Ładowanie zawartości pliku do skryptu php

- Wyrażenie **include(sciezka_do_pliku)** pozwala na załadowanie (wnętrza) pliku do skryptu php. Plik ten może zawierać wszystko, co może się znaleźć w obrębie skryptu.
- Wyrażenia include() i require() są niemal identyczne, ale w przypadku **niepowodzenia** (gdy nie znaleziono podanego pliku) te pierwsze wygeneruje **wyjątek**, podczas gdy **require** wygeneruje **błąd**.
- Pliki są wstawiane na podstawie podanej jako parametr ścieżki, lub jeśli nie została ona określona to pobierana jest ścieżka z parametru include_path serwera. Jeżeli i tak nie zostanie znaleziony szukany plik, to funkcja include(), próbuje go szukać w ścieżce, w której przechowywany jest aktualnie uruchomiony skrypt.
- Ścieżki można podawać w sposób **absolutny** (rozpoczynając ścieżkę od \ lub / w zależności od systemu Windows lub Unix), lub **relatywnie w stosunku do bieżącego katalogu** (stosując znak kropki . lub dwóch kropek ..)

Funkcje include() i require()

- Funkcje include() i require() występują w dwóch odmianach: **include_once()** i **require_once()** – gwarantują one, że plik zostanie dołączony **tylko raz.** Unika się wtedy problemów redefiniowania funkcji, ponownego przypisania wartości zmiennym itp. Jednakże odmiany te działają **wolniej** niż ich pierwowzory.
- PHP **nie zwraca uwagi na rozszerzenie** pliku. Dołączyć można każdy plik. Istnieje konwencja, która mówi, że pliki dołączane można zastąpić rozszerzeniem ***.inc**. Należy jednak uważać:

  Każdy może taki plik sobie odczytać (jeśli nie ustawimy odpowiednio praw dostępu) i zobaczy uwzględniony w nim kod php jako **zwykły tekst**. Jeśli są tam dane logowania do bazy lub inne wrażliwe…

Można więc stosować rozszerzenie ***.php** lub trzymać pliki dołączane w osobnym (dobrze zabezpieczonym) folderze.

Funkcje include() i require() c.d.

Przykład:

<?php require('naglowek.php'); ?>

echo(„Witamy Państwa na naszej stronie");

<?php require('stopka.php'); ?>

- Aby mieć pewność, że plik dołączany zostanie potraktowany jako tekst lub HTML, a nie zostanie wykonany przez PHP można użyć funkcji readfile() – wyświetla ona zawartość pliku bez parsowania go (i zwraca liczbę odczytanych bajtów).

int **readfile** (string $filename[,bool $use_include_path = false [, resource $context]])

Stosując opcje konfiguracyjne pliku php.ini: **auto_prepend_file** i **auto_append_file** możemy zagwarantować sobie, że zawsze pewne pliki np. nagłówka i stopki będą dołączane przed lub po każdej stronie

Można ustawić te opcje również w pliki **.htaccess** jeśli serwer został odpowiednio do tego skonfigurowany.

Wywoływanie funkcji

- PHP nie zwraca uwagi na wielkość liter podczas wywoływania funkcji: Test(), TeST(), test() to poprawne wywołania tej samej funkcji. Dobrą konwencją jest jednak stosowanie np. małych liter lub camelCase.
- Deklaracja własnej funkcji rozpoczyna się od słowa kluczowego **function**, potem następuje nazwa funkcji, ewentualne jej parametry w nawiasach okrągłych i kod samej funkcji w nawiasach klamrowych.

  Przykład:

```
function moja_funkcja() {
        echo(„To jest moja funkcja”);
}
```

Wywołanie moja_funkcja() da oczekiwany efekt.

Funkcje c.d.

Należy trzymać się następujących zasad przy nazywaniu funkcji:

- Funkcja nie może mieć takiej samej nazwy jak już istniejąca.
- Nazwa funkcji może zawierać jedynie litery, cyfry i znak podkreślnika _.
- Nazwa funkcji nie może się rozpoczynać od cyfry.

Wywołanie funkcji **$nazwa() jest prawidłowe**, ale zależy od aktualnej wartości zmiennej nazwa. PHP pobiera wartość przechowywaną w tej zmiennej i funkcję o takiej właśnie nazwie próbuje wywołać

PHP **nie umożliwia przeciążania funkcji**, więc każda funkcja nie może mieć nazwy takiej samej jak inna już zdefiniowana. Proszę jednak pamiętać, że **funkcje** zdefiniowane przez **użytkownika istnieją jedynie w obrębie skryptu**, w którym zostały one zadeklarowane.

Parametry funkcji

```
<?php
function stworz_tabele2($dane , $border=1, $cellpadding=4, $cellspacing=4) {
    echo "<table border = \"".$border ."\ " cellpadding = \"". $cellpadding". "\"
    cellspacing =\" " .$cellspacing. "\">";
    reset( $dane) ;
    $wartosc = current($dane) ;
    while ($wartosc) {
            echo "<tr><td>". $wartosc. "</td></tr>\ n";
            $wartosc = next($dane).
    }
    echo "</table>" ;
}

$tablica = array(„Jeden”, „Dwa”, „Trzy”);
stworz_tabele2($tablica, 3, 8, 8);
?>
```

W przedstawionym przykładzie jeden parametr jest wymagany, pozostałe 3 są opcjonalne bo mają przypisane domyślne wartości.

Zmienna liczba parametrów funkcji

Funkcje mogą przyjmować zmienną liczbę parametrów. Wówczas dostęp do tych parametrów dają nam 3 funkcje:

1. int **func_num_args** (void) – zwraca liczbę parametrów przekanazych do funkcji
2. mixed **func_get_arg** (int $arg_num) – pobiera wartość parametru pod podanym indeksem
3. array **func_get_args** (void) – zwraca tablicę wszystkich parametrów funkcji.

Przykład:

```
function zmienne_parametry() {
    echo "Liczba parametrów :";
    echo func_num_args();
    $argumenty = func_get_args();
    foreach ($argumenty as $argument) {
            echo $argument.„\n";
    }
```

Funkcja wyświetla na wejście wszystkie podane parametry.

Zasięg zmiennych

Zasięg zmiennej kontroluje miejsca, w których zmienna ta jest widzialna i zdatna do użytku. Obowiązują następujące zasady odnośnie zasięgu zmiennych:

- Zmienne zadeklarowane **wewnątrz funkcji** posiadają zasięg od miejsca, w którym zostały zadeklarowane, do końca funkcji. Mówimy o zasięgu *funkcji,* a zmienne są nazywane ***zmiennymi lokalnymi.***
- Zmienne zadeklarowane **na zewnątrz funkcji** mają zasięg od miejsca, w którym zostały wywołane, do końca pliku, **ale nie wewnątrz funkcji.** Mówimy o *zasięgu globalnym,* a zmienne noszą nazwę ***zmiennych globalnych****.*
- Specjalne zmienne **superglobalne** są widoczne zarówno **wewnątrz**, jak i **na zewnątrz funkcji.**
- Stosowanie wyrażeń **require() i include() nie zmienia zasięgu**. Jeżeli wyrażenie to zostało wywołane wewnątrz funkcji, odnosi się do niego zasięg lokalny, jeżeli na zewnątrz funkcji – zasięg globalny.
- Słowo kluczowe **global** może zostać zastosowane do **ręcznego przypisania globalnego zasięgu** zmiennej zdefiniowanej bądź stosowanej wewnątrz funkcji.
- Zmienne mogą być ręcznie usunięte przez wywołanie funkcji **usnet**($nazwa_zmiennej). Jeżeli zmienna została usunięta, **nie posiada** już dłużej **zasięgu.**

Zasięg zmiennych – przykłady

- ```
  function fn () {   //kod ten nie da żadnego efektu
          $zmienna = „test”; //zmienna zadeklarowana w funkcji nie jest widoczna poza nią
  }
  fn ();
  echo $zmienna;
  ```

Można używać tej samej nazwy zmiennej wewnątrz i na zewnątrz funkcji, ale może to być mylące i prowadzić do błędów.

- ```
  function fn() {    //wyświetli się słowo test 2 razy
  global $zmienna;          //zmienna zadeklarowana jako globalna mimo iż jest wewnątrz funkcji
  $zmienna = „test”:
  echo "Wewnątrz funkcji: ".$zmienna. "<br/>";
  }
  fn();
  echo "Na zewnątrz funkcji: ".$zmienna ."<br/>":
  ```

Ważne jest miejsce wywołania funkcji – zmienna ma zasięg globalny od momentu wykonania global $zmienna;

Przekazywanie przez referencje vs. wartość

- Domyślnie parametry funkcji są **przekazywane przez wartość**. Kiedy przekazuje się zmienną jako parametr, tworzona jest jej kopia i wszelkie operacje wykonywane są na kopii.

Przykład:

```
function powieksz($liczba, $dodaj = 1) {
    $liczba = $liczba + $dodaj;
}
$liczba = 10;
powieksz($liczba);
echo $liczba;          //wyświetli 10
```

- Można albo zadeklarować zmienną $liczba jako globalną (nie zalecane bo zawsze musi się ta zmienna tak samo nazywać)
- Lepiej zastosować **przekazywanie przez referencje** – w tym przypadku funkcja nie tworzy kopii zmiennej a otrzymuje jedynie referencje (**odwołanie**) **do oryginału**. Parametr przekazuje się przez referencje umieszczając przed jego nazwą **znak &**.

```
function powieksz(&$licz, $dodaj = 1) {
    $licz = $licz + $dodaj;
}
```

Słowo kluczowe return

- Słowo kluczowe return kończy dalsze wykonywanie funkcji i zwraca sterowanie do fragmentu programu, który daną funkcję wywołał. Czasem wykorzystywane jest do przerwania działania funkcji, gdy wystąpił jakiś błąd:

```
function powieksz($liczba, $dodaj) {
if($dodaj < 0)
    return;
else
    $liczba = $liczba + $dodaj;
}
```

Jest to jednak niezgodne z zasadą programowania strukturalnego „jedno wejście i jedno wyjście z modułu”

- Najczęściej return służy do zwracania wartości przez funkcję:

```
function porownaj($x, $y){
if ($x == $y)
    return true;
else
    return false;
}
```

Jako, że nie podajemy typu zwracanej wartości należy w pewnych przypadkach użyć operatora === aby nie pomylić wartości false z 0!