Керуючі конструкції мови PHP.

1.1. Умовні оператори <вверх>

Оператор if

Це один з найважливіших операторів багатьох мов, включаючи PHP. Він дозволяє виконувати фрагменти коду в залежності від умови. Структуру оператора if можна представити наступним чином:

```
if (вираз) блок_виконання
```

Тут виразом є будь-який правильний PHP-вираз (тобто все, що має значення). У процесі обробки скрипта вираз набуде логічного типу. Якщо в результаті перетворення значення виразу істинне (True), то виконується блок_виконання. В противному випадку блок_виконання ігнорується. Якщо блок_виконання містить кілька команд, то він повинен бути вкладений у фігурні дужки { }.

Правила перетворення виразу до логічного типу:

1. У FALSE перетворяться наступні значення:
   - логічне False
   - цілий нуль (0)
   - дійсний нуль (0.0)
   - порожній рядок і рядок "0"
   - масив без елементів
   - об'єкт без змінних (докладно про об'єкти буде розказано в одній з наступних лекцій)
   - спеціальний тип NULL
2. Всі інші значення перетворяться в TRUE.

```
<?
```

```
$names = array("Іван","Петро","Семен");
if ($names[0]=="Іван") {
    echo "Привіт, Ван!";
    $num = 1;
    $account = 2000;
}
if ($num) echo "Іван перший у списку!";
$bax = 30;
if ($account > 100*$bax+3)
    echo "Цей рядок не з'явиться на екрані, тому що умова не виконана";
?>
```

Ми розглянули тільки одну, основну частину оператора if. Існує кілька розширень цього оператора. Оператор else розширює if на випадок, якщо вираз що перевіряється в if є невірним, і дозволяє виконати які-небудь дії за таких умов.

Структуру оператора if, розширеного за допомогою оператора else, можна представити наступним чином:

```
if (вираз) блок_виконання
else блок_виконання1
```

Цю конструкцію if...else можна інтерпретувати приблизно так: якщо виконано умову (тобто вираз=true), то виконуються дії з блоку_виконання, інакше – дії з блоку_виконання1. Використовувати оператор else не обов'язково.

Оператор elseif

Ще один спосіб розширення умовного оператора if – використання оператора elseif. elseif – це комбінація else і if. Як і else, він розширює if для виконання різних дій у тому випадку, якщо умова, що перевіряється в if, невірна. Але на відміну від else, альтернативні дії будуть виконані, тільки якщо elseif-умова є вірною. Структуру оператора if, розширеного за допомогою операторів else і elseif, можна представити наступним чином:

```
if (вираз) блок_виконання
elseif(вираз1) блок_виконання1
...
else блок_виконання
```

Операторів elseif може бути відразу кілька в одному if-блоці. Elseif-твердження буде виконано, тільки якщо попередня if-умова є False, усі попередні elseif-умови є False, а дана elseif-умова – True.

Альтернативний синтаксис

PHP пропонує альтернативний синтаксис для деяких своїх керуючих структур, а саме для if, while, for, foreach і switch. У кожнім разі відкриваючу дужку потрібно замінити на двокрапку (:), а закриваючу – на endif;, endwhile; і т.д. відповідно.

Наприклад, синтаксис оператора if можна записати наступним чином:

```
if(вираз): блок_виконання endif;
```

Зміст залишається тим же: якщо умова, записана в круглих дужках оператора if, виявилась істиною, буде виконуватися весь код, від двокрапки «:» до команди endif;. Використання такого синтаксису корисно при вбудовуванні php у html-код.

```
<?php
$names = array("Іван","Петро","Семен");
if ($names[0]=="Іван"):
?>
Привіт, Ван!
<?php endif ?>
```

Оператор switch

Ще одна конструкція, що дозволяє перевіряти умову і виконувати в залежності від цього різні дії, – це switch. На українську мову назва даного оператора перекладається як «перемикач». І зміст у нього такий же. У залежності від того, яке значення має змінна, він виконує ту чи іншу дію. switch дуже схожий на оператор if...elseif...else або набір операторів if. Структуру switch можна записати наступним чином:

```
switch (вираз або змінна){
case значення1:
    блок_дій1
break;
case значення2:
    блок_дій2
break;
...
default:
    блок_дій_по_замовчуванні
}
```

На відміну від if, тут значення виразу не зводиться до логічного типу, а просто порівнюється зі значеннями, перерахованими після ключових слів case (значення1, значення 2 і т.д.). Якщо значення виразу співпадає з якимось варіантом, то виконується відповідний блок_дій – від двокрапки після значення, що співпало, до кінця switch або до першого оператора break, якщо такий знайдеться. Якщо значення виразу не співпало з жодним з варіантів, то виконуються дії по замовчуванні (блок_дій_по_замовчуванні), що знаходяться після ключового слова default. Вираз в switch обчислюється тільки один раз, а в операторі elseif – щоразу, тому, якщо вираз достатньо складний, то switch працює швидше.

```
<?
$names = array("Іван","Петро","Семен");
switch ($names[0]){
case "Іван":
    echo "Привіт, Ван!";
break;
case "Петро":
    echo "Привіт, Петя!";
break;
case "Семен":
    echo "Привіт, Сеня!";
break;
default:
    echo "Привіт, $names[0].
   А як Вас кличуть?";
}
?>
```

Якщо в цьому прикладі опустити оператор break, наприклад, у case "Петро":, то, якщо змінна виявиться рівною рядкові "Петро", після виведення на екран повідомлення "Привіт, Петя!" програма піде далі і виведе також повідомлення "Привіт, Сеня!" і тільки потім, зустрівши break, продовжить своє виконання за межами switch.

Для конструкції switch, як і для if, можливий альтернативний синтаксис, де відкриваюча switch фігурна дужка заміняється двокрапкою, а закриваюча – endswitch; відповідно.

7.2. Оператори циклу <вверх>

У PHP існує кілька конструкцій, що дозволяють виконувати повторювані дії в залежності від умови. Це цикли while, do..while, foreach і for. Розглянемо їх більш докладно.

Цикл з передумовою while

Структура:

```
while (вираз) { блок_виконання }
```

або

```
while (вираз): блок_виконання endwhile;
```

while – простий цикл. Він пропонує PHP виконувати команди блоку_виконання доти, поки вираз обчислюється як True (тут, як і в if, відбувається переведення виразу до логічного типу). Значення виразу перевіряється щоразу на початку циклу, так що, навіть якщо його значення змінилося в процесі виконання блоку_виконання, цикл не буде зупинений до кінця ітерації (тобто поки всі команди блоку_виконання не будуть виконані).

```
<?
//ця програма надрукує всі парні цифри від 1 до 10
    $i = 1;
    while ($i < 10) {
        if ($i % 2 == 0) print $i;
    // друкуємо цифру, якщо вона парна
        $i++;
    // і збільшуємо $i на одиницю
    }
?>
```

Цикл з післяумовою do...while

Цикли do..while дуже схожі на цикли while, з тією лише різницею, що істинність виразу перевіряється наприкінці циклу, а не на початку. Завдяки цьому блок_виконання цикл do...while гарантовано виконається хоча б один раз.

Структура:

```
do {блок_виконання} while (вираз);
<?
// ця програма надрукує число 12, незважаючи на те
// що умова циклу не виконується
$i = 12;
    do{
        if ($i % 2 == 0) print $i;
    // якщо число парне, те друкуємо його
```

```
        $i++;
    // збільшуємо число на одиницю
    }while ($i<10)
?>
```

Цикл з лічильником for

Це самі складні цикли в PHP. Вони нагадують відповідні цикли Pascal.

Структура:

```
for (вираз1; вираз2; вираз3) {блок_виконання}
```

або

```
for (вираз1; вираз2; вираз3): блок_виконання endfor;
```

Тут, як ми бачимо, умова складається відразу з трьох виразів. Перший вираз вираз1 обчислюється безумовно один раз на початку циклу. На початку кожної ітерації обчислюється вираз2. Якщо він є True, то цикл продовжується і виконуються всі команди блоку_виконання. Якщо вираз2 обчислюється як False, то виконання циклу зупиняється. Наприкінці кожної ітерації (тобто після виконання всіх команд блоку_виконання) обчислюється вираз3.

Кожен з виразів 1, 2, 3 може бути порожнім. Якщо вираз2 є порожнім, то це означає, що цикл повинен виконуватися невизначену кількість разів (у цьому випадку PHP вважає цей вираз завжди вірним). Це не так непотрібно, як здається, адже цикл можна зупиняти, використовуючи оператор break.

Наприклад, усі парні цифри від 1 до 10 можна вивести з використанням циклу for наступним чином:

```
<?php
for ($i=0; $i<10; $i++){
    if ($i % 2 == 0) print $i;
    // друкуємо парні числа
}
?>
```

Якщо опустити другий вираз (умову $i<10), то таку ж задачу можна розв’язати, зупиняючи цикл оператором break.

```
<?php
for ($i=0; ; $i++){
    if ($i>=10) break;
      // якщо $i більше або дорівнює 10,
      // те припиняємо роботу циклу
    if ($i % 2 == 0) print $i;
     // якщо число парне,
      // те друкуємо його
}
?>
```

Можна опустити всі три вирази. У цьому випадку просто не буде задано початкове значення лічильника $i і воно не буде змінюватися щоразу наприкінці циклу. Усі ці дії можна записати у вигляді окремих команд або в блоці_виконання, або перед циклом:

```
<?php
$i=0; // задаємо початкове значення лічильника
for ( ; ; ){
    if ($i>=10) break;
     // якщо $i більше або дорівнює 10,
      // те припиняємо роботу циклу
    if ($i % 2 == 0) print $i;
     // якщо число парне,
      // те друкуємо його
$i++; // збільшуємо лічильник на одиницю
}
?>
```

У третій вираз конструкції for можна записувати через кому відразу кілька найпростіших команд. Наприклад, якщо ми хочемо просто вивести всі цифри, то програму можна записати зовсім просто:

```
<?php
for ($i=0; $i<10; print $i, $i++)
/* Якщо блок_виконання не містить команд
або містить тільки одну команду,
фігурні дужки, у які він укладений,
можна опускати*/
?>
```

Цикл для роботи з масивами foreach

Ще одна корисна конструкція. Вона з'явилася тільки в PHP4 і призначена винятково для роботи з масивами.

Синтаксис:

```
foreach ($array as $value) {блок_виконання}
```

або

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

У першому випадку формується цикл по всіх елементах масиву, заданого змінною $array. На кожному кроці циклу значення поточного елемента масиву записується в змінну $value, і внутрішній лічильник масиву пересувається на одиницю (так що на наступному кроці буде видно наступний елемент масиву). Всередині блоку_виконання значення поточного елемента масиву може бути отримане за допомогою змінної $value. Виконання блоку_виконання відбувається стільки разів, скільки елементів у масиві $array.

Друга форма запису на додаток до перерахованого вище на кожному кроці циклу записує ключ поточного елемента масиву в змінну $key, що теж можна використовувати в блоці_виконання.

Коли foreach починає виконання, внутрішній покажчик масиву автоматично встановлюється на перший елемент.

```
<?php
$names = array("Іван","Петро","Семен");
foreach ($names as $val) {
    echo "Привіт, $val <br>";
```

```
    // виведе усім вітання
}
foreach ($names as $k => $val) {
    // крім вітання,
    // виведемо номера в списку, тобто ключі
    echo "Привіт, $val ! Ти в списку під номером $k <br>";
}
?>
```

7.3. Оператори передачі керування <вверх>

Іноді у випадку особливих обставин потрібно негайно завершити роботу циклу і передати керування першої інструкції програми, розташованої за останньою інструкцією циклу. Для цього використовують оператори break і continue.

break

Оператор break закінчує виконання поточного циклу, чи то for, foreach, while, do..while або switch. break може використовуватися з числовим аргументом, що дає вказівку, роботу скількох керуючих структур, що містять його, потрібно завершити.

```
<?php
$i=1;
while ($i) {
    $n = rand(1,10);
    // генеруємо довільне число від 1 до 10
    echo "$i:$n ";
    // виводимо номер ітерації і згенероване число
    if ($n==5) break;
/* Якщо було згенеровано число 5,
то припиняємо роботу циклу. У цьому випадку
усе, що знаходиться після цього рядка
у середині циклу, не буде виконане */
    echo "Цикл працює <br>";
    $i++;
}
echo "<br>Число ітерацій циклу $i ";
?>
```

Результатом роботи цього скрипта буде приблизно наступне:

```
1:7 Цикл працює
2:2 Цикл працює
3:5
Число ітерацій циклу 3
```

continue

Іноді потрібно не цілком припинити роботу циклу, а тільки почати його нову ітерацію. Оператор continue дозволяє пропустити подальші інструкції з блоку_виконання будь-якого циклу і продовжити виконання з початку. continue можна використовувати з числовим аргументом, що вказує, скільки утримуючих його керуючих конструкцій повинні завершити роботу.

Замінимо в прикладі попереднього параграфа оператор break на continue. Крім того, обмежимо кількість кроків циклу чотирма.

```
<?php
$i=1;
```

```
while ($i<4) {
    $n = rand(1,10);
   // генеруємо довільне число від 1 до 10
    echo "$i:$n ";
   // виводимо номер ітерації і згенероване число
    if ($n==5) {
        echo "Нова ітерація ";
        continue;
/* Якщо було сгенерировано число 5,
то починаємо нову ітерацію циклу,
$i не збільшується */
    }
    echo "Цикл працює <br>";
    $i++;
}
echo "<br>Число ітерацій циклу $i ";
?>
```

Результатом роботи цього скрипта буде

```
1:10 Цикл працює
2:5 Нова ітерація 2:1 Цикл працює
3:1 Цикл працює
Число ітерацій циклу 4
```

7.4. Оператори включення <вверх>

include

Оператор include дозволяє включати код, що міститься в зазначеному файлі, і виконувати його стільки разів, скільки програма зустрічає цей оператор. Включення може використовуватися кожним з перерахованих способів:

```
include 'ім'я_файлу';
include $file_name;
include ("ім'я_файлу");
```

Приклад. Нехай у файлі params.inc у нас зберігається набір якихось параметрів і функцій. Щоразу, коли нам потрібно буде використовувати ці параметри (функції), ми будемо вставляти в текст нашої основної програми команду include 'params.inc'.

params.inc

```
<?php
$user = "Вася";
$today = date("d.m.y");
/* функція date() повертає дату
і час (тут – дату у форматі
день.місяць.рік) */
?>
```

include.php

```
<?php
include ("params.inc");
/* змінні $user і $today задані у файлі
params.inc. Тут ми теж можемо ними
користуватися завдяки команді
include("params.inc") */
echo "Привіт, $user!<br>";
    // виведе "Привіт, Вася!"
echo "Сьогодні $today";
    // виведе, наприклад, "Сьогодні 7.07.05"
?>
```

Помітимо, що використання оператора include еквівалентно простій вставці змістовної частини файлу params.inc у код програми include.php. Можливо, тоді можна було б в params.inc записати простий текст без всяких тегів, що вказують на те, що це php-код? Не можна! Справа в тому, що в момент вставки файлу відбувається переключення з режиму обробки PHP у режим HTML. Тому код всередині файлу, що включається, який потрібно обробити як PHP-скрипт, повинен бути вкладений у відповідні теги.

Пошук файлу для вставки відбувається за наступними правилами.

- Спочатку ведеться пошук файлу в include_path щодо поточного робочого каталогу.
- Якщо файл не знайдений, то пошук ведеться в include_path щодо каталога поточного скрипта.
- Параметр include_path, обумовлений у файлі налаштувань PHP, задає імена каталогів, у яких потрібно шукати файли, що включаються.

Наприклад, ваш include_path це . (тобто поточний каталог), поточний робочий каталог це /www/. В основний файл include.php ви включаєте файл my_dir/a.php, що у свою чергу включає b.php. Тоді парсер першою справою шукає файл b.php у каталозі /www/, і якщо такого немає, то в каталозі /www/my_dir/.

Крім локальних файлів, за допомогою include можна включати і зовнішні файли, вказуючи їхні url-адреси. Дана можливість контролюється каталогом url_fopen_wrappers у файлі налаштувань PHP і за замовчуванням, як правило, включена. Але у версіях PHP для Windows до PHP 4.3.0 ця можливість не підтримується зовсім, поза залежністю від url_fopen_wrappers.

include() – це спеціальна мовна конструкція, тому при використанні всередині умовних блоків її потрібно взяти у фігурні дужки.

```
<?php
/*  Це невірний запис. Одержимо помилку.
    Ми ж вставляємо не одну команду,
   а декілька, вони лиш записані в іншому файлі */
if ($condition) include("first.php");
else include("second.php");
// А от так правильно.
if ($condition){ include("first.php"); }
else { include("second.php"); }
?>
```

При використанні include можливі два види помилок – помилка вставки (наприклад, не можливо знайти зазначений файл, невірно написана сама команда вставки і т.п.) або помилка виконання (якщо помилка міститься у файлі, що вставляється,). У будь-якому випадку при помилці в команді include виконання скрипта не завершується.

require

Цей оператор діє приблизно так само, як і #include у C++. Усе, що ми говорили про include, лише за деякими виключеннями, справедливо і для require. Require також дозволяє включати в програму і виконувати який-небудь файл. Основна відмінність require і include полягає в тому, як вони реагують на виникнення помилки. Як вже говорилося, include видає попередження, і робота скрипта продовжується. Помилка в require викликає фатальну помилку роботи скрипта і припиняє його виконання.

Умовні оператори на require() не впливають. Хоча, якщо рядок, у якому з'являється цей оператор, не виконується, то жоден рядок коду з файлу, що вставляється, теж не виконується. Цикли також не впливають на require(). Хоча код, що міститься у файлі, що вставляється, є об'єктом циклу, але вставка сама по собі відбувається лише один раз.