Kurs języka PHP dla początkujących

Dawid Birówka

2011

Spis treści

Rozdział 1 – Jak wypisać tekst na ekranie? 4
Od czego zacząć? 4
Instalacja serwera 4
Znajomość języka HTML 4
Edytor tekstu 4
Pierwszy program w PHP 4
Rozdział 2 – Czym są zmienne i kiedy się ich używa? 5
Czym jest zmienna? 5
Po co używać zmienne? 5
Rozdział 3 – Parametry 6
Jak pobrać parametr z tablicy GET? 6
Jak możemy wykorzystać go w PHP? 6
Jak pobrać dane przesłane metodą POST? 7
Rozdział 4 – Napiszmy coś przydatnego! 8
Piszemy własny kalkulator 8
Rozdział 5 – Instrukcje warunkowe 8
Czym one są? 8
Do czego są przydatne? 9
Przykładowa instrukcja warunkowa 9
Jak to działa? 9
Alternatywa do *if* oraz *else* – instrukcja *switch* 10
Do czego to służy? 10
Objaśnienie działania 10
Rozdział 6 – Pętle 10
Kiedy się ich używa? 10
Rodzaje pętli 11
Pętla *while* – najprostszy przykład pętli 11
Pętla *for* 11
Petla *do..while* 11
Rozdział 7 – Funkcje 12
Co to jest funkcja? 12
Funkcja korzystająca z parametrów 12
Po co je używać? 12

Rozdział 8 – Operacje na plikach tekstowych 13
Odczytywanie danych z pliku 13
Zapisywanie danych do pliku 14
Dopisanie danych do pliku 14
Podsumowanie 14

Rozdział 1 – Jak wypisać tekst na ekranie?

Od czego zacząć?

Instalacja serwera

Aby rozpocząć jakiekolwiek działania w PHP potrzebujemy serwera obsługującego język PHP. Najpopularniejszym z nich jest Apache. Osobiście (preferując głównie programy Open Source) używam pakietu WampServer[1]. Jest to zestaw trzech modułów – Apache, PHPMyAdmin oraz system zarządzania bazami danych MySQL. Aktualnie najnowszą wersją jest 2.1.

Znajomość języka HTML

Chcąc „brać się" za PHP, musimy znać język HTML bądź jakikolwiek język programowania podobny w składni do PHP. Jest on podstawą i bez jego znajomości nauczenie się tego języka jest co najmniej ciężkie. Jeśli go jeszcze nie znasz, najpierw się go naucz a dopiero potem szukaj czegoś trudniejszego.

Edytor tekstu

Do pisania w języku PHP można używać dowolnego edytora tekstu. Może to być nawet windowsowy notatnik. Jednak istnieją edytory, które pomagają „połapać się" w składni programu i czynią go bardziej przejrzystym dzięki nadawaniu różnych kolorów różnym funkcjom języka. Ja (stawiając na programy za friko) korzystam z edytora Notepad++[2]. Jest on bardzo przydatny nie tylko do pisania w PHP. Oferuje różne opcje dla wielu języków m.in. dla HTML, JavaScript czy też C++, Pascal i Perl.

Jeżeli wszystko już omówiliśmy, zaczynajmy!

Pierwszy program w PHP

Prawie każdy programista zaczynając naukę w nowym języku, pisze pierwszy program zwany „Hello World!". Wyświetla on na ekranie ten właśnie napis. My nie bądźmy inni. Zróbmy dokładnie to samo.

```
<?php
 echo('Hello world!');
?>
```

Zapiszmy to jako plik index.php. Aby móc go uruchomić musimy usadowić go w folderze serwera. W przypadku WampServera, na dysku C:\ tworzony jest katalog o nazwie wamp. Wchodzimy w katalog www, tworzymy tam własny folder (ja np. nazwałem go Dawid) i umieszczamy w nim swój plik. Aby ucieszyć się widokiem naszego pierwszego programu, w przeglądarce wchodzimy na adres:

http://localhost/(nazwa twojego folderu)/index.php[3]

Jak możemy zauważyć, każdy program w PHP zaczynamy od znaczników początku i końca:

```
<?php
?>
```

To właśnie w nich wpisujemy zawartość programu.

[1] Pobrać go można stąd: http://www.wampserver.com/en/download.php.
[2] Również do pobrania stąd: http://notepad-plus-plus.org/download.
[3] Jeśli ten adres nie działa, oznacza to, że masz inaczej skonfigurowany lub wyłączony pakiet WampServer.

W parametrze funkcji *echo* możemy bez obaw wpisywać znaczniki formatowania kodu HTML. Jak już mówiłem, język HTML współdziała z PHP i jest potrzebny do opanowania wielu funkcji tego języka.

Jeśli dobrze się przyjrzymy, możemy zauważyć, że część kodu

```
echo('Hello world!');
```

została zakończona średnikiem (;). Jest to bardzo ważne, ponieważ bez niego program nie wykona się i najprawdopodobniej wyświetli błąd na ekranie.

Dobrze – potrafimy już wypisać tekst na monitorze. Przejdźmy teraz do omówienia zmiennych.

Rozdział 2 – Czym są zmienne i kiedy się ich używa?

Czym jest zmienna?

Jak podaje Wikipedia[4], zmienna to „konstrukcja programistyczna posiadająca trzy podstawowe atrybuty: symboliczną nazwę, miejsce przechowywania i wartość; pozwalająca w programie odwoływać się przy pomocy nazwy do wartości lub miejsca przechowywania".

Dla nas najważniejszy jest fakt, że zmienna posiada nazwę i wartość. Wartość tą można zawsze zmieniać, stąd nazwa zmienna. Przeciwieństwem zmiennej jest stała, która posiada stale zdefiniowaną wartość i nie można jej przestawić.

Do zmiennej można przypisywać dowolne wartości – dwie najważniejsze z nich to:

- integer (wartość liczbowa)
- string (wartość tekstowa)

W języku PHP, zmienne zaczynają się od znaku $ (dolara). Nazwa zmiennej może być dowolna z wyjątkiem ciągów ze spacjami, nazw zarezerwowanych dla języka oraz ciągów samych cyfr.

Po co używać zmienne?

Zobaczmy to na przykładzie. Utworzymy zmienną imie:

```
<?php
 $imie = 'Dawid';
?>
```

Zapiszmy i odświeżmy stronę. Co się stało? Nic. Zmienna jest zdefiniowana, ale nie jest do niczego użyta. Chcemy więc wypisać ją na ekranie:

```
<?php
 $imie = 'Dawid';
 echo("Twoje imie to: $imie");
?>
```

Właśnie zobaczyliśmy zmienną wypisaną na ekranie.

Jedna, ważna uwaga – PHP rozróżnia wielkość liter (np. *imie* to nie to samo co *IMIE*). Dobrze jest mieć to na względzie, kiedy coś w nim piszemy.

[4] Źródło: pl.wikipedia.org/wiki/Zmienna_(informatyka).

I teraz nasuwa się pytanie: Po co używać zmiennych, jeżeli możemy wpisać to ręcznie? Wyobraźmy sobie, że do zmiennej zapisaliśmy jakiś długi numer, np. numer konta bankowego. Mamy wyświetlić go w kilku miejscach strony. Czy będziemy kopiować go w każde pole gdzie ma się znaleźć? Po pierwsze – będzie to niewygodne (bo po co wklejać tak długi numer zamiast wpisać krótką nazwę zmiennej) a po drugie – zwiększy rozmiar pliku ze stroną. Sensowniejsze więc będzie wstawienie tam zmiennych.

Ale zmienne mają jeszcze jedną pożyteczna opcję – mogą zawierać w sobie operacje matematyczne.

```
<?php
 $dzialanie = 120/3;
 echo("Wynik tego dzialania to: $dzialanie");
?>
```

I co widzimy? *Wynik tego dzialania to: 40*

Rozdział 3 – Parametry

Jak pobrać parametr z tablicy GET?

Jak powinniśmy wiedzieć, wpisując dane do formularza w języku HTML, parametr (o nazwie nadanej w polu name, załóżmy, że jest to parametr *imie*), jest dopisywany do adresu w formie:

```
?imie=To_co_wpisaliśmy
```

Jak możemy wykorzystać go w PHP?

W języku PHP występują tablice globalne i hiperglobalne. Jedna z nich nazywa się *GET*. To właśnie z niej pobiera się dane w formie *?imie=Dawid*. Zobaczmy jak to wygląda na przykładzie:

```
<html><body>
 <form>
  Podaj imie: <input type='text' name='imie'>
  <input type='submit' value='wyslij'>
 </form>
 <br>
  <?php
   @$imie = $_GET['imie'];
   echo("Twoje imie to: $imie");
  ?>
</body></html>[5]
```

Najbardziej powinna nas interesować część kodu:

```
$imie = $_GET['imie'];
```

Widzimy tu zastosowanie tablicy GET. Początek *$_* już pokazuje nam, że mamy do czynienia z tablicą. Przyda się to wam w przyszłości. Ale przejdźmy do omówienia powyższego kodu.

Dla większości z was pierwsza część kodu (tj. formularz), nie powinna być nowością. Zauważmy, że wartość z formularza nosi nazwę imie.

```
Podaj imie: <input type='text' name='imie'>
```

[5] Znak małpy został tu wstawiony tylko po to, aby nie został wyświetlony błąd o niezdefiniowanej zmiennej.

Tak więc do zmiennej PHP *$imie*, przypisaliśmy wartość tablicy *$_GET* o nazwie imie.

Tak właśnie wyciągamy dane z formularza przesłane metodą GET (czyli poprzez dopisanie do adresu). Z tablicy $_GET wyciągamy dane o nazwie z formularza podanej w kwadratowych nawiasach i apostrofach. Możemy oczywiście przypisać to wszystko do jednej zmiennej. Będzie to wtedy dużo czytelniejsze i łatwiejsze.[6]

Jak pobrać dane przesłane metodą POST?

Jak chwilę się zastanowimy, działanie metody POST jest podobne do tablicy GET, z tym jednak wyjątkiem, że dane nie są dopisywane do adresu, lecz są niewidocznie dla użytkownika przesyłane do dowolnego docelowego pliku. Aby użyć tablicy POST, musimy w formularzu zdefiniować coś takiego:

```
<form method='post'>
  Podaj imie: <input type='text' name='imie'>
  <input type='submit' value='wyslij'>
</form>
```

Dane przesłane metodą POST wyciągamy identycznie jak z tabeli GET. Jak? Zamiast $_GET piszemy $_POST. Oto przykład (nie różni się on bardzo od poprzedniego):

```
<html><body>
 <form method='post'>
  Podaj imie: <input type='text' name='imie'>
  <input type='submit' value='wyslij'>
 </form>
 <br>
  <?php
   @$imie = $_POST['imie'];
   echo("Twoje imie to: $imie");
  ?>
</body></html>
```

Jest to bardziej przydatne gdy chcemy przesłać jakieś bardziej "tajne" dane, które nie powinny być widoczne na ekranie. Zarówno za pomocą GET czy też POST, dane można wysłać do innego pliku.

Aby to zrobić, dodajemy do formularza jeszcze jeden parametr:

```
<form method='post' action='jakis_plik.php'>
```

Możemy i raczej powinniśmy w takim wypadku usunąć znacznik
 i część z kodem PHP.

Dane zostaną przesłane do pliku "jakis_plik.php" i zostaniemy do niego automatycznie przekierowani.

Jednak jeśli plik "*jakis_plik.php*" będzie pusty, nic nie zobaczymy… Musimy więc wyświetlić w nim parametr podany z formularza we wcześniejszym pliku. Wpiszmy więc w nim:

```
<?php
 @$imie = $_POST['imie'];
 echo("Twoje imie to: $imie");
?>
```

[6] Przypisywanie zmiennych do innych zmiennych upraszcza zrozumienie kodu - takie operacje są oczywiście dopuszczalne, lecz duża liczba takich działań, zwiększa użycie zasobów komputera i jest odradzana.

Zobaczmy więc końcowy efekt. Uruchommy plik index.php (z formularzem), coś do niego wpiszmy i wyślijmy. Co się stało? Zostaliśmy przekierowani do pliku *jakis_plik.php* i została nam zwrócona wartość, którą wpisaliśmy.

Poznaliśmy teraz sposoby niejakiego "kontaktowania się" z programem. My podajemy dane a komputer coś z nimi robi. Spróbujmy wykorzystać teraz tą wiedzę w praktyce.

Rozdział 4 – Napiszmy coś przydatnego!

Co możemy zrobić dysponując wiedzą o zmiennych? Napiszmy prosty kalkulator. Moglibyśmy posłużyć się metodą GET ale nie zrobimy tego. Użyjemy trochę dłuższej ale "ładniejszej" opcji z metodą POST i formularzem. Użyjemy też dodatkowo parametru *action* dla metody POST. Może nie będzie to zbyt szczupłe lecz chcemy sprawdzić tylko nasze umiejętności.

Piszemy własny kalkulator

Zaczynamy więc od formularza – utwórzmy plik index.php o treści:

```
<html><body>
 <form method='post' action='wyniki.php'>
  Podaj pierwsza liczbe: <input type='text' name='liczba1'><br>
  Podaj druga liczbe: <input type='text' name='liczba2'>
  <input type='submit' value='Licz!'>
 </form>
</body></html>
```

Jak widzimy, dane będą przesłane do pliku *wyniki.php*. Tworzymy go więc i wpisujemy:

```
<?php
 $liczba1 = $_POST['liczba1'];
 $liczba2 = $_POST['liczba2'];

 $wynik1 = $liczba1 + $liczba2;
 $wynik2 = $liczba1 - $liczba2;
 $wynik3 = $liczba1 * $liczba2;
 $wynik4 = $liczba1 / $liczba2;

 echo("Suma wynosi: $wynik1<br>");
 echo("Roznica wynosi: $wynik2<br>");
 echo("Iloczyn wynosi: $wynik3<br>");
 echo("Iloraz wynosi: $wynik4");
?>
```

Działa! Wpisujemy dwie liczby, klikamy i mamy podane wyniki. Proste – myślę, że wszystko już rozumiecie. Przejdźmy teraz do poznania instrukcji warunkowych.

Rozdział 5 – Instrukcje warunkowe

Czym one są?

W języku PHP, można przeprowadzić sprawdzenie jakiegoś warunku. Może to być warunek, że wprowadzone dane są ciągiem liczb, cyfr, że dana jest zmienną, że zmienna jest zdefiniowana, ma jakąś długość itp.

Do czego są przydatne?

Są one bardzo pomocne przy sprawdzaniu danych podanych przez użytkownika np. w trakcie rejestracji w serwisie. Chronią one w pewnym stopniu przez atakiem bota. Sprawdzone może być czy nick ma długość co najmniej x znaków czy też jest nie dłuższy niż y znaków. Można sprawdzić bądź porównać w ten sposób dane wysyłkowe (np. poprawny format kodu pocztowego), narzucić stosowanie liczb bądź znaków interpunkcyjnych w hasłach aby zwiększyć ich bezpieczeństwo itp.

Przykładowa instrukcja warunkowa

Zobaczmy na przykładzie, jak wygląda i jaką ma składnię zapytanie warunkowe[7]:

```
<?php
 @$nick = $_GET['nick'];
 if ($nick == 'Dawid') {
  echo('Witaj Adminie!');
 } else {
  echo('Kim jestes?');
 }
?>
```

Zapiszmy i uruchommy. Co zobaczyliśmy? Komunikat *"Kim jestes?"*.

Dopiszmy teraz ręcznie parametr metody GET do końca adresu: *?nick=Dawid*. Co widzimy? Witaj Adminie! Przeanalizujmy teraz ten program.

Jak to działa?

Warunek sprawdzający zaczynamy krótką instrukcją *if*. Zaraz za nim podajemy w okrągłych nawiasach warunek do spełnienia. Później otwieramy nawias klamrowy "*{*" i w nim podajemy co ma się wykonać jeśli warunek będzie spełniony. Kończymy oczywiście zamykającym nawiasem "*}*".

Gdy warunek nie zostanie spełniony, nie zobaczymy kompletnie nic. Aby jakoś to wyglądało, zdefiniujmy działanie, które ma zostać podjęte gdy warunek nie zostanie spełniony.

Służy do tego instrukcja *else*. Składniowo niczym nie różni się od *if*-a. Działanie do wykonania również podajemy w nawiasach. Oczywiście można łączyć wiele warunków jednocześnie. Utwórzmy taki zapis (pokazuję tylko środkową część):

```
if ($nick == 'Dawid'){
 echo('Witaj Adminie!');
} elseif ($nick == 'Jacek') {
 echo('Witaj uzytkowniku!');
} else {
 echo('Kim jestes?');
}
```

Załóżmy, że Dawid jest administratorem serwisu a Jacek jedynym użytkownikiem.
Przypiszmy do zmiennej *nick* wartość Dawid. Widzimy – *Witaj Adminie!*
Chwilowo przypiszmy jej wartość Jacek. Widzimy – *Witaj użytkowniku!*
Wpiszmy już dowolny ciąg znaków. Widzimy – *Kim jestes?*

[7] Zarówno w tym jak i innych kodach tu podanych, zamiast przypisywania zmiennej do innej zmiennej, można posłużyć się zwrotem bezpośrednim – np. zamiast używania zmiennej *$nick* (czyli zapisanego w krótszej formie *$_GET['nick']*), można użyć samego zwrotu *$_GET['nick']*. Jest to szczególnie wskazane, gdy chcemy zdefiniować wiele zmiennych (patrz -> *przypis* [6]).

Tak więc mamy już objaśnione działanie instrukcji *else* oraz *if*. Jak możemy zauważyć, istnieje również instrukcja *elseif*. Używa się jej w przypadku kilku warunków. W takich przypadkach można zamiast niej użyć oddzielnych instrukcji *else if*, nikt tego nie broni. Jest to jednak trochę krótsze.

Alternatywa do *if* oraz *else* – instrukcja *switch*

Do czego to służy?

Zamiast pisania instrukcji warunkowej opartej na *if* oraz *else*, możemy posłużyć się instrukcją *switch*.

Napiszmy odpowiednik powyższej instrukcji, posługując się tą funkcją:

```php
<?php
 @$nick = $_GET['nick'];
 switch($nick) {
  case 'Dawid':
  echo('Witaj Adminie!');
  break;
  case 'Jacek':
  echo('Witaj uzytkowniku!');
  break;
  default:
  echo('Kim jestes?');
  break;
 }
?>
```

Objaśnienie działania

Instrukcja *switch* jest bardzo podobna do wariantu z *if* oraz *else*. Wyjaśnijmy krok po kroku działanie powyższego kodu:

Instrukcję zaczynamy od polecenia *switch* – przyjmuje ona za parametr zmienną, którą będzie sprawdzać – u nas będzie to *$nick*.

W przypadku (*case*), gdy wartość zmiennej wynosi '*Dawid*' wykonywana jest część kodu pod spodem (aż do znacznika *break*, który oznacza koniec działania). Jeżeli warunek nie zostanie spełniony, sprawdzanie przechodzi dalej. Jeśli *$nick* ma wartość '*Jacek*', wykonuje się część kodu pod nim.

Możemy wypisywać tak sobie do woli. Przydałoby się jednak określić działanie, które będzie podejmowane, kiedy żaden z warunków nie zostanie spełniony. Służy to tego polecenie *default* – jak sama nazwa wskazuje, oznacza ono domyślną akcję, gdy żaden warunek się nie spełni. W naszym przypadku, wyświetlimy na ekranie tekst „*Kim jestes?*”.

Myślę, że rozumiecie działanie instrukcji warunkowych. Przejdźmy do pętli.

Rozdział 6 – Pętle

Kiedy się ich używa?

Pętla – nazwa mówi chyba sama za siebie. Pętla to nic innego, niż struktura pozwalająca na kilkukrotne powtórzenie części kodu. Nie trzeba chyba niczego wyjaśniać.

Wyobraźmy sobie, że chcemy kilka razy wypisać na ekranie jakiś tekst – niech to będzie „Cześć!”.

Czy będziemy wypisywać kilka razy pod rząd funkcję *echo*? Będzie to niewygodne a zarazem nieekonomiczne. Plik przybierze trochę na wadze. Właśnie w takich sytuacjach używamy pętli.

Rodzaje pętli

Pętla *while* – najprostszy przykład pętli

Pętla *while* przyjmuje tylko jeden warunek – do kiedy ma być ona wykonywana. Potrzebujemy do tego zmiennej o jakiejś wartości liczbowej – niech nazywa się *$liczba* i posiada przypisaną wartość 1. Chcemy wypisać *„Cześć!”* załóżmy 10 razy:

```
<?php
 $liczba = 0;
 while ($liczba<10) {
  echo ('Czesc!<br>');
 ++$liczba;
 }
?>
```

Działa. Teraz objaśnienie. Jak już wspomniałem, pętla *while* potrzebuje zmiennej o jakiejś wartości liczbowej. Widzimy to i myślę, że rozumiemy. Na czym polega reszta?

Na początku *$liczba* wynosi 1. Pętla będzie wykonywana dopóki *$liczba* nie będzie wynosiła 10. Za każdy powtórzeniem pętli wypisany będzie tekst *„Cześć!”*, a do zmiennej *liczba* będzie dodawana 1. Tak więc po jednym powtórzeniu pętli *liczba* wynosić będzie 1 (warunek nie zostanie spełniony i pętla wykona się znowu), potem 2,3,4,5,6,7,8, aż dojdzie do 9. Wtedy pętla się zakończy.

Myślę, że to nie było zbyt trudne. Omówmy teraz drugi rodzaj pętli.

Pętla *for*

Działanie pętli *for* jest bardzo podobne do działania pętli *while*. Różnica polega na tym, że zamiast ręcznie definiować zmienną, przypisywać jej wartość a w zawartości pętli jej wartość zmieniać, wszystko to podaje się jako parametry funkcji. Mamy dużo mniej do roboty. Oto przykład (zrobi to samo co poprzedni przykład – wyświetli nam 10 razy *„Cześć!”):*

```
<?php
 for ($a=0; $a<10; ++$a) {
  echo ('Czesc!<br>');
 }
?>
```

- Pierwszy parametr określa początkową wartość zmiennej *$a*
- Drugi jest warunkiem do kiedy ma wykonywać się pętla
- Trzeci parametr określa działanie, które ma się wykonać z każdym powtórzeniem pętli

Petla *do..while*

Pętla *do..while* jest podobna do pętli *while*. Pokaże tylko jej strukturę.

```
<?php
 $a = 0;
 do {
  echo('Czesc!<br>');
  ++$a;
 } while($a<5);
?>
```

Różni się ona od pętli *while* tym, że sprawdzanie warunku odbywa się na końcu – oznacza to, że nawet jeżeli warunek nie będzie spełniony, pętla wykona się co najmniej raz.

Rozdział 7 – Funkcje

Co to jest funkcja?

Funkcja jest w pewnym sensie podobna do działania zmiennych, lecz zamiast przypisanej jej wartości zawiera w sobie program (którego nie można przypisać do zmiennej).

Zdefiniujmy i uruchommy funkcję, która ma zwrócić nam na ekranie znany nam już tekst *„Cześć!”.*

```
<?php
 function wypisz(){
  echo('Czesc!<br>');
 }
 wypisz();
?>
```

Funkcje najpierw definiujemy. Czynimy to używając polecenia *function*. Zaraz po nim stoi jej nazwa (w tym przypadku *wypisz*). W nawiasie (okrągłym) możemy podać zmienne, które będą brane za parametry. Omówimy to za chwilę. Działania do wykonania podajemy w nawiasach klamrowych.

Wywoływanie funkcji jest bardzo proste – po prostu wpisujemy w kodzie jej nazwę z ewentualnymi parametrami.

Funkcja korzystająca z parametrów

Napiszmy teraz funkcję określającą wielkość monitora, która będzie korzystała z jakiegoś parametru.

```
<?php
 function monitor($przekatna){
  if ($przekatna<17){
   echo('Maly<br>');
  } elseif ($przekatna<=20) {
   echo('Spory<br>');
  } else {
   echo('Duzy<br>');
  }
 }
 monitor(14);
 monitor(23);
 monitor(20);
?>
```

Jak widać, funkcja ta korzysta z parametru *$przekatna* oznaczającą przekątną monitora. Jeśli jest mniejsza niż 17 zwraca nam *„Maly”*, jeśli mniejszy lub równy 20 zwraca *„Spory”*, a jeśli większy niż 20 zwraca *„Duzy”*.

Wywołaliśmy funkcję monitor z parametrami 14, 23 oraz 20. Funkcja zwróciła nam odpowiednio: Mały, Duży, Spory. Działa jak należy.

Po co je używać?

Sens ich używania jest identyczny jak sens używania zmiennych – bardzo skracają one pracę. Kiedy mamy kilkukrotnie wykonać tą samą instrukcje warunkową, używamy funkcji i podajemy parametr.

Zbliżamy się już do końca kursu – pozostał nam jeszcze jeden rozdział, mianowicie – zapisywanie i odczytywanie danych do pliku tekstowego.

Rozdział 8 – Operacje na plikach tekstowych

Odczytywanie danych z pliku

Z pozoru prosta sprawa (bo co to jest dla nas otwarcie pliku), w PHP nie wydaje się na pierwszy rzut oka taka łatwa. Stwórzmy w folderze strony plik o nazwie *plik.txt*. Napiszmy w nim: „Zawartosc pliku tekstowego". Zamierzamy go teraz otworzyć i wyświetlić. Oto jak to zrobić:

```
<?php
 $plik = 'plik.txt';
 $uchwyt = fopen($plik, 'r');
 $zawartosc = fread($uchwyt, filesize($plik));
 fclose($uchwyt);
 echo($zawartosc);
?>
```

1. Na początku, lokalizację i nazwę pliku (w naszym przypadku *plik.txt*) przypisujemy do jakiejś zmiennej. Najlepiej aby sugerowała nazwą zawartość. My przypisaliśmy ją do zmiennej *$plik*.
2. Następnym krokiem jest otwarcie pliku. Pliki otwiera się funkcją *fopen*[8]. Funkcja ta przyjmuje dwa parametry[9]. Pierwszym z nich jest lokalizacja pliku i nazwa (u nas wystarczy zmienna *$plik*). Drugi parametr określa tryb otwarcia pliku. Wyróżniamy ich kilka – oto najważniejsze:

Tryb	Znaczenie
r	Tryb odczytu
r+	Tryb odczytu i zapisu
w	Tryb zapisu
w+	Tryb zapisu i odczytu
a	Tryb dodawania
a+	Tryb dodawania i odczytu

   My użyliśmy akurat trybu *r*, ponieważ chcemy tylko odczytać plik. Zapisaliśmy otwarty już plik (nazywany *uchwytem*) jako zmienną *$uchwyt*
3. Odczytujemy zawartość pliku (jak się pewnie domyślacie) funkcją *fread*. Ona również przyjmuje 2 parametry. Mianowicie uchwyt do pliku oraz liczbę bajtów, które mają zostać odczytane. Gdy chcemy zobaczyć cały plik, najłatwiej i zarazem najlepiej jest użyć funkcji *filesize*[10]. Przyjmuje ona za parametr lokalizacje pliku, czyli zmienną *$plik*. Zapiszmy zawartość tego pliku do zmiennej o nazwie *$zawartosc.*
4. Otwarty już plik powinniśmy zamknąć. Robimy to funkcją *fclose*, a za parametr podajemy otwarty wcześniej plik czyli *$uchwyt.*
5. Teraz pozostaje nam wyświetlić dane z pliku. Wyświetlamy zmienną *$zawartosc* za pomocą funkcji *echo*.

Jest to dość zawiły proces (nawet ja potrafię czasem się w nim pogubić), więc trudności z jego zrozumieniem są naturalne.

[8] Dla osób znających j. angielski jest to zapewne bardzo proste.
[9] Istnieje również trzeci (wyszukiwanie pliku w *include_path*) oraz czwarty (kontekst), są one jednak trudniejsze do zrozumienia i nie są nam potrzebne.
[10] Funkcja *filesize* sprawdza i zwraca nam rozmiar pliku w bajtach – znacznie ułatwia nam ona życie.

Zapisywanie danych do pliku

Zapisywanie danych do pliku jest niemal identyczne z odczytywaniem danych z pliku. Zmienia się tylko parametr, z którym otwieramy plik (zamiast *r* wpisujemy *w*), oraz funkcja, którą użyjemy.

Dane zapisuje się za pomocą funkcji *fputs*. Jako parametry podajemy odpowiednio lokalizację (*$plik*) oraz dane do zapisania. Zdefiniujmy zmienną *$daneZapis* i przypiszmy do niej tekst: *„Te dane zostaly zapisane za pomoca funkcji puts”*. Będzie wyglądać to tak:

```php
<?php
 $plik = 'plik.txt';
 $daneZapis = 'Te dane zostaly zapisane za pomoca funkcji puts';
 $uchwyt = fopen($plik, 'w');
 fputs($uchwyt, $daneZapis);
 fclose($uchwyt);
?>
```

Zobaczmy teraz jak wygląda *plik.txt*. Widzimy: *„Te dane zostaly zapisane za pomoca funkcji puts”.*

Jak widzimy zapisywanie danych do pliku zadziałało.

UWAGA! Użycie parametru '*w*' przy otwieraniu pliku powoduje usunięcie danych znajdujących się w nim do tej pory. Aby dopisać coś do pliku należy użyć parametru '*a*'.

Dopisanie danych do pliku

Dopisywanie danych do pliku działa identycznie jak zapisywanie, z tym jednak wyjątkiem, że przy otwieraniu pliku używamy parametru '*a*' zamiast '*w*'. Dane zostaną dopisane na końcu danego pliku.

Podsumowanie

Tak więc ukończyliśmy (a w zasadzie ukończyłeś) kurs podstaw języka PHP. Jeżeli wciągnął Cię ten temat możesz kontynuować naukę. Jest wiele książek, czasopism na temat programowania obiektowego. Jest ogrom funkcji PHP, o których nie wspomnieliśmy. My poznaliśmy jedynie niezbędne z nich.

Język PHP wyposażony jest w obsługę dokumentów cookie(ciasteczek), sesji itp. Przy połączeniu PHP z systemem baz danych MySQL staje się on potęgą możliwości. Dzięki zainstalowaniu MySQLa możemy z łatwością operować danymi użytkowników, kto wie może i pracowników…

Ten język jest skarbnicą możliwości. Jeżeli dalej chcesz się go uczyć mogę życzyć Ci tylko jednego:

Powodzenia!