



Въведение в PHP

Контролни структури и функции

Георги Гроздев
g.grozdev@viscomp.bg

Атанас Василев
a.vasilev@viscomp.bg

www.viscomp.bg

About us

■ Лектори

- Георги Гроздев – g.grozdev@viscomp.bg
- Атанас Василев – a.vasilev@viscomp.bg

■ Инфо и слайдовете ще намерите на:

- <http://phplab.viscomp.bg>

vis|comp
we develop the web

Контролни структури

- Позволяват да се контролира хода на изпълнение на програмата
- Основополагащ елемент в структурата на PHP

Условни структури

Промяна на хода на изпълнение в зависимост от едно или повече условия

■ *if*

- Блокът в инструкцията IF ще се изпълни само ако стойността на тествания израз се оцени на TRUE.
- Ако типа на стойността на израза не е булев, ще се оцени автоматично-трансформираната стойност към булев тип

```
if (expression1) {  
    // ...  
}  
  
$a = 4;  
if ($a) {  
    // true  
}  
if (true === $a) {  
    // false - двата операнда са от различен тип  
}
```

Условни структури

- *if - else*
 - Използва се в случаите, когато искаме да дефинираме алтернативен блок от кода, който да се изпълни, ако условието, тествано в IF има стойност FALSE

```
if (expression1) {  
    // ...  
} else {  
    // ...  
}
```

Условни структури

- *if-elseif-else*

- Допълнителен условен тест в рамките на една IF конструкция.

```
if (expression1) {  
    // ...  
} else if (expression2) {  
    // Интервалът между ELSE и IF не е задължителен  
} else {  
    // ...  
}
```

Условни структури

- IF конструкциите могат да се влагат една в друга:

```
if (expression1) {  
    if (expression2) {  
        // Code  
    } else {  
        // More code  
    }  
} else {  
    if (expression3) {  
        // ...  
    }  
}
```

Условни структури

- алтернативен синтаксис

```
if (expression1):  
    // ...  
else if (expression2):  
    // ...  
else:  
    // ...  
endif;
```

Условни структури

■ *switch*

- Алтернативна конструкция за промяна на хода на изпълнение, в зависимост от стойността на даден израз

```
$a = 0;
if ($a) {
} elseif ($a == 0) {
} else { }
```

- В няколко последователни теста се оценява един и същ израз (променлива).
Проблеми:

- Повече код
- Стойността на \$a се изчислява всеки път, когато попадне в тестван израз

```
$a = 0;
switch ($a) {
  case true: // сравнява с true
    break;
  case 0: // сравнява се с 0
    break;
  default:
    // ако до момента не е изпълнена break инструкцията
    break;
}
```

Циклични структури

- Позволяват при определени условия част от кода да се изпълнява повече от веднъж

Циклични структури

■ *break*

- Преждевременно прекъсване на изпълнението на циклични конструкции (while, do..while, for, foreach), както и switch

```
while (expression1) {  
    // ...  
    break;  
}
```

■ *continue*

- Прекъсва текущата итерация и прехвърля изпълнението в началото на следващата

```
while (expression1) {  
    // ...  
    continue;  
    // ...  
}
```

Циклични структури

- *do ... while()*
 - Блокът с инструкциите се изпълнява преди условния тест
 - Гарантира поне една итерация, дори ако условният тест е отрицателен – за разлика от `while()`, където ако тестваното условие изобщо не се изчисли като `TRUE`, няма да се изпълни нито една итерация от цикъла

```
do { // code to be executed
} while (expression);
```

- Тестът след `while` при тази конструкция задължително завършва със знак за точка и запетая.

Циклични структури

■ *for*

- Специфичен еквивалент на `while()` цикъл. По-четлива нотация – инициализиращите и модифициращите инструкции, както и условният тест се изписват на един ред.

```
for (init_expr_1,2,N; test_expr; mod_expr_1,2,N) {  
    // code to be executed  
}  
  
$total = 0;  
for ($i= 1; $i <= 10; $i++) {  
    $total += $i;  
}
```

Циклични структури

■ *for* [2]

- И трите вида инструкции може да са повече от една – в този случай се разделят със запетая.
- Ако в условията тест има повече от една инструкция, се изпълняват всички, но се оценява само последната

```
$total = 0;
for ($i = 0, $j = 1; $j++ <= 10; $i++, $j *= 2) {
    $total += $j;
}
```

- Всяка от 3-те инструкции може да бъде пропусната, но знаците за точка и запетая – не!

```
for (;;) {
    // безкраен цикъл
    // трябва да е предвидено прекъсване (break, return, exit, die)
}
```

Циклични структури

■ *foreach*

- Служи за обхождане на елементите на масив
- Вътрешният указател на масива автоматично се нулира в при първаначалното изпълнение на цикъла
- `foreach()` работи с копие на масива, така че ако в тялото на цикъла променяме елементите на оригиналния масив, това няма да се отрази на итерациите му

```
foreach (array as $value) {  
    // ...  
}
```

```
foreach (array as $key => $value) {  
    // ...  
}
```

Циклични структури

■ *foreach* [2]

- От PHP 5 насам съществува възможност стойността на текущия елемент да се присвои на `$value` по референция, така че в този случай можем да променяме съдържанието на масива в тялото на цикъла

```
$a = array (1, 2, 3);
```

```
foreach ($a as $k => &$v) {  
    $v += 1;  
}
```

```
unset($v); // премахваме последната референция присвоена на $v
```

```
// $a вече ще съдържа (2, 3, 4)
```

Други контролни структури

■ *exit*

- `exit` спира изпълнението на текущия скрипт, в момента, в който се изпълни
- `exit` може да приеме като параметър стойност:
 - ако стойността е от тип низ, тя ще се отпечата преди изпълнението на скрипта да спре
 - ако стойността е цяло число, ще се използва като `exit` статус: 0 за нормално изпълнение; 1 – 254 са статуси за грешка

```
// нормален изход
```

```
exit;  
exit();  
exit(0);
```

```
// изход с код за грешка
```

```
exit(1);  
exit(2);  
exit() е алиас на die()
```

Други контролни структури

■ *return*

- Ако е извикано от глобалния обхват, изпълнението на текущия скрипт се прекратява
- Ако е извикано от включен (с `require` или `include`) скрипт, тогава контролът се връща на извикващия скрипт
 - Ако е извикано от скрипт, включен с `include`, тогава стойността подадена на `return()` се връща като стойност на извикването на `include()`
- Ако е извикано от функция, `return()` прекратява изпълнението ѝ и връща аргумента си като стойност от извикването на функцията.

```
/* included.php */  
return(3)
```

```
/* including file */  
$result = include 'included.php'; // $result == 3
```

Други контролни структури

- *include()*, *require()*

```
<?php
include 'file_a.php';
require 'file_b.php';

// ...
?>
```

- Включват и изпълняват даден файл
- Когато PHP включва даден файл, интерпретаторът излиза от режим PHP, така че ако искаме там да изпълним код, трябва да поставим отварящи и затварящи PHP тагове
- Ако се направи опит да се включи липсващ файл, и двете конструкции ще върнат Warning, но *require()* ще предизвика и фатална грешка.

Други контролни структури

- *include()*, *require()* [2]
 - И двете са езикови конструкции, а не функции, но само *include()* ще върне стойност ако се присвои на променлива:
 - 1 при успешно включване на файла,
 - FALSE при неуспех,
 - стойността на `$a` ако във включения файл е използвана `return($a)`
 - Файловете за включване се търсят първо в `include_path`, после в директорията на текущия скрипт
 - Кодът във включения файл наследява обхвата на действие на променливите на реда, в който става включването
 - Ако включването е станало във функция, то все едно, че кодът от включения файл е бил част от дефиницията на функцията – променливите от него ще наследят локалния обхват на променливите на функцията

Други контролни структури

- *include_once()*, *require_once()*

- Поведението им е идентично с това на *require()* и *include()*, с тази разлика, че ако файлът вече е бил включен, той няма да бъде включен повторно.
- Преди PHP 5, на операционни системи, които не са чувствителни към регистъра е възможно файл да се включи повторно, например:

```
require_once "a.php";  
require_once "A.php"; // Под Windows и PHP4 ще включи отново a.php
```

Потребителски функции

ФУНКЦИИ

- Крайъгълен камък в процедурното и обектно-ориентираното програмиране
- Приемат стойности, изпълняват инструкции и връщат стойност
- Синтаксис

```
function name() {...}
```

- Имената на функциите не са чувствителни към регистъра
- Имената съдържат букви, цифри , под-черта и не могат да започват с цифра
- Във функция можем да разположим всякакъв валиден PHP код, включително дефиниция на функции и класове

ФУНКЦИИ

■ Връщане на стойност

- Всички функции връщат стойност – ако изрично не се връща никаква стойност, функцията ще върне NULL

```
function hello()  
{  
    return "Hello World"; // Не се отпечатва нищо  
}  
$txt = hello(); // Присвоява се "Hello World" на $txt  
echo hello(); // Отпечатва "Hello World"
```

- Можем да прекъснем изпълнението на функция с *return*

```
function hello($who)  
{  
    echo "Hello $who";  
    if ($who == "World") {  
        return; // Изпълнението се прекъсва  
    }  
    echo ", how are you";  
}  
hello("World"); // Отпечатва "Hello World"  
hello("Reader") // Отпечатва "Hello Reader, how are you?"
```

ФУНКЦИИ

■ Връщане на стойност [2]

- По подразбиране когато връщат променлива, функциите връщат нейно копие за всички типове, освен обекти
- За да върнем по референция използваме референтния оператор
- По референция може да се връщат само променливи – не може да се върне израз или празен return с идеята да върнем NULL

```
function &query($sql)
{
    $result = mysql_query($sql);
    return $result;
}
// The following is incorrect and will cause PHP to emit a notice
// when called.
function &getHello()
{
    return "Hello World";
}
```

Функции

- Обхват на видимост на променливите
 - Глобален
 - Локален
 - Класов

- Три начина за достъп до глобални променливи във функция
 - global
 - \$GLOBALS
 - Подаване на аргументи по референция

Функции

■ Предаване на аргументи

- Аргументите позволяват да се вмъкват стойности във функцията, така че да се повлияе на изпълнението ѝ
- Може да се предават повече аргументи, отколкото са посочени в декларацията на функцията
- Не може да се предават по-малко от декларираните, освен ако не са декларирани със стойност по подразбиране.
- Незадължителните аргументи се декларират най-вдясно в списъка с аргументи

ФУНКЦИИ

■ Предаване на аргументи [2]

- Аргументи се предават по референция с помощта на референтния оператор
- Само променливи могат да бъдат предавани по референция
- Преди PHP 5 не можеше да се задават стойности по подразбиране за аргументи, декларирани като предадени по референция

```
<?php
function add_some_extra(&$string)
{
    $string .= 'и още нещо.';
}
$str = 'Това е низ ';
add_some_extra($str);
echo $str;    // извежда 'Това е низ и още нещо.'
?>
```

Въпроси?

**Благодаря ви
за вниманието!**