarka natrafi na TAG , wie, że ma włączyć pogrubianie do czasu, gdy nie natrafi na TAG . Gdy chcemy użyć *kursywy*, to stosujemy oraz , a do podkreślenia <u> oraz </u>. Znaczniki można ze sobą mieszać tj. dany tekst może być zarówno pogrubiony jak i podkreślony. Oto prosty przykład:

```
Ala ma <b><u>kota</u></b>
```

Zauważ, że znaczniki zamykamy w odwrotnej kolejności, czyli najpierw otworzyliśmy później <u>, a następnie zamknęliśmy TAG </u>, który był otworzony jako ostatni i na końcu zamknęliśmy .

Jeśli tego nie rozumiesz, to mam dobrą wiadomość: dyskutując na na forach internetowych, na pewno zauważyłeś, że tam jest podobnie – aby pogrubić tekst używamy (na forach nazwano to BBCODE): [b] oraz [/b] (kursywa: [i] oraz [/i], a podkreślenie: [u] oraz [/u]).

Teraz mała uwaga: jeśli stworzysz prostą stronę i wrzucisz ją na serwer, to musisz zwrócić uwagę na wielkość liter. Pod systemem Windows nie ma różnicy czy plik się nazywa **INDEX.HTML,** czy może **index.html**. Serwery, na które się wrzucasz strony internetowe, zazwyczaj pracują pod kontrolą jakiejś dystrybucji Linuksa i dla nich to dwa różne pliki! Tam może w jednym katalogu istnieć zarówno plik **INDEX.HTML** jak i **index.html**. Aby uniknąć różnych niespodzianek (np. odwołujesz się do pliku nieistniejącego pliku **INDEX.HTML,** a masz na serwerze plik o nazwie **index.html**), proponuję, abyś wszystkim plikom nadawał nazwy, korzystając tylko z małych liter. Może nie jest to wygodne, ale w przyszłości oszczędzi kłopotów.

Budowa pliku HTML

Poprzedni przykład był bardzo prosty. Zawierał jedynie tekst, który się pojawił na stronie. Oprócz treści, dokument HTML powinien też zawierać pewne informacje konfiguracyjne. Prosty przykład: jeśli pisząc stronę, użyjemy polskich znaków (tj. ą, ć, ę, ó, ł, ń, ś, ź, ż), to zamiast nich, mogą pojawić się jakieś krzaczki. Dlaczego? Otóż te znaki są niestandardowe. Wiele języków posiada specyficzne dla siebie znaki (np. niemiecki zawiera „umlałty", rosyjski zawiera zupełnie inny alfabet (cyrylica), język czeski z kolei zawiera znaki z kreskami, daszkami i kółkami np. č, á, ů). Jeśli chcemy korzystać z tego typu niestandardowych znaków (narodowych), to należy poinformować przeglądarkę internetową „z jakiego alfabetu" chcemy korzystać. Dlatego też prawidłowy dokument HTML powinien zawierać dodatkowe informacje między `<head>` a `</head>`. Poniżej przedstawiam poprawie stworzony dokument:

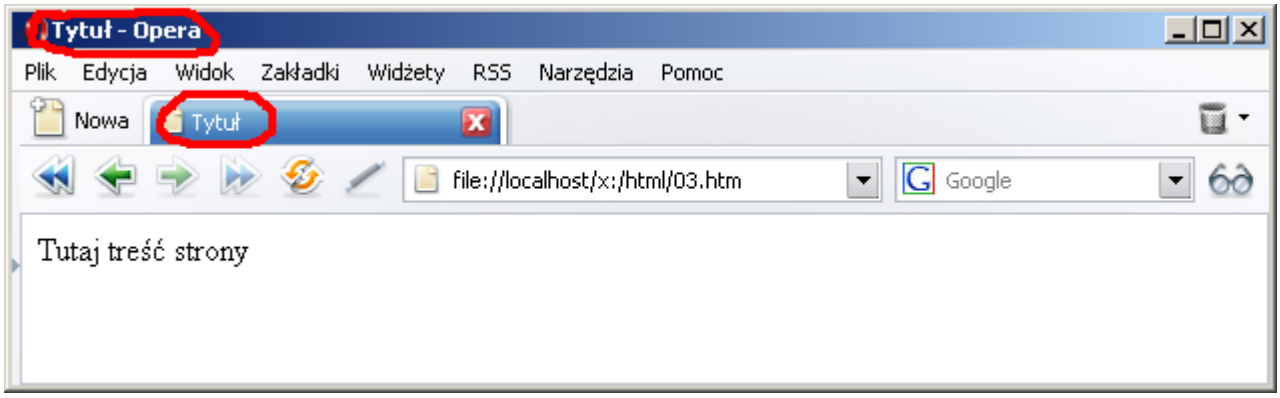
```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
<title>Tytuł</title>
</head>
<body>
Tutaj treść strony
</body>
</html>
```

Znaczniki `<html>` i `</html>` informują, że przeglądarka ma do czynienia z dokumentem HTML (a nie np. plikiem tekstowym, czy graficznym). Pomiędzy `<head>` a `</head>` wpisuje się pewne dane konfiguracyjne - w tym przypadku podaliśmy tytuł strony i informacje o tym „z jakiego alfabetu" chcemy korzystać. Pomiędzy `<body>` a `</body>` znajduje się treść właściwa (czyli tekst, który zostanie wyświetlony przez przeglądarkę internetową). Każdy dokument powinien mieć taką budowę, bo wtedy przeglądarka wie, jak ma zinterpretować dokument. Dzięki temu zamiast krzaczków pojawią się polskie litery.

Co do polskich liter - użyłem tutaj kodowania **windows-1250**. Nie jest to zalecany standard, ale każda współczesna przeglądarka go poprawnie obsługuje. Korzystanie z innych formatów wymaga albo korzystania ze specjalnych edytorów HTML albo konwertowania stworzonych dokumentów co mogłoby być dość uciążliwe dla osób, które poznają dopiero HTML. Ponieważ ten ebook jest o PHP, a nie o HTML, więc nie będę dokładnie poruszał tego problemu. Powiem tylko tyle, że według polskiej normy powinno się korzystać z kodowania **iso-8859-2,** jednak i to jest powoli wypierane przez standard **UTF-8,** który pozwala na używanie w jednym dokumencie wielu alfabetów. Dzięki temu bez problemu można zaprojektować stronę zawierającą litery charakterystyczne np. dla języka polskiego i rosyjskiego co jest przydatne, gdy chcesz stworzyć stronę zawierającą słowniczek polsko-rosyjski).

Jeśli poznasz już troszkę dokładniej HTML, to zalecam zapoznać się z informacjami na temat kodowania polskich znaków: http://www.ogonki.agh.edu.pl/ Zorganizuj sobie wygodny edytor tekstów potrafiący w locie konwertować znaki do zalecanego formatu – ja używam program EditPadPro.

Jest jeszcze jedna rzecz do wyjaśnienia: tytuł strony (który jest zawarty pomiędzy **`<title>`** a **`</title>`**. Otóż ten tekst zostanie wyświetlony na pasku tytułowym przeglądarki internetowej (zobacz rysunek na następnej stronie)



Zwróć uwagę na fragmenty zaznaczone na czerwono. Jak porównasz ten rysunek z poprzednim, to zauważysz, że w tamtym dokumencie (w którym nie zdefiniowaliśmy tytułu strony) w tych miejscach została wyświetlona nazwa pliku.

Parę słów o akapitach

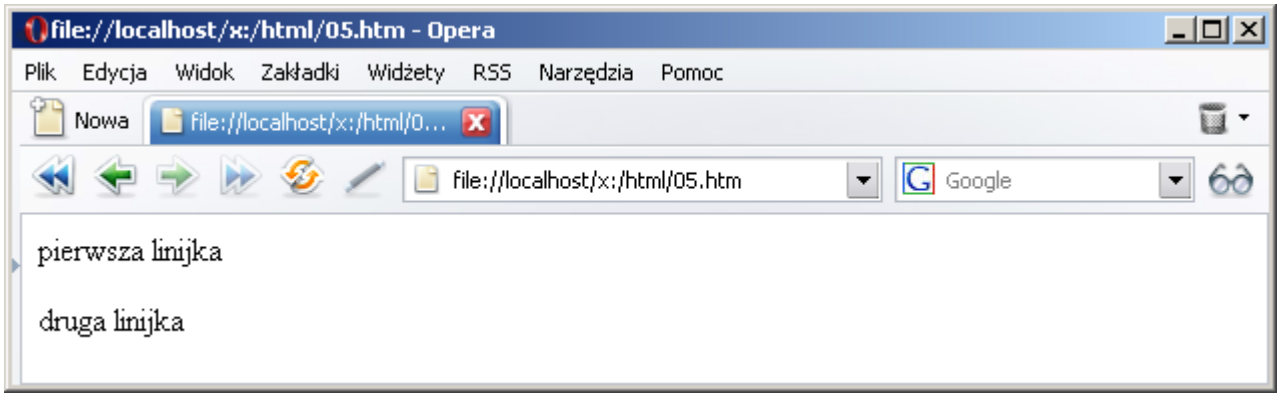
Jeśli stworzysz taki dokument HTML (aby nie zaciemniać przykładu pominąłem <head> </head>)

```
pierwsza linijka

druga linijka
```

To przeglądarka internetowa zignoruje znak enter i wyświetli tekst: pierwsza linijka druga linijka (enter został zamieniony na spację). Nie jest do najlepsze wyjście, bo tekst pisany jest ciągiem i trudno to czytać. Jest na to rada: należy tworzyć akapity. Akapit się otwiera znacznikiem , a zamyka . Oto zmodyfikowany przykład:

```
<p>pierwsza linijka</p>

<p>druga linijka</p>
```

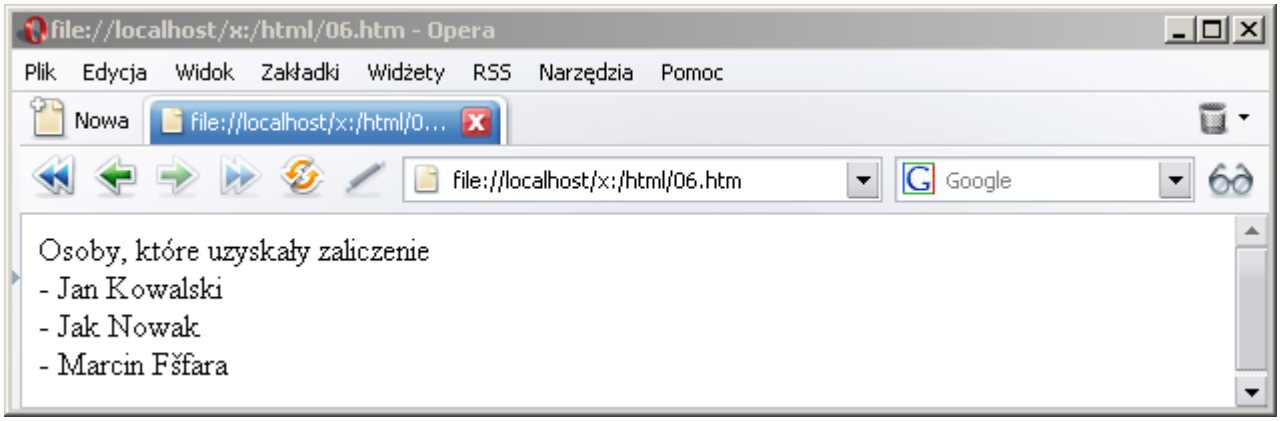
Teraz tak to będzie wyglądało w przeglądarce:



Jak widzisz, teraz jest dużo lepiej. Czasami zachodzi potrzeba, aby „enter” nie był podwójny. Można posłużyć się znacznikiem
. Oto kolejny przykład:

```
<p>Osoby, które uzyskały zaliczenie<br>

   - Jan Kowalski<br>

   - Jak Nowak<br>

   - Marcin Fąfara</p>
```

Oto efekt:



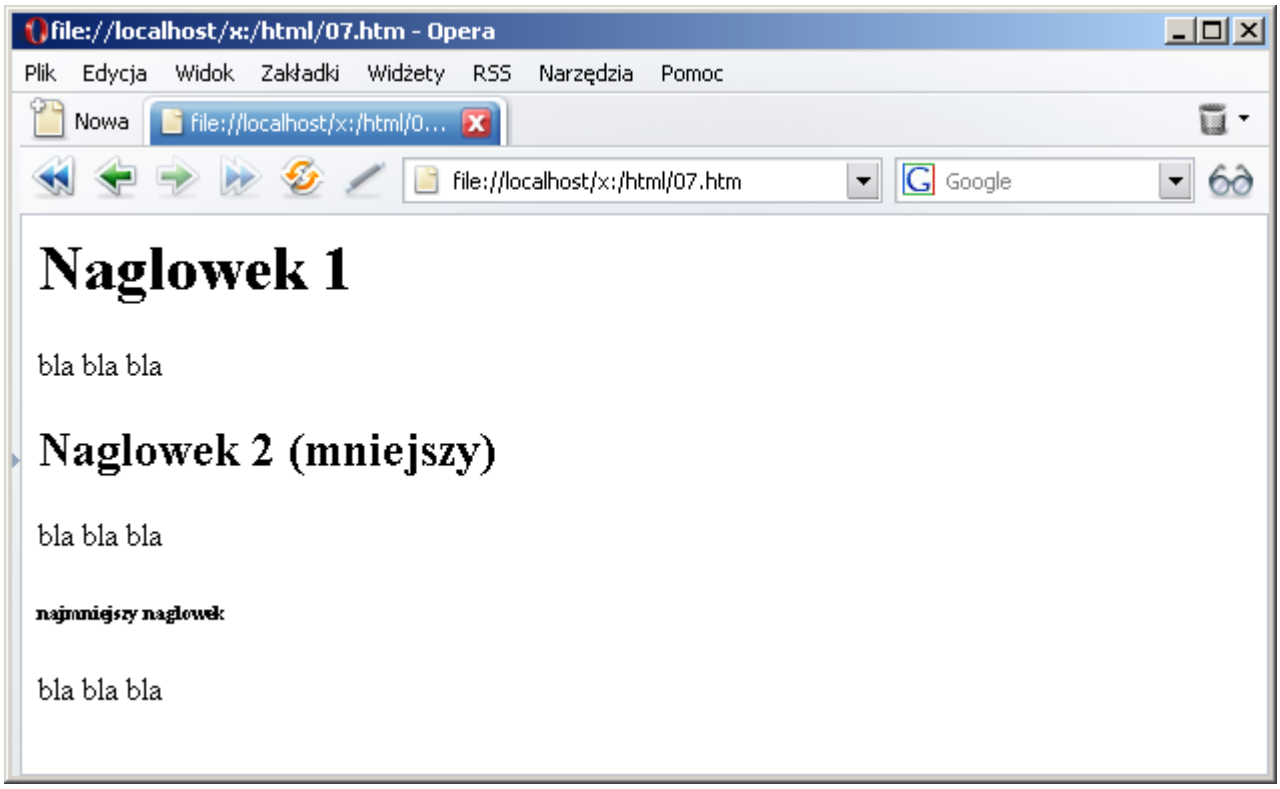
Jak widzisz, teraz entery nie są podwójne. Tag `<br>` jest przydatny, gdy chcemy wymienić jakieś np. nazwiska. Jednak pisząc normalny tekst, stosuj normalne akapity, korzystając z `<p>` i `</p>` (zwiększy to czytelność strony). Jak widzisz, w tym dokumencie, aby nie zaciemniać przykładu, nie użyłem odpowiednich nagłówków. Nazwisko Fąfara zostało wyświetlone niepoprawnie. To jest kolejny argument za tym, aby używać `<head></head>`.

Nagłówki w tekście

Taki monotonny tekst nie jest zbyt interesujący. Dlatego warto w dokumencie wstawić jakiś nagłówek (tytuł), napisany większą czcionką. Do tego celu służą znaczniki od `<h1>` do `<h6>`, gdzie `<h1>`, to nagłówek pisany największą czcionką. Oto prosty przykład:

```
<h1>Naglowek 1</h1>
<p>bla bla bla</p>
<h2>Naglowek 2 (mniejszy)</h2>
<p>bla bla bla</p>
<h6>najmniejszy naglowek</h6>
<p>bla bla bla</p>
```

A oto efekt (w różnych przeglądarkach nagłówki mogą mieć inne rozmiary, jeśli chciałbyś, aby w każdej przeglądarce nagłówki wyglądały tak samo albo np. uważasz, że pierwszy nagłówek jest zbyt duży, to poczytaj sobie coś o stylach CSS). A oto efekt:



Odnośniki (hiperłącza)

Sama strona, bez możliwości tworzenia hiperłączy jest dość nudna. Zazwyczaj każdy większy serwis składa się z wielu podstron, które są połączonych hiperłączami. Gdy klikniesz w hiperłącze, to w przeglądarce załaduje się inna strona. Prosty przykład: wchodzisz np. na Onet.pl i na głównej stronie masz tytuły różnych wiadomości. Gdy klikniesz w jakiś link, to załaduje się strona z danym artykułem. Oto prosty przykład stworzenia odnośnika:

```
<a href="http://www.onet.pl">Pierwszy polski portal</a>
```

W przeglądarce zobaczysz tekst:

```
Pierwszy polski portal
```

Gdy klikniesz w link, zostaniesz przeniesiony na stronę Onetu. Można również tworzyć linki do podstron swojego serwisu np. gdy chcesz przekierować kogoś na podstronę o nazwie **plik.htm,** to użyjesz następnego kodu:

```
<a href="plik.htm">Kliknij</a>
```

Musisz tylko pamiętać, aby w tym samym katalogu znajdował się plik o nazwie **plik.htm**. Przypomnij sobie też, co wcześniej pisałem o wielkości liter w nazwach plików!

Tabele

Tabele w HTML służą do prezentacji danych w formie tabelarycznej.

Oto prosty przykład wyświetlający tabele:

```
<table border="1" width="400">
      <tr>
            <td><b>Nazwisko</b></td>
            <td><b>Telefon</b></td>
      </tr>
      <tr>
            <td>Pan Policjant</td>
            <td>997</td>
      </tr>
      <tr>
            <td>Pan strażak</td>
            <td>998</td>
      </tr>
</table>
```

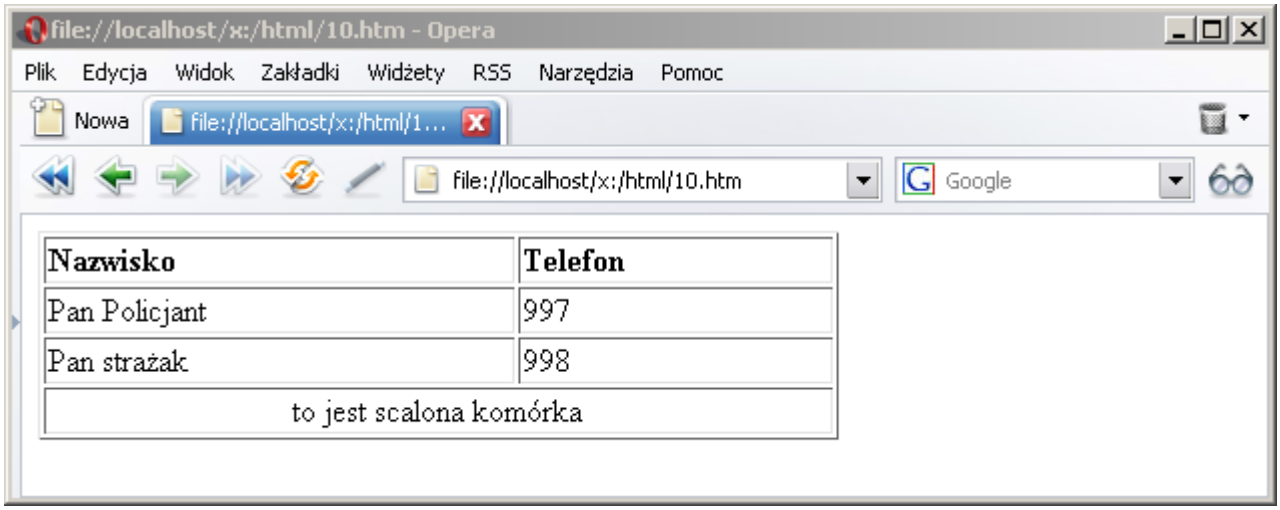
Oto efekt:

W pierwszej linijce definiujemy parametry tabeli (obramowanie na 1 pixel, szerokość tabeli 400 pixeli). Znacznik **<tr>** otwiera nową „linijkę”, a wewnątrz niego znaczniki **<td>** otwierają nowe „kolumny”. Każdy wiersz tabeli musi mieć tyle samo kolumn. Jeśli chcemy, aby dany wiersz tabeli miał mniej kolumn, to należy scalić dwie kolumny (atrybut colspan). Oto przykład tabeli ze scaloną kolumną:

```
<table border="1" width="400" id="table1">
	<tr>
		<td><b>Nazwisko</b></td>
		<td><b>Telefon</b></td>
	</tr>
	<tr>
		<td>Pan Policjant</td>
		<td>997</td>
	</tr>
	<tr>
		<td>Pan strażak</td>
		<td>998</td>
	</tr>
	<tr>
		<td colspan="2">
```

```
            <p align="center">to jest scalona komórka</td>
        </tr>
</table>
```

Efekt:



Podstawowe wiadomości o PHP

Czym jest PHP?

PHP jest językiem skryptowym, który służy do rozszerzania możliwości stron internetowych. Jego składnia jest podobna do popularnych języków programowania C/C++, ale programista PHP jest w nieco lepszej sytuacji - zazwyczaj nie musi przejmować się poprawnością typów zmiennych, przydzielaniem dla nich pamięci itp. Dodatkowo wbudowana obsługa wielu popularnych baz danych, ułatwia twórcy operacje na tych bazach. Dzięki połączeniu z biblioteką GD możliwe jest także dynamiczne tworzenie obrazków GIF (starsze wersje GD) lub PNG (nowsze wersje).

Jakie jest praktyczne zastosowanie PHP?

Wyobraź sobie, że chcesz stworzyć serwis ogłoszeniowy, w którym każdy może zamieścić jakieś anons. W najprostszej wersji wcale nie trzeba korzystać z PHP – wystarczy, że ktoś wyśle do Ciebie e-mail, a Ty ręcznie wpiszesz treść ogłoszenia do pliku HTML. Niestety, musisz wtedy codziennie sprawdzać, które ogłoszenia są nieaktualne (np. usuwasz wszystkie ogłoszenia mające więcej niż 14 dni). W przypadku małego serwisu można te operacje wykonywać ręcznie. Jednak gdyby w serwisie było kilkaset kategorii (kilkaset plików HTML), a codziennie pojawiało się kilkaset nowych ogłoszeń (i kilkaset traciło ważność), to musiałbyś kilka godzin dziennie spędzać na aktualizowaniu serwisu.

PHP pozwala na zautomatyzowanie tych czynności. PHP, to jakby połączenie języka programowania ze stroną internetową. Gdy ktoś chce dodać ogłoszenie, to wypełnia formularz i naciska przycisk. Po naciśnięciu przycisku jest wykonywany odpowiedni kod, który powoduje dodanie treści nowego ogłoszenia do bazy danych. Natomiast dla osoby, która przegląda ogłoszenia zostanie wykonany kod pobierający z bazy danych ogłoszenia z danej kategorii i wyświetlający je w przeglądarce internetowej. Osobny skrypt wykonywany raz na dobę ma za zadanie usuwać z bazy danych ogłoszenia mające więcej niż 14 dni

Jak widać, udało się zautomatyzować szereg czynności takich jak dodawanie nowego ogłoszenia, wyświetlanie ogłoszeń z danej kategorii i usuwanie nieaktualnych ogłoszeń. Administrator strony nie musi nic robić, bo wszystko się wykonuje automatycznie.

Oczywiście to jest bardzo prosty przykład zastosowania PHP. PHP ma dużo większe możliwości, można bez trudu napisać sklep internetowy, który da się zintegrować z jakimś systemem płatności (np. Platnosci.pl, AllPay.eu, PayPal). Sklep może wystawiać klientowi faktury VAT (np. w postaci pliku PDF), a także generować naklejki z adresem klienta na paczki.

Jednym z najpopularniejszych skryptów napisanych w języku PHP jest znane wszystkim forum oparte o skrypt phpBB2

Różnice pomiędzy PHP a JavaScript (JS)

PHP to język wykonywany po stronie serwera, natomiast JavaScript jest wykonywany po stronie klienta. Użytkownik może wyłączyć wykonywanie skryptów JavaScript, natomiast nie może wyłączyć wykonywania PHP, bo PHP jest interpretowany na serwerze i strona wygenerowana przez skrypt PHP dopiero wtedy jest wysyłana do przeglądarki.

Najlepiej widać to na prostym przykładzie: skrypt wyświetlający aktualną datę i godzinę. Skrypt napisany w JS wyświetli datę i godzinę, którą masz ustawioną w komputerze, natomiast skrypt napisany w PHP wyświetli datę i godzinę, która jest na serwerze, na którym jest skrypt. Różnica jest widoczna wtedy, gdy serwer, na którym jest skrypt napisany w PHP leży w kraju znajdującym się w innej strefie czasowej.

Inna ważna różnica jest związana z przechowywaniem danych. Skrypty pisane w JS nie mogą przechowywać danych na serwerze, na którym jest strona, mogą jedynie przechowywać proste dane w plikach cookies. Tak więc w JS nie da się napisać takich skryptów jak np. księga gości (gdzie każdy może się dopisać i każdy może przeglądać wpisy), forum itp.

Co trzeba posiadać, aby móc programować w PHP?

Do samego programowania wystarczy zwykły Notatnik. Natomiast, aby skrypt został wykonany na serwerze, musi być zainstalowany interpreter języka PHP. Ponieważ PHP jest bardzo popularny (i darmowy) to każda porządna firma oferująca konta na strony www oferuje PHP (można wręcz powiedzieć, że PHP jest standardem). Jednak tutaj pojawia się pewien problem – bardzo niewygodnie się edytuje pliki:

- modyfikujesz plik (np. usuwasz literówkę w nazwie zmiennej)
- łączysz się z serwerem FTP i wrzucasz do odpowiedniego katalogu plik PHP
- w przeglądarce internetowej wpisujesz adres danego pliku PHP

Proces można uprościć instalując u siebie na swoim komputerze interpreter PHP, wtedy testowanego pliku nie trzeba będzie wrzucać na inny serwer, bo plik zostanie zinterpretowany przez

Twój komputer. Dopiero jak skończysz pisać skrypt, to go wrzucisz w docelowe miejsce.

Osobiście polecam gotowy pakiet WAMP (http://www.wampserver.com/en/index.php) zawierający serwer Apache, interpreter języka PHP i bazę danych MySQL. Instalacja tego pakietu jest bardzo wygodna i ogranicza się do kilku kliknięć myszą. Mało tego nie trzeba ręcznie instalować każdego składnika z osobna i go konfigurować, bo instalator WAMP zrobi to automatycznie.

Podczas instalacji zostaniesz poproszony o podanie folderu, w którym trzymasz strony internetowe. Ja podałem **f:\strony**

Pierwszy prosty skrypt

Po przyswojeniu podstawowej teorii, czas na napisanie pierwszego skryptu.

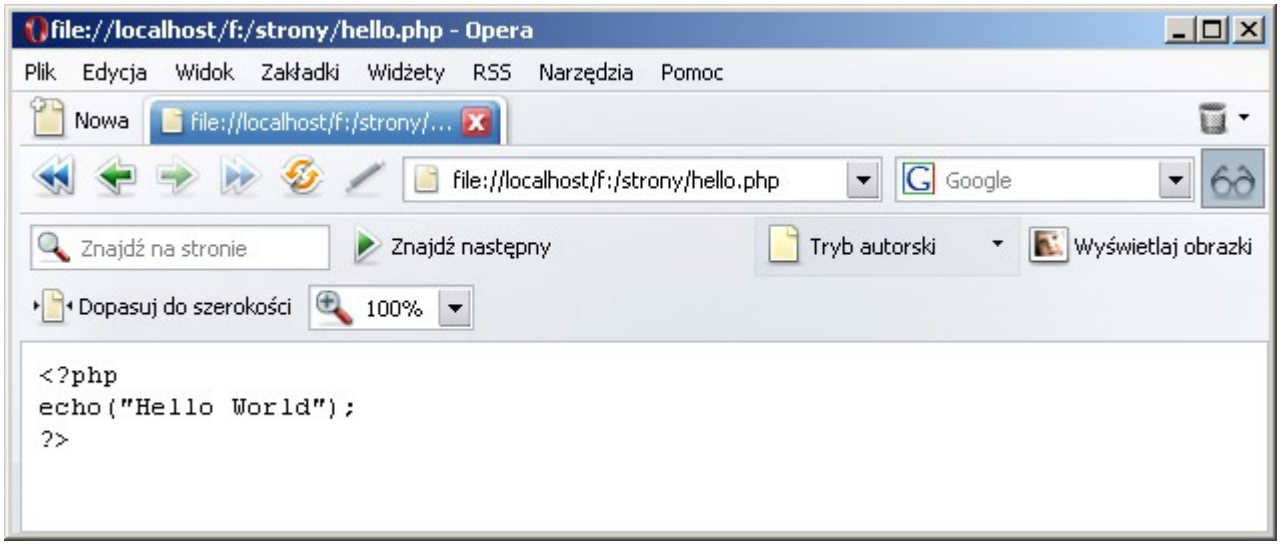
Zanim zaczniemy pisać prosty skrypt, należy uruchomić WAMP. (menu Start | Programy | WampServer | start Wampserver). Zapamiętaj, że zawsze, gdy chcesz uruchomić jakiś skrypt PHP, WAMP musi chodzić. Uwaga! Zauważyłem, że WAMP nie chodzi prawidłowo, gdy jest włączony Skype. Zatem zawsze przed uruchomieniem WAMP wyłącz Skype.

Skrypt ten będzie wyświetlał na ekranie komunikat **Hello World!** Oto treść skryptu:

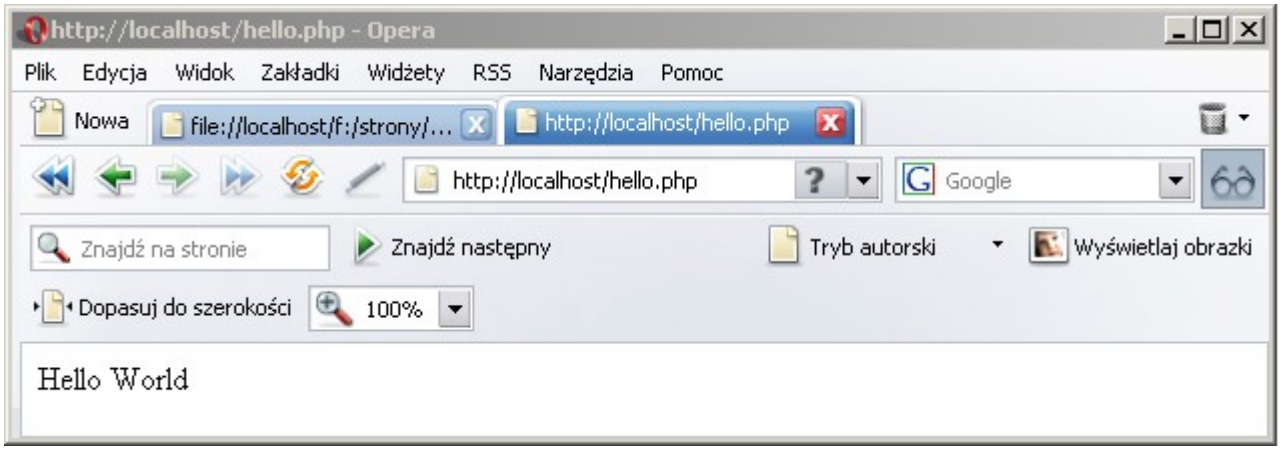
```
<?php
echo("Hello World");
?>
```

Tak stworzony plik należy zapisać w folderze, który podczas instalacji WAMP ustaliłeś jako folder, w którym trzymasz strony). Plik powinien mieć rozszerzenie .**php**. Ja plik nazwałem **hello.php**

Jeśli w przeglądarce internetowej wpiszesz adres: **f:\strony\hello.php**, to zobaczysz, że zostanie wypisane źródło strony (a przecież nie o to nam chodziło, tylko o wykonanie skryptu!)



Dlaczego tak się dzieje? Otóż WAMP nie *przechwycił* tego pliku PHP i go nie zinterpretował. Aby WAMP zinterpretował dany plik PHP, należy go wywołać w inny sposób, wpisując w przeglądarce taki adres: **http://localhost/hello.php** (zamiast localhost można wpisać 127.0.0.1). Jeśli plik hello.php umieściłbym w folderze **f:\strony\pierwsza\hello.php**, to musiałbym w przeglądarce wpisać **http://localhost/pierwsza/hello.php**. Nie istnieje sposób pozwalający na wykonanie się skryptu, który nie znajduje się wewnątrz folderu **f:\strony** (który podczas instalacji WAMP ustawiłem jako folder, wewnątrz którego przechowuję strony). Oto efekt wykonania się skryptu:



Teraz czas na dokładne omówienie skryptu. Pierwsza linijka zawierająca ciąg znaków **<?php** informuje, serwer, że od tego miejsca, do miejsca, w którym znajduje się ciąg znaków **?>**, znajduje się skrypt PHP. Jak widać, nasz pierwszy skrypt PHP składa się tylko z jednej instrukcji **echo**, która wyświetla na ekranie ciąg znaków **Hello World!** Każda instrukcja w PHP powinna się kończyć średnikiem. Składnia PHP jest bardzo podobna do języka C/C++.

Jeśli zerkniesz do źródła strony, to zobaczysz tylko ciąg znaków: **Hello World!** Dlaczego tak się dzieje? Otóż serwer Apache, ma za zadanie wysłać do klienta chcącego oglądać daną stronę, zawartość danej strony. W przypadku statycznego pliku **.html**, który nie zawiera skryptów pisanych w PHP) proces jest bardzo prosty: do przeglądarki jest wysyłana cała zawartość pliku **.html** (jakby on był kopiowany). Natomiast, gdy klient wywołuję stronę pisaną w języku PHP, to Apache wysyła do przeglądarki zawartość strony WWW, z tym że jeśli natrafi na ciąg znaków **<?php** to wtedy uruchamia interpreter PHP (który interpretuje skrypt, dopóki nie natrafi na ciąg znaków **?>**) i zamiast treści skryptu, do przeglądarki wysyłane są dane wygenerowane przez ten skrypt. W tym prostym przypadku zamiast **echo("Hello World");** do przeglądarki został wysłany efekt tego prostego skryptu, czyli ciąg znaków **Hello World**.

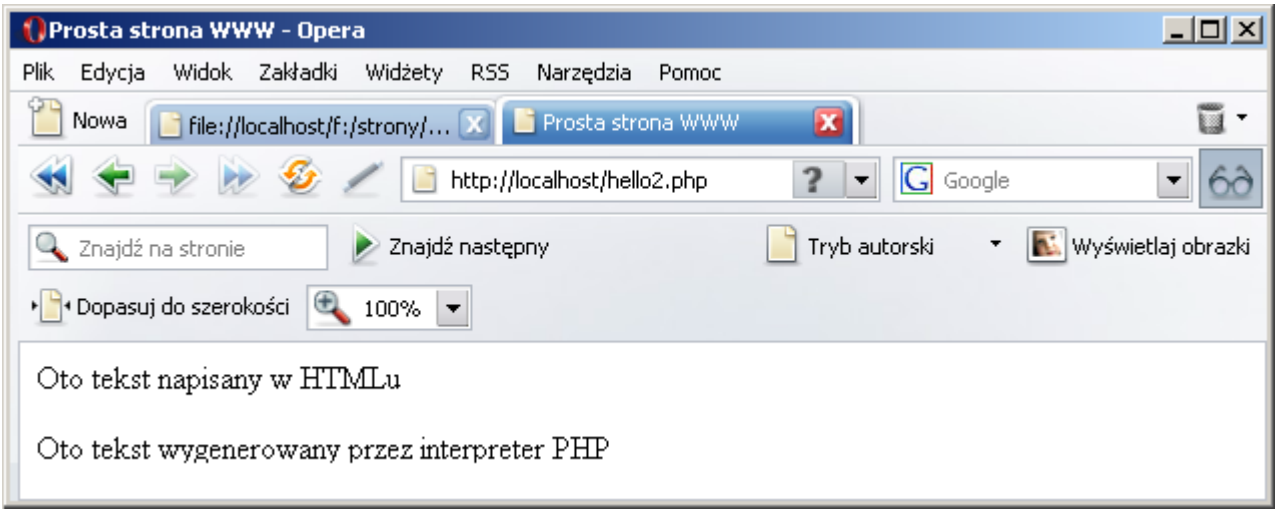
Uwaga: zamiast **<?php** można pisać **<?** jest to krótsza i wygodniejsza forma.

Wplatanie skryptu PHP w znaczniki HTML

Największą zaletą języka PHP jest fakt, że można wplatać go w zwykłe pliki HTML (podobnie jak to jest z JavaScript). Oto przykładowy kod:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1250">
<title>Prosta strona WWW</title>
</head>
<body>
<p>Oto tekst napisany w HTMLu</p>
<?php
echo("<p>Oto tekst wygenerowany przez interpreter PHP</p>");
?>
</body>
</html>
```

Zwróć uwagę na pogrubiony fragment. Nie jest to nic innego jak skrypt PHP, który został wpleciony między znaczniki HTML. Oto przykład wykonania tego kodu:



Pierwsza linijka, to tekst wyświetlony przez HTML, a druga linijka to tekst wyświetlony za pomocą instrukcji **echo**. W każdym pliku HTML może znajdować się dowolna ilość wstawek PHP, nie jest powiedziane, że skrypt PHP musi się zaczynać w pierwszej linijce i kończyć w ostatniej.

Komentarze

W PHP można stosować komentarze. Są dwa rodzaje komentarzy: jednolinijkowy i wielolinijkowy

```
// ten komentarz jest ważny do końca linii

/* Ten
komentarz może
być wielolinijkowy
*/
```

Zmienne

W PHP zmienne nie mają swojego typu. Dana zmienna w jednej chwili może przechowywać liczby zmiennoprzecinkowe, a później łańcuchy znaków. Z jednej strony jest to udogodnienie, bo nie trzeba pamiętać o deklarowaniu wszystkich zmiennych, z drugiej strony taka liberalność czasami sprawia drobne problemy (szczególnie w dłuższych i bardziej skomplikowanych projektach). W PHP nazwy zmiennych poprzedza się znakiem **$** np. **$a**

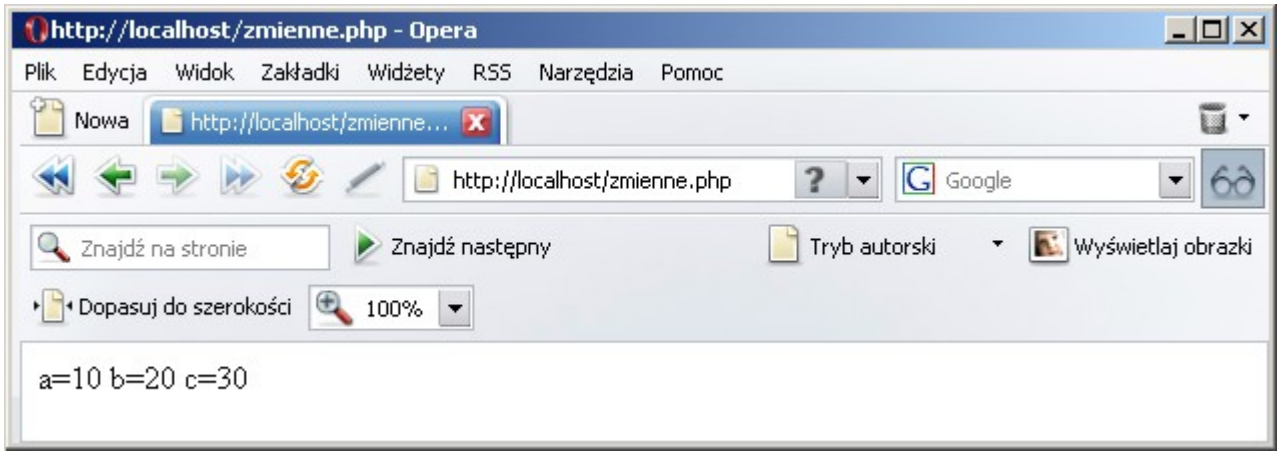
Oto prosty przykład demonstrujący użycie zmiennych

```
<?
$a=10;
$b=20;
$c=$a+$b;
echo("a=$a b=$b c=$c");
?>
```

Pierwsze dwie linijki zawierają deklaracje zmiennych **$a** i **$b**. W trzeciej linijce zmiennej **$c** zostaje przypisana suma zmiennych **$a** i **$b**.

Zwróć uwagę na to, że w instrukcji **echo** znajdują się nazwy zmiennych. Interpreter PHP sam podmieni nazwę zmiennej na jej wartość, czyli zamiast wyświetlić **$a**, wyświetli zawartość zmiennej **$a**. Jeśli w instrukcji **echo** zamiast cudzysłowów, użyłbyś apostrofów, to wtedy PHP nie zinterpretuje nazw zmiennych i jak pojawi się **$a**.

Oto efekt wykonania skryptu:



Operatory

Aby programować w PHP, należy znać kilka podstawowych operatorów umożliwiających takie czynności jak np. dodawanie liczb.

Operatory arytmetyczne

Przykład	Nazwa	Opis
$a + $b	Dodawanie	Suma $a i $b.
$a - $b	Odejmowanie	Różnica $a od $b.
$a * $b	Mnożenie	Iloczyn $a i $b.
$a / $b	Dzielenie	Iloraz $a przez $b.
$a % $b	Dzielenie Modulo	Reszta z dzielenia $a przez $b.

Oto prosty przykład wykorzystujący poznane operatory:

```
<?
$a=11;  //zmiennej $a przypisujemy wartość 11
$b=3;

echo("Suma "); echo($a+$b);
echo("<br>Roznica "); echo($a-$b);
echo("<br>Iloczyn "); echo($a*$b);
echo("<br>Dzielenie "); echo($a/$b);
echo("<br>Reszta z dzielenia "); echo($a%$b);
?>
```

Operatory przypisania i porównania

Przykład	Nazwa	Opis
$a=$b	Przypisanie	Przypisz zmiennej $a zawartość zmiennej $b
$a==$b	Porównanie	Czy zmienna $a jest równa zmiennej $b?
$a+=$b	Dodawanie	Inaczej: $a=$a+$b
$a-=$b	Odejmowanie	Inaczej: $a=$a-$b
$a*=$b	Mnożenie	Inaczej: $a=$a*$b
$a/=$b	Dzielenie	Inaczej: $a=$a/$b
$a%=$b	Dzielenie Modulo	Inaczej: $a=$a%$b

Operator przypisania wykorzystaliśmy w poprzednim przykładzie pisząc: $a=11, czyli zmiennej $a przypisaliśmy wartość 11. Operator porównania (==) poznasz w praktyce, gdy będę omawiał instrukcję warunkową IF. Natomiast poniżej jest prosty przykład wykorzystujący operatory przypisania:

```
<?
$a=11;
$b=3;
echo("a=a+b ");
$a+=$b;  //można również napisać $a=$a+$b;
echo($a);
echo("<br>a=a-b");
$a-=$b; //można również napisać $a=$a-$b
echo($a);
?>
```

Uwaga: pomiędzy `$a-=$b` a `$a=-$b` jest pewna subtelna różnica. Otóż `$a-=$b` to skrócony zapis wyrażenia `$a=$a-$b`, natomiast `$a=-$b` oznacza, że pod zmienną `$a` podstawić trzeba zmienną `$b` i zmienić jej znak na przeciwny, czyli w tym przypadku przypisalibyśmy do zmiennej `$a` liczbę -3

Operatory inkrementacji i dekrementacji

Przykład	Nazwa	Opis
++$a	Pre-inkrementacja	Najpierw zwiększa wartość $a o jeden, potem zwraca $a.
$a++	Post-inkrementacja	Najpierw zwraca $a, potem zwiększa $a o jeden.
--$a	Pre-dekrementacja	Najpierw zmniejsza wartość $a o jeden, potem zwraca $a.
$a--	Post-dekrementacja	Najpierw zwraca $a, potem zmniejsza $a o jeden.

Inkrementacja, to zwiększenie wartości zmiennej o 1. Dekrementacja, to zmniejszenie wartości zmiennej o 1. Zamiast $a++ można napisać $a=$a+1. Zamiast $a-- można napisać $a=$a-1, jednak ta krótsza forma jest wygodniejsza.

A teraz wyjaśnię, czym się różni $a++ od ++$a (i analogicznie $a-- i --$a). Przeanalizuj ten krótki skrypt:

```
<?
echo("Post-inkrementacja: ");
$a=10;
$c=$a++;
echo($c);
echo("<br>Pre-inkrementacja: ");
$a=10;
$c=++$a;
echo($c);
?>
```

W pierwszym wypadku zmiennej $c zostanie przypisana liczba 10, a w drugim 11. Dlaczego? Otóż w pierwszym wypadku zmiennej $c zostanie przypisana wartość 10, a następnie zmienna $a zostanie zwiększona o 1. Natomiast w drugim przypadku, najpierw zmienna $a zostanie zwiększona o 1, a dopiero później do zmiennej $c zostanie przypisana wartość zmiennej $a.

Jeśli do końca tego nie rozumiesz, to się nie przejmuj, zawsze można tę jedną linijkę zawierającą dwie operacje rozbić na dwie linijki:

```
$a++;
$c=$a;
```

i już nie będziesz miał wątpliwości, która operacja się wykona jako pierwsza. Nawet, jeśli nie będziesz w swoich skryptach używał tego typu konstrukcji, to warto wiedzieć, że coś takiego jest, bo czasami może zajść potrzeba, że będziesz modyfikował cudzy skrypt.

Operatory logiczne

Przykład	Nazwa	Opis
$a and $b	I	**TRUE** jeśli zarówno $a jak i $b są **TRUE**.
$a or $b	Lub	**TRUE** jeśli $a lub $b jest **TRUE**.
$a xor $b	Wyłącznie-Lub	**TRUE** jeśli $a lub $b jest **TRUE**, ale nie jednocześnie.
! $a	Nie	**TRUE** jeśli $a nie jest **TRUE**.
$a && $b	I	**TRUE** jeśli zarówno $a jak i $b są **TRUE**.
$a || $b	Lub	**TRUE** jeśli $a lub $b jest **TRUE**.

Operatory logiczne nie są zbyt często używane (szczególnie w prostych skryptach), ale czasami przydają się. Mógłbym tutaj nic nie pisać, że istnieje coś takiego, ale warto jest o tym wiedzieć, gdy będziesz analizował skrypt napisany przez kogoś innego.

Otóż w informatyce bardzo popularny jest system dwójkowy, gdzie dana zmienna może mieć tylko dwa stany: prawda (TRUE) lub fałsz (FALSE). Można to porównać np. do włącznika światła: albo on jest włączony (TRUE) albo wyłączony (FALSE).

PHP może dowolną wartość liczbową traktować jako zmienną logiczną (!). Wtedy, gdy zmienna przyjmuje wartość 0, to PHP traktuje ją jako FALSE, a gdy zawartość zmiennej jest różna od zera, to interpreter traktuje ją jako TRUE.

Poniżej przedstawiam prosty przykład wykorzystujący niektóre z poznanych operatorów. Jeśli nigdy wcześniej nie programowałeś, to proponuję pominąć ten przykład i wrócić do jego analizy, gdy poznasz instrukcję warunkową IF.

```
<?
$a=true; //można też napisać $a=1;
$b=false; //można też napisać $b=0;

if ($a==true){
echo("A jest TRUE<br>");
}
$a=!$a; //negacja - gdy $a jest TRUE, to $a przypisz FALSE, natomiast, gdy $a
jest FALSE, to $a przypisz TRUE

if ($a==false){
echo("A teraz A jest FALSE<br>");
}
if ($a==false and $b==false){
echo("Obie zmienne sa false<br>");
}
if ($a==false or $b==false){
echo("przynajmniej jedna ze zmiennych jest FALSE<br>");
}
?>
```

Jak widać operatory AND i OR są bardzo przydatne. Otóż możemy chcieć wykonać dany fragment programu gdy wszystkie warunki są spełnione (AND) lub gdy jest spełniony przynajmniej jeden warunek (OR).

Operacje na stringach

Przykład	Nazwa	Opis
$a = $b . $c	łączenie stringów	Do stringu $a zapisz połączenie dwóch stringów: $b i $c
$a .=$b	łączenie stringów	Do stringa $a doklej zawartość stringa $b

Zmienne mogą przyjmować nie tylko wartości liczbowe i logiczne. Są jeszcze zmienne, które mogą przechowywać ciągi znaków. W informatycznym żargonie takie zmienne się nazywa stringami (czasami w polskojęzycznej literaturze informatycznej można się spotkać z pojęciem literał). Oto prosty przykład demonstrujący użycie stringów:

```
<?
$imie="Damian "; //wygląda prawie tak jak $a=1;
$nazwisko="Daszkiewicz";
echo("Imie: "); echo($imie); //instrukcja echo wygląda dziwnie znajomo
echo("<br>Nazwisko: "); echo($nazwisko);

//Łączenie stringów
$metryka=$imie.$nazwisko; //wygląda to podobnie jak $c=$a+$b

echo("<br>Metryka: ");
echo($metryka);
?>
```

Źródło: http://php.net.pl/manual/pl/language.operators.php

Wyświetlanie daty i godziny

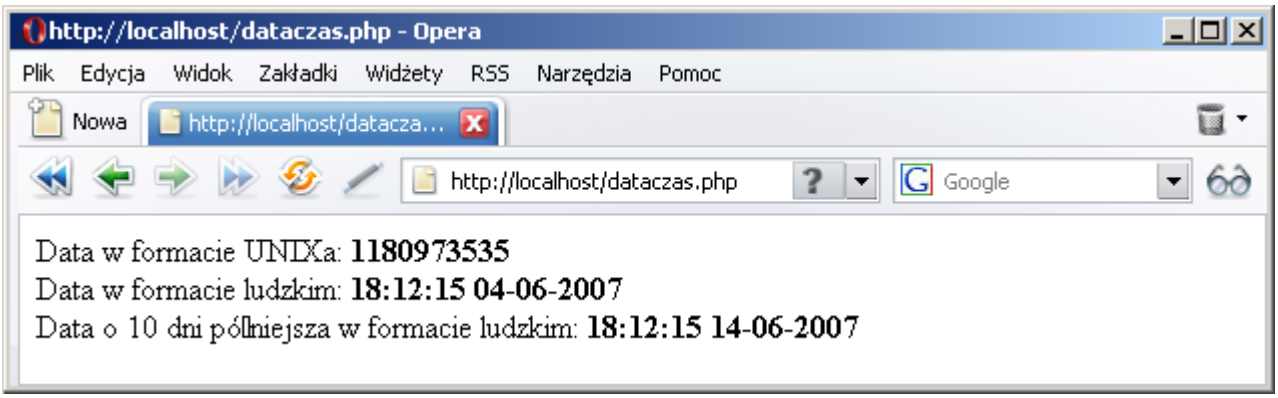
PHP przechowuje datę w zmiennej **long** jako ilość sekund, które upłynęły od północy 1 stycznia roku 1980. Jest to bardzo wygodny sposób zapisu dat, bo chcąc obliczyć datę np. o 1 dzień późniejszą wystarczy do zmiennej dodać liczbę 86400 (60*60*24).

Oto prosty przykład:

```
<?
$czas=date("U"); //odczytaj date do zmiennej $czas
echo("Data w formacie UNIXa: <b>".$czas."</b><br>");
echo("Data w formacie ludzkim: <b>".date("H:i:s d-m-Y")."</b><br>");
$czas+=60*60*24*10;
echo("Data o 10 dni późniejsza w formacie ludzkim: <b>".date("H:i:s d-m-
Y",$czas)."</b><br>");
?>
```

Funkcja **date** z parametrem **U** zwraca do zmiennej **$czas** ilość sekund, które upłynęły od początku roku 1980. W następnych linijkach funkcja **date** wyświetla czas i datę w ludzkim formacie. Pewnie zastanawiasz się, co oznacza dość dziwny format: `H:i:s d-m-Y` przy dacie? Oznacza to format, w jakim ma być wyświetlona data (godzina dwukropek minuta dwukropek sekunda dwukropek spacja dzień kreska miesiąc kreska rok).

Najciekawsze jest obliczanie daty z przeszłości lub z przyszłości, po prostu dodajemy/odejmujemy odpowiednią ilość sekund, a za pomocą instrukcji **date** z odpowiednimi parametrami, na ekranie pojawi się data przedstawiona w ludzkim formacie. Poniżej przedstawiam efekt działania programu:



Poniżej przedstawiam dokładnie jakie parametry można wstawić do funkcji **date**:

- a - "am" lub "pm"
- A - "AM" lub "PM"
- B - Czas internetowy Swatcha
- d - dzień miesiąca, 2 cyfry z zerem na początku; tzn. od "01" do "31"
- D - dzień tygodnia, tekst, 3 litery; n.p. "Fri"
- F - miesiąc, tekst, pełna nazwa; np. "January"
- g - godzina, format 12-godzinny bez zera na początku; tzn. od "1" do "12"
- G - godzina, format 24-godzinny, w jakim początku; tzn. od "0" do "23"
- h - godzina, format 12-godzinny z zerem na początku; tzn. od "01" do "12"
- H - godzina, format 24-godzinny z zerem na początku; tzn. od "00" do "23"
- i - minuty; tzn. od "00" do "59"
- I (duża litera i) - "1" jeśli czas oszczędzania światła słonecznego (w Polsce - czas letni), "0" jeśli czas standardowy (w Polsce - zimowy)
- j - dzień miesiąca bez zera na początku; tzn. od "1" do "31"
- l (mała litera 'L') - dzień tygodnia, tekst, pełna nazwa; np. "Friday"
- L - "1" jeśli rok przestępny, "0" w przeciwnym razie
- m - miesiąc; tzn. "01" to "12"
- M - miesiąc, tekst, 3 litery; np. "Jan"
- n - miesiąc bez zera na początku; tzn. "1" to "12"
- O - różnica w stosunku do czasu Greenwich; np. "+0200"
- r - data sformatowana według RFC 822; np. "Thu, 21 Dec 2000 16:01:07 +0200" (dodane w PHP 4.0.4)
- s - sekundy; i.e. "00" to "59"
- S - standardowy angielski sufiks liczebnika porządkowego, 2 litery; tzn. "st", "nd", "rd" lub "th"
- t - liczba dni w danym miesiącu; tzn. od "28" do "31"
- T - strefa czasowa ustawiona na tej maszynie; np. "EST" lub "MDT"
- U - liczba sekund od uniksowej Epoki (1 stycznia 1970 00:00:00 GMT)
- w - dzień tygodnia, liczbowy, tzn. od "0" (Niedziela) do "6" (Sobota)
- W - numer tygodnia w roku według ISO-8601, tydzień zaczyna się w poniedziałek (dodane w PHP 4.1.0)

- Y - rok, 4 liczby; np. "1999"
- y - rok, 2 liczby; np. "99"
- z - dzień roku; tzn. od "0" do "365"
- Z - offset strefy czasowej w sekundach (tzn. pomiędzy "-43200" a "43200"). Offset dla stref czasowych na zachód od UTC (południka zero) jest zawsze ujemny a dla tych na wschód od UTC jest zawsze dodatni.

Źródło: http://php.net.pl/manual/pl/function.date.php

A oto program, który do odpowiednich zmiennych z aktualnej daty, wydobywa dzień miesiąc i rok:

```
<?
$dzien=date("d");
$miesiac=date("m");
$rok=date("Y");
$godzina=date("H");
$minuta=date("i");
$sekunda=date("s");
echo("$dzien-$miesiac-$rok $godzina:$minuta:$sekunda");
?>
```

Posiadając poszczególne składniki daty w osobnych zmiennych, można tworzyć bardzo ciekawe skrypty (np. posiadając bazę imion, można pokusić się o napisanie skryptu wyświetlającego dzisiejszych solenizantów).

Pętla for

Pętla **for** jest jedną z najpopularniejszych instrukcji iteracyjnych :-) Jej składnia jest bardzo podobna do składni pętli **for** z języka C/C++

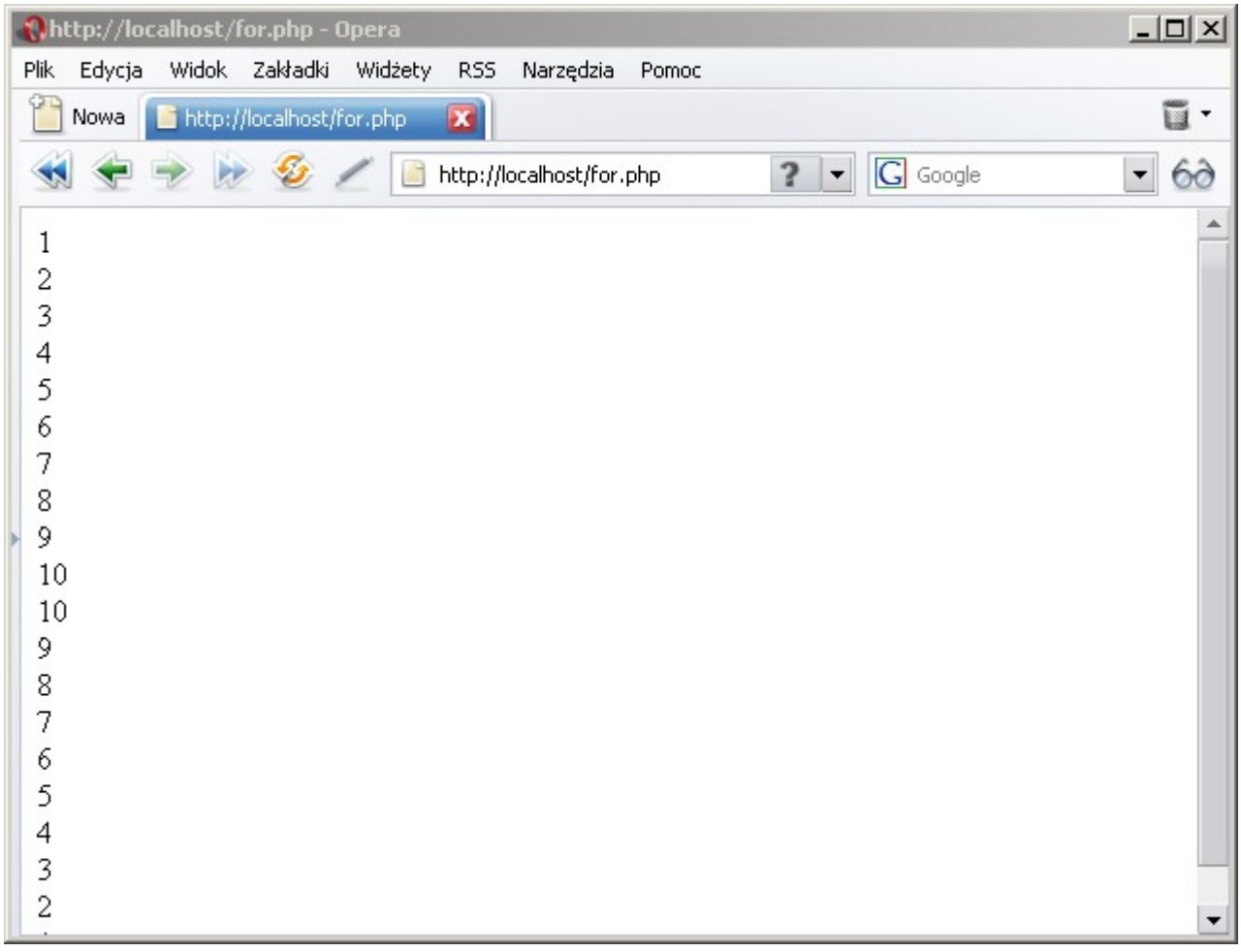
for(wyrażenie_inicjujące ; wyrażenie_testujące ; wyrażenie_modyfikujące)

    wykonywana_instrukcja ;

Oto prosty przykład:

```
<?
for ($a=1; $a<=10; $a++) echo("$a<br>");
for ($a=10; $a>=1; $a--) echo("$a<br>");
?>
```

A oto efekt:



Dwukrotnie w tym przykładzie użyłem pętlę **for** - za pierwszym razem do wyświetlenia liczb od 1 do 10, a za drugim razem, aby wyświetlić liczby od 10 do 1.

Zauważ, że w instrukcji **echo**, oprócz wyświetlenia zawartości zmiennej **$a**, kazałem również wyświetlić znacznik `<br>`. Dzięki temu każda liczba została wyświetlona w nowej linii.