PHP – podstawy

PHP jest językiem skryptowym działającym po stronie serwera. Osadza się go w kodzie HTML w postaci bloków ograniczonych znacznikami <?php ?>, które są przekształcane na HTML podczas każdorazowego odświeżenia strony. Kod PHP jest wykonywany po stronie serwera, który interpretuje składnie i wysyła odpowiednio zmodyfikowany kod HTML. Użytkownik strony może zobaczyć jedynie efekt, nie mając wglądu do napisanego przez nas skryptu.

Język PHP stworzony został w 1994 roku przez Rasmusa Lerdorfa. Jest to produkt Open Source, czyli każdy ma swobodny dostęp do jego kodu źródłowego, który można dowolnie modyfikować i rozprowadzać. Strona główna PHP wraz ze szczegółową specyfikacją to www.php.net.

Wszelkie skrypty napisane w języku PHP wykonywane są po stronie serwera, po czym przeglądarką użytkownika otrzymuje przetworzony kod HTML. Łatwo się domyślić, że bez takiego serwera, zawierającego interpreter PHP, nic nam nie zadziała.

Należy wiec pobrać i zainstalować jeden z dostępnych pakietów, np.: program WebServ, który jest dostępny za darmo. Po zainstalowaniu WebServa wybieramy najnowsze wersje PHP i MySQL, resztę zostawiamy zaznaczona domyślnie. Żeby serwer działał musimy go najpierw włączyć. Jeśli w pasku skrótów po prawej stronie nie ma ikonki serwera, należy znaleźć i uruchomić plik webserv.exe. Znajdzie się on w pasku systemowym (umieszczonym zazwyczaj po prawej stronie na dole), trzeba kliknąć najpierw w ikonkę, po czym klikamy polecenie uruchom. Żeby sprawdzić, czy poprawnie uruchomiliśmy serwer, włączamy przeglądarkę i wpisujemy adres http://localhost/. Jeśli pojawi się strona powitalna, znaczy to, że wszystko jest gotowe do rozpoczęcia nauki.

Jeszcze kilka słów o strukturze w WebServie. Wchodząc do katalogu, w którym zainstalowaliśmy serwer, interesować nas będą głównie dwa miejsca. Jedno nazywa się httpd, do którego wrzucamy wszelkie pliki, które mają być dostępne pod adresem http://localhost/. Jeśli natomiast chcemy podzielić naszą pracę na kilka kategorii, wchodzimy do katalogu o nazwie httpd-users. Tworzymy wewnątrz nowy katalog, który nazywamy dowolnie, np. zmienne. Wszystkie pliki w katalogu "zmienne" będą dostępne pod adresem http://localhost/~zmienne/.

Na początek napiszmy kilka linijek w tradycyjnym HTMLu, np.:

```
<html>
<head>
<title>Pierwszy skrypt php</title>
</head>
<body>
Przykładowy tekst na stronie.
</body>
</html>
```

Teraz zobaczmy na przykładzie, jak wykonać te sama operacje za pomocą języka php...

```
<html>
<head>
<title>Pierwszy skrypt php</title>
</head>
<body>
<?php
    echo "Przykładowy tekst na stronie";
?>
</body>
</html>
```

Zobaczmy, co zmieniło sie w stosunku do poprzedniej wersji. Po pierwsze pojawiły się sie znaczniki <?php oraz ?>. Można również używać skróconej wersji <? oraz ?>.To chyba najbardziej kluczowa wiedza o php - żeby kod został przetworzony przez interpreter, musi znajdować sie właśnie miedzy takimi oznaczeniami. Teraz zobaczmy, co zostało miedzy nimi wpisane. Komenda echo po prostu wyświetla nam na ekranie to, co wpiszemy jej, jako argument. W tym przypadku jest to ten sam tekst, co w czystym HTML, wiec efekt będzie taki jak poprzednio.

Stałe i zmienne

ważne, czy jest to wartość logiczna, liczba całkowita, wymierna czy tez tekst. Ciekawostka w języku PHP jest brak konieczności deklaracji zmiennych, co jest wymagane w innych językach, jak C czy Pascal. Stała jest podobnym kontenerem do zmiennej, z tym ze, jak sama nazwa wskazuje, jej wartości nie można zmienić. Stałej przypisujemy wartość tylko przy definicji.

```
<?
define("NR_TEL","666258147");
$nr_tel = "792297792";
?>
<html>
<head>
<title>Zmienne i stale</title>
</head>
<body>
```

```
<?
echo NR_TEL;
echo $nr_tel;
 ?>
 </body>
 </html>
```

Patrząc na powyższy kod widzimy dwa przypisania na samym początku. Pierwsze z nich to definicja stałej o strukturze - define("NAZWA_STALEJ", "WARTOSC_STALEJ");, gdzie NAZWA_STALEJ to dowolny ciąg liter, cyfr oraz znaku podkreślenia, a WARTOSC_STALEJ to nadana wartość. Dodatkowo nazwa nie może zaczynać się liczbą. Z racji tego, że PHP rozróżnia wielkość liter, stałe NR_TEL i NR_Tel to dwa różne kontenery. Linijka niżej to definicja zmiennej. Nazwy zmiennych kierują się tymi samymi zasadami, co stałych, z tą jedną różnicą, że nazwę zmiennej musimy poprzedzić znakiem $. Do nadawania wartości zmiennym używa się operatora przypisania " = ". Jego zadanie polega na przypisaniu wartości znajdującej się po prawej stronie operatora do zmiennej po lewej stronie.

Użycie stałych i zmiennych

Stworzyliśmy naszą zmienną oraz stałą, przypisaliśmy im wartości i co dalej z nimi począć? Oczywiście można je wyświetlić na ekranie za pomocą komendy echo, tak jak zostało to pokazane na przykładzie. Nie jest to jednak jedyna możliwość. Stała lub zmienna jest traktowana przez PHP jako liczba, ciąg znaków lub wartość logiczna - zależnie co jej przypisaliśmy. Skoro tak, to przypisując zmiennej $liczba = 5; będziemy mogli jej używać tak jak tradycyjnej liczby. Będzie można ją dodać, odjąć, mnożyć, dzielić itp.

Jeszcze jedna istotna sprawa odnośnie przypisywania wartości. Mianowicie możemy przypisać zmiennej $zm = 15; lub $zm = "15";. W czym tkwi różnica? Wszystkie ciągi znaków (potocznie zwane stringami) są zapisywane z wykorzystaniem znaków cudzysłowu (") lub apostrofu ('). W PHP nie deklarujemy typów zmiennych, dlatego interpreter naszego kodu musi w jakiś sposób wywnioskować, czy chcemy traktować 15 jako liczbę, czy też jako stringa.

Istnieje jeszcze grupa zmiennych, o których chciałbym wspomnieć, mianowicie zmienne formularza. Pisząc formularz w HTML należy później obsłużyć wypełnione pola, np. wysłać maila lub zalogować użytkownika. Istnieją dwie metody wysyłania danych - POST oraz GET. Do obsłużenia danych wysyłanych pierwszą metodą służy zmienna $_POST['nazwa_pola']. Odpowiednio dla drugiej metody będzie to $_GET['nazwa_pola']. Mając więc pole tekstowe o nazwie 'imie' w formularzu GET, po wysyłaniu formularza pojawi się zmienna $_GET['imie'] zawierająca wpisaną treść przez użytkownika.

Operatory

Pierwszą kategorią operatorów, którą chciałem tutaj przedstawić, są operatory arytmetyczne. Ich użycie jest bardzo intuicyjne, a działanie niemal oczywiste. Ich zestawienie wygląda następująco:

" + " zwraca sumę dwóch liczb lub ciągów,

" - " zwraca różnicę,

" * " zwraca iloczyn,

" / " zwraca iloraz,

" % " zwraca resztę z dzielenia.

Drugą kategorię stanowią operatory porównania. PHP umożliwia nam sprawdzenie, czy dwie zmienne są sobie równe, lub jeśli nie, to która jest większa, a która mniejsza. Do tego celu służą właśnie operatory porównania. Są nimi:

" == " sprawdza, czy dwie zmienne są równe, co do wartości,

" != " sprawdza, czy zmienne są różne co do wartości,

" === " sprawdza, czy zmienne są identyczne,

" !== " sprawdza, czy zmienne są nieidentyczne,

" > " sprawdza, czy zmienna z lewej strony jest większa od zmiennej z prawej strony,

" < " sprawdza, czy zmienna z prawej strony jest większa od zmiennej z lewej strony,

" >= " sprawdza, czy zmienna z lewej strony jest większa bądź równa od zmiennej z prawej strony,

" <= " sprawdza, czy zmienna z prawej strony jest większa bądź równa od zmiennej z lewej strony.

Trzecią są operatory logiczne. Służą głównie do sprawdzania warunków, o których mowa w kolejnym rozdziale, gdzie zajmiemy się nimi bardziej szczegółowo. W tym miejscu napiszę jedynie że są trzy do wyboru:

" ! " operator zaprzeczenia (logiczne NOT),

" && " operator koniunkcji (logiczne AND),

" || " operator alternatywy (logiczne OR).

Odpowiednikiem operatorów arytmetycznych są tzw. operatory inkrementacji i dekrementacji, które w bardzo szybki i przyjemny sposób zwiększają lub zmniejszają wartość naszej zmiennej o 1. Dzielą się one na:

" $i++ " postinkrementację, zwiększa wartość zmiennej o 1,

" ++$i " preinkrementację, zwiększa wartość zmiennej o 1,

" $i-- " postdekrementację, zmniejsza wartość zmiennej o 1,

" --$i " predekrementację, zmniejsza wartość zmiennej o 1.

Różnica między pre- i post- inkrementacją leży w momencie zwiększenia wartości. Preinkrementacja zwiększa wartość przed wykonaniem polecenia, natomiast postinkrementacja zwiększa wartość zmiennej po wykonaniu polecenia. Z dekrementacją jest analogicznie, z tym że wartość jest zmniejszana.

Innym sposobem zmiany wartości zmiennej jest użycie operatorów przypisania. Chcąc zwiększyć daną wartość o pewną liczbę, zamiast pisać $i = $i + 7, możemy od razu przypisać zmiennej $i wartość o 7 większą, czyli $i += 7. Poniżej prezentuję inne możliwości przypisania zmienionej wartości:

" $i+=5 " zwiększenie wartości o 5,

" $i-=5 " zmniejszenie wartości o 5,

" $i*=5 " przypisanie wartości 5 razy większej,

" $i/=5 " przypisanie wartości 5 razy mniejszej,

" $i%=5 " przypisanie wartości reszty z dzielenia zmiennej przez 5.

Wartym przedstawienia operatorem jest operator obsługi błędów "@", który powoduje ukrycie komunikatu o błędach. Na razie jednak nie warto sobie nim zaprzątać głowy. Zostanie on omówiony na późniejszych lekcjach. Operator ciągu " . " łączy nam dwa ciągi w jeden. Np. $x = "Kod ", $y = "PHP", echo $x.$y wyświetli nam na ekranie "Kod PHP".

Stosowanie komentarzy

Tematem tej lekcji są komentarze, rzecz, bez której bardzo trudno pracować przy dużych projektach programistycznych. Nie tyczy się to wyłącznie PHP, ale każdego języka programowania. Zasada jest prosta - komentujmy wszystko, co może stać się niejasne po pewnym czasie. Na przykład napisanie kodu pewnej witryny zajęło tysiąc linijek, w których użyto pięćdziesięciu zmiennych. Zleceniodawca przez pierwszy rok był bardzo zadowolony, lecz później stwierdził, że zamiast wyświetlać na stronie głównej dziesięć najnowszych produktów, chciałby, żeby było ich dwadzieścia i to w dodatku losowych. Zobaczmy więc jakie problemy możemy napotkać na swojej drodze.

Jeśli nazwaliśmy nasze zmienne $i, $j, $k, $zmienna1 itp. to bardzo ciężko będzie nam znaleźć odpowiedni fragment kodu. Musimy analizować krok po kroku składnię, żeby wywnioskować, co się dzieje w danym miejscu. Wierzcie mi, że mimo iż podczas pisania wszystko wydaje się oczywiste, to po miesiącu już takie nie będzie. Wtedy z pomocą przychodzą nam komentarze. Jeśli komentowaliśmy skrupulatnie naszą pracę z pewnością szybko odszukamy odpowiedni fragment. Podsumowując, jeśli $ilosc jest odpowiedzialna za ilość produktów w sklepie, to należy ją obkomentować "Przechowuje ilość wszystkich produktów w sklepie". Wtedy łatwo będzie nam ją odszukać i dokonać potrzebnych modyfikacji.

Komentarze w praktyce

Istnieją dwa sposoby umieszczania komentarzy w kodzie PHP. Pierwszy z nich stosujemy, gdy chcemy obkomentować kilka lub więcej linijek tekstu. Umieszczamy wtedy taki blok tekstowy między znakami " /* " oraz " */ ". Wszystko zawarte pomiędzy tymi znacznikami zostanie zignorowane przez interpreter podczas generowania kodu html. Co ciekawe, komentarze nawet nie zostaną wysłane do przeglądarki użytkownika. Drugim sposobem komentowania jest umieszczenie tekstu za dwoma ukośnikami " // ". Jest to sposób szybszy i wygodniejszy od pierwszego z racji, że nie trzeba umieszczać znaków zamykających komentarz. Wszystko znajdujące się w jednym wierszu po tym oznaczeniu będzie traktowane jako komentarz. Obsługuje on jednak tylko jedną linijkę, więc jeżeli mamy długi tekst składający się z kilku wierszy, należy użyć pierwszego typu.

Zadanie

Po tej solidnej dawce nowego materiału należałoby go uporządkować. Włączamy więc nasz edytor do pisania dokumentów tekstowych i tworzymy dwa nowe pliki o nazwie sklep.html oraz zamowienie.php.

Naszym zadaniem będzie napisanie aplikacji, która liczy sumę zamówienia składanego w sklepie internetowym, liczy podatek VAT 22% od tej kwoty, a następnie prezentuje na ekranie cenę netto i brutto zamówionych przedmiotów.

Ceny netto wszystkich produktów będą przechowywane w formie stałych o nazwie szablonowej NAZWA_PRODUKTU. Zmienne $ile_nazwa_produktu będą przechowywać informacje odnośnie ilości sztuk danego produktu, zamówionych przez klienta. Wysokość podatku VAT również będzie przechowywana w stałej - P_VAT. Takie rozwiązanie jest bardzo wygodne w przypadku późniejszych zmian podatku. Jeżeli zaistnieje potrzeba zmiany wartości wystarczy to zrobić raz przy deklaracji stałej, a w całym dalszym kodzie wartość zostanie zmieniona. Zrobimy prosty interfejs, zawierający formularz POST. Na podstawie wpisanych w nim danych zostanie obliczone zamówienie. W celach instruktażowych kolejne etapy obliczeń są przypisywane do nowych zmiennych. Nie jest to rozwiązanie optymalne, ale na pewno bardziej przejrzyste.

Najpierw plik sklep.html. Jego kod wygląda tak:

```
<html>
<head>
<title>Sklep odzieżowy</title>
</head>
<body>
<form action="zamowienie.php" method="post">
Liczba zamawianych koszulek: <input type="text"
 name="koszulki" size=3 maxsize=3 />
Liczba zamawianych spodni: <input type="text"
 name="spodnie" size=3 maxsize=3 />
Liczba zamawianych czapek: <input type="text"
 name="czapki" size=3 maxsize=3 />
<input type="submit" value="złóż zamówienie" />
</body>
</html>
```

Listing pliku zamowienie.php:

```
<?php
define("KOSZULKA", 14.99); // cena koszulki jako stała
define("SPODNIE", 45.99); // cena spodni
define("CZAPKA", 9.63); // cena czapki
define("P_VAT", 0.22); // wysokość podatku VAT
$ile_koszulki = $_POST['koszulki']; // przypisanie zmiennych
formularza
$ile_spodnie = $_POST['spodnie'];
$ile_czapki = $_POST['czapki'];
$kwota_koszulki_netto = $ile_koszulki*KOSZULKA; // wartość netto
zamówionych koszulek
```

```
$kwota_spodnie_netto = $ile_spodnie*SPODNIE; // wartość netto spodni

$kwota_czapki_netto = $ile_czapki*CZAPKA; // wartość netto czapek

$kwota_zamowienia_netto = $kwota_koszulki_netto +
$kwota_spodnie_netto + $kwota_czapki_netto; // cena netto całego
zamówienia

$kwota_koszulki_brutto = $kwota_koszulki_netto +
$kwota_koszulki_netto*P_VAT; // wartość brutto koszulek

$kwota_spodnie_brutto = $kwota_spodnie_netto +
$kwota_spodnie_netto*P_VAT; // wartość brutto spodni

$kwota_czapki_brutto = $kwota_czapki_netto +
$kwota_czapki_netto*P_VAT; // wartość brutto czapek

$kwota_zamowienia_brutto = $kwota_koszulki_brutto +
$kwota_spodnie_brutto + $kwota_czapki_brutto; // cena zamówienia
brutto

 ?>

 <html>

 <head>

 <title>Obsługa zamówienia</title>

 </head>

 <body>

<?php

echo "Cena netto zamówionych koszulek:
".$kwota_koszulki_netto."<br/>";

echo "Cena netto zamówionych spodni: ".$kwota_spodnie_netto."<br/>";

echo "Cena netto zamówionych czapek: ".$kwota_czapki_netto."<br/>";

echo "Wartość netto całego zamówienia:
".$kwota_zamowienia_netto."<br/>";

echo "Cena brutto zamówionych koszulek:
".$kwota_koszulki_brutto."<br/>";

echo "Cena brutto zamówionych spodni:
".$kwota_spodnie_brutto."<br/>";

echo "Cena brutto zamówionych czapek:
".$kwota_czapki_brutto."<br/>";
```

```
echo "Wartość brutto całego zamówienia:
".$kwota_zamowienia_brutto."<br/>";

 ?>

 </body>

 </html>
```

Omówienie skryptu

Plik sklep.php jest odpowiedzialny za wyświetlenie formularza z możliwością wpisania liczby zamawianych przedmiotów. Nic wielkiego się tutaj nie dzieje, po prostu zwykły HTML. Po wpisaniu i zgłoszeniu formularza dane zostają przesłane do pliku zamowienie.php. Tutaj na wstępie definiujemy stałe z cenami przedmiotów oraz wysokością podatku VAT. Później tworzymy zmienne z pól formularza, przesłanych ze sklepu. Kwoty netto obliczamy po prostu mnożąc ilość zamówionych przedmiotów przez cenę netto przechowywaną w stałej. Następnie w celu obliczenia całości sumujemy trzy kwoty netto. Z cenami brutto robi się podobnie, z tym że należy dodać wartość podatku, czyli 0,22*kwota.

To zakańcza naszą część obliczeniową aplikacji, teraz czas to wszystko wyświetlić. Funkcję echo pokazałem już na drugiej lekcji, lecz wyświetlała ona sam tekst. Teraz oprócz tekstu wpisanego przez nas dodatkowo jako argument podajemy zmienną. Jedną i drugą część tekstu musimy jednak połączyć operatorem " . ", żeby interpreter poprawnie zrozumiał nasze intencje. Działa to w ten sposób, że przeglądarka wyświetli najpierw to, co jest w cudzysłowiu, później wyświetli wartość zmiennej, a następnie to, co jest w kolejnym cudzysłowiu. Spokojnie, to nic trudnego. Po kilku napisanych skryptach dojdziesz do wprawy.

Instrukcja warunkowa if.

Powyżej opisany przypadek w prosty sposób rozwiąże zastosowanie instrukcji if. Jej składnia jest następująca:

```
<?php

if (warunek) // w nawiasie podajemy warunek do sprawdzenia

{

  instrukcje

}

?>
```

Sprawdza ona, czy warunek podany w nawiasie został spełniony. Warunki w argumencie można łączyć za pomocą operatorów logicznych, poznanych w rozdziale pierwszym. Jeśli chcesz sobie przypomnieć ich funkcje, kliknij tutaj - operatory PHP. Całkowita wartość nawiasu sprowadza sie do określenia wartości logicznej - TRUE lub FALSE. Jeśli warunek jest spełniony (wartość TRUE) instrukcje zostaną wykonane. Jeśli natomiast nie jest (wartość FALSE) instrukcje zostaną pominięte.

Napiszemy prosty kod, który wyświetli komunikat o parzystości liczby. W zależności, czy wartość zmiennej będzie parzysta, czy tez nie, wykona się inna instrukcja. Zobaczmy skrypt poniżej:

```
<?php
$a = 7; // przypisujemy wartość zmiennej $a
if ($a%2 > 0) // reszta z dzielenia przez 2
{
  echo "Liczba nieparzysta";
}
if ($a%2 == 0) // brak reszty z dzielenia przez 2
{
  echo "Liczba parzysta";
}
?>
```

Myślę, że powyższego przykładu nie trzeba komentować. Wydaje się być oczywisty. W skrócie... Jeśli interpreter natrafia na warunek if, sprawdza wartość logiczną w nawiasie. W przypadku zwrócenia TRUE, wykona się instrukcja w nawiasach klamrowych. Jeżeli natomiast zwrócona zostanie wartość FALSE, kod w nawiasach zostanie pominięty. W przypadku, gdy mamy tylko jedną instrukcję, nie musimy stosować nawiasów klamrowych.

Poprawny będzie również zapis:

```
<?php
$a = 7; // przypisujemy wartość zmiennej $a
if ($a%2 > 0) // reszta z dzielenia przez 2
  echo "Liczba nieparzysta";
if ($a%2 == 0) // brak reszty z dzielenia przez 2
  echo "Liczba parzysta";
?>
```

Warunek if... else...

Poprzednim omawianym przeze mnie tematem był warunek if. Miał on pewna niedogodność. Gdy chcieliśmy rozważyć, czy liczba jest parzysta czy tez nie, musieliśmy sprawdzać dwa warunki. Pierwszym była parzystość liczby, drugim nieparzystość. Musieliśmy użyć dwóch ifów. Nie trzeba być wybitnym matematykiem, żeby wiedzieć, ze kiedy liczba nie jest nieparzysta, to na pewno jest parzysta.

Nie ma potrzeby sprawdzania dwa razy. Wystarczy sprawdzić raz, a w przypadku niepowodzenia wykonać instrukcje alternatywna. Zobaczmy poniższy listing:

```
<?php
$a = 7; // przypisujemy wartość zmiennej $a
if ($a%2 > 0) // reszta z dzielenia przez 2
{
  echo "Liczba nieparzysta";
}
else // brak reszty z dzielenia przez 2
{
  echo "Liczba parzysta";
}
?>
```

Jak łatwo sie domyślić po słowie kluczowym else umieszczamy instrukcje, które wykonają sie w przypadku niespełnienia warunku. W przypadku, kiedy instrukcja ma tylko jedna linijkę, nie musimy umieszczać jej w nawiasach klamrowych, podobnie jak przy zwykłym warunku if. W przypadku braku nawiasów interpreter przeczyta pierwsza napotkana komendę, jako kompletna instrukcje.

Pętla for

Ostatnią omawianą przez nas pętlą będzie for. Jej konstrukcja jest następująca:

```
for($i=0;$i<10;$i++)
{
  // instrukcje do wykonania
  // z każdą iteracją
}
?>
```

Pierwszym elementem w nawiasie jest przypisanie zmiennej iteracyjnej początkowej wartości. Drugim jest warunek końcowy. Trzecim wskazujemy, jak ma przebiegać zmiana wartości zmiennej, zazwyczaj jest to inkrementacja lub dekrementacja.

Pętla będzie wykonywać sie tak długo, aż warunek nie zostanie spełniony. Brzmi to dość podobnie do działania pętli While. Różnica jednak polega na tym, ze w While modyfikowaliśmy zmienna warunkowa wewnątrz instrukcji, natomiast w for deklarujemy przebieg w nagłówku pętli. Dodatkowo w nagłówku przypisujemy początkową wartość zmiennej.

Poniższy przykład pokazuje, że pętle for i while są równoważne. Różnią sie jedynie zapisem:

```
<?php

// użycie pętli for

for($i=0;$i<10;$i++)

{

  // instrukcje

}

// ten sam efekt z użyciem funkcji while

$i=0

while($i < 10)

{

  // instrukcje

  $i++;

}


?>
```

Operator ?

Na pierwszych lekcjach w tym dziale omówiliśmy zasadę działania instrukcji warunkowej if oraz jej rozszerzenie - else. Pokażę teraz, jak zastosować podobną konstrukcję z użyciem operatora " ? ". Jest ona bardzo przydatna przy budowaniu krótkich instrukcji, mając proste warunki.

Zobaczmy, jak wygląda jej struktura:

```
<?php
$a = 5; // przypisujemy wartość zmiennej $a
$odpowiedz = ($a>5) ? 'Większa od 5' : 'Mniejsza, bądź równa 5';
echo $odpowiedz;
?>
```

Czas na krótkie wyjaśnienie. $odpowiedz jest zmienną, do której przypiszemy wynik zwracany przez operator ?. Wyrażenie w nawiasie oznacza nasz warunek (w tym przypadku sprawdzamy, czy $a jest większa od 5). Jeżeli jest, $odpowiedz przyjmuje wartość pierwszą, czyli 'Większa od 5'. Na ekranie wyświetli się komunikat 'Większa od 5'. Jeśli natomiast $a nie będzie większa od 5, $odpowiedz przyjmie wartość drugą (po dwukropku), wyświetlając 'Mniejsza, bądź równa 5'.

Przypisywanie wyniku do zmiennej pomocniczej nie jest konieczne. Spójrzmy na poniższy przykład:

```
<?php
$a = 5; // przypisujemy wartość zmiennej $a
echo ($a>5) ? 'Większa od 5' : 'Mniejsza, bądź równa 5';
?>
```

Bazy danych.

[PHP/MySQL] Jak połączyć się z serwerem i bazą danych MySQL?

```
<?php

// nawiazujemy polaczenie

$connection = @mysql_connect('localhost', 'uzytkownik', 'haslo')

// w przypadku niepowodznie wyświetlamy komunikat

or die('Brak połączenia z serwerem MySQL.<br />Błąd:
'.mysql_error());

// połączenie nawiązane ;-)

echo "Udało się połączyć z serwerem!<br />";

// nawiązujemy połączenie z bazą danych

$db = @mysql_select_db('nazwa_bazy', $connection)

// w przypadku niepowodzenia wyświetlamy komunikat

or die('Nie mogę połączyć się z bazą danych<br />Błąd:
'.mysql_error());

// połączenie nawiązane ;-)

echo "Udało się połączyć z bazą dancych!";

// zamykamy połączenie

mysql_close($connection);

?>
```

Zamknięcie połączenia za pomocą funkcji mysql_close() nie jest konieczne, ponieważ połączenie i tak zostanie zamknięte z chwilą zakończenia działania skryptu.

Poniżej przedstawię Wam funkcję, której ja używam w codziennej pracy z bazą danych MySQL.

```
<?php
/*******************************************************
* connection.php
* konfiguracja połączenia z bazą danych
*******************************************************/

function connection() {
    // serwer
    $mysql_server = "localhost";
    // admin
    $mysql_admin = "uzytkownik";
    // hasło
    $mysql_pass = "haslo";
    // nazwa baza
    $mysql_db = "baza_danych";
    // nawiązujemy połączenie z serwerem MySQL
    @mysql_connect($mysql_server, $mysql_admin, $mysql_pass)
    or die('Brak połączenia z serwerem MySQL.');
    // łączymy się z bazą danych
    @mysql_select_db($mysql_db)
    or die('Błąd wyboru bazy danych.');
}

?>
```

Potem wystarczy taki plik podłączyć za pomocą instrukcji require() i wywołać funkcję connection(), aby mieć aktywne połączenie do serwera i bazy danych.

```
<?php

// podłączamy plik  connection.php

require "connection.php";

// wywołujemy funkcję connection()

connection();


?>
```

Na podstawie:

http://kursphp.com/