Условные операторы

- Конструкция if
- Сложные условия
- Неявное преобразование к boolean
- Конструкции else, elseif
- Тернарный оператор
- Оператор ??

Конструкция if

Условные операторы являются, пожалуй, **наиболее распространенными конструкциями** во всех алгоритмических языках программирования. В сценариях PHP условные операторы обеспечивают **основу для принятия решений**.

Условные операторы, по существу, определяют, будет ли выполняться часть сценария в зависимости от результата конкретного выражения (т.е., возвращает ли выражение логическое значение **true** или **false**).

> ***К сведению***. **Логическое выражение**, возвращающее значение **true** или **false** называют **предикатом** (в программировании понятие используется часто).

Рассмотрим основные условные операторы языка PHP.

Синтаксис конструкции if:

```
<?php
     if (логическое выражение) инструкция;
```

Конструкция if содержит **логическое выражение**. Если логическое выражение (**предикат**) истинно, то инструкция, следующая за конструкцией if, будет **исполнена**, если логическое выражение (**предикат**) ложно, то следующая за if инструкция исполнена **не будет**.

Пример:

Если переменная $a больше переменной $b, будет выведена строка "значение а больше, чем b":

```
<?php
    if ($a > $b) echo "Значение а больше, чем b";
```

Если переменная $a не равна нулю, будет выведена строка "значение а истинно (true)":

```
<?php
    if ($a) echo "Значение а истинно (true)";
```

Если переменная $a равна нулю, будет выведена строка "значение а ложно (false)":

```
<?php
    // применение логического оператора отрицания в условии
    if (!$a) echo "Значение а ложно (false)";
```

Часто при написании кода необходим **блок инструкций**, который будет выполняться при определенном условном критерии, тогда эти инструкции необходимо поместить в **фигурные скобки {...}** .

Пример:

```
<?php
    if ($a > $b) {
        echo "a больше b"; // инструкция_1
        $b = $a; // инструкция_2
    }
```

Если $a > $b приведенный пример выведет сообщение: "a больше b". Затем значение переменной $a будет присвоено переменной $b. Данные операторы выполняются в **теле конструкции if**.

Сложные условия

Иногда может возникнуть необходимость составить какое-то **сложное условие**. Для этого существуют операторы:

- **and** (логическое **И**),
- **or** (логическое **ИЛИ**).

Логическое И

Логическое И позволяет задать **одновременность условий**. В следующем примере логическое выражение будет истинно, если переменная $**age** попадает в заданный диапазон:

```
<?php
    if ($age > 2 and $age < 10) {
        // ...
    }
```

Условия могут налагаться не на одну переменную, а на разные:

```
<?php
    if ($login == "admin" and $password == "12345") {
        // ...
    }
```

Логическое ИЛИ

Логическое ИЛИ требует выполнения **хотя бы одного условия**. В следующем примере логическое выражение будет истинно, если выполняется хотя бы одно из условий:

```
<?php
    if ($login == "admin" or $role == "master") {
```

```
        // ...
    }
```

Неявное преобразование к boolean

В теме про типы данных в PHP мы разбирали, как **явно привести** значение к какому-либо типу.

Пример явного приведения типов:

```
<?php
    $x = 3;
    var_dump($x); // int(3)
    echo "<p>";
    $x = (boolean)3;
    var_dump($x); // bool(true)
```

> ***Важно***. В логических выражениях условных конструкций всегда происходит **неявное преобразование к типу boolean**.

Например, следующее условие:

```
<?php
    if (3) {
        echo 'Условие выполнено';
    }
```

выполнится успешно, так как число 3 будет преобразовано к **true**. Любая непустая строка тоже будет преобразована к логическому **true**.

```
<?php
    if ("Строка") {
            echo 'Условие выполнено';
```

```
}
```

К **false** будут приводиться следующие значения:

- '' (**пустая строка**)
- 0 (**число 0**)
- '0' (**строка, состоящая из одного нуля**)

Таким образом, любое ненулевое число и ненулевая строка (кроме строки - 0), **будут преобразованы в true и условие выполнится**.

Конструкции else, elseif

Часто возникает необходимость исполнения операторов не только в **теле конструкции if** (если условие выполнено), но и в случае, если **условие конструкции if не выполнено**. В данной ситуации нельзя обойтись без конструкции **else** (иначе).

Конструкция else

В целом, такая конструкция будет называться конструкцией **if-else**.

Синтаксис конструкции if-else:

```
<?php
     if (логическое выражение)
          инструкция_1;
     else
          инструкция_2;
```

Действие конструкции **if-else** следующее:

если логическое выражение истинно:

- **то** выполнится инструкция_1,

- **иначе** - выполнится инструкция_2.

Если на месте **инструкция_1 или инструкция_2** должны находиться несколько команд, то они **заключаются в фигурные скобки**.

Например:

```
<?php
     if (логическое выражение) {
          инструкция_1;
          инструкция_2;
          .......
          инструкция_n;
     } else {
          инструкция_1;
          инструкция_2;
          .......
          инструкция_n;
     }
```

Конструкция **if-else** имеет **альтернативный синтаксис**:

```
<?php
     if (логическое выражение):
          инструкция_1;
     else:
          инструкция_2;
     endif
```

Обратите внимание на расположение двоеточия (:). Если его пропустить, будет сгенерировано **сообщение об ошибке**.

Конструкция elseif

Если нужно проверить несколько условий подряд, то для этой цели используется оператор **elseif**.

Elseif - это комбинация конструкций **if** и **else**. Эта конструкция расширяет условную конструкцию **if-else**.

Синтаксис конструкции elseif

```
<?php
     if (логическое выражение_1)
          инструкция_1;
     elseif (логическое выражение_2)
          инструкция_2;
     else
          инструкция_3;
```

Практический пример использования конструкции **elseif**.

example_1. Конструкция elseif

```
<?php
     $a = 8;
     $b = 5;
     if ($a > $b) {
          echo "<h3>a больше, чем b</h3>";
     } elseif ($a == $b) {
          echo "<h3>a равен b</h3>";
     } else {
          echo "<h3>a меньше, чем b</h3>";
     }
```

```
?>
```

Считается, что конструкция **elseif** не очень удобна и засоряет код, поэтому в качестве альтернативы используют **конструкцию выбора** (рассматривается дальше в этой теме).

И еще: как обычно, блоки **else** и **elseif** можно опускать

Тернарный оператор

В PHP есть оператор, который представляет собой сокращённую форму конструкции **if-else**. Это **тернарный оператор**.

Существует распространенная в программировании ситуация, когда в случае выполнения некоторого условия переменной необходимо присвоить одно значение и в случае невыполнения этого условия другое значение.

В следующем примере переменной $min присваивается наименьшее из значений $var1 и $var2 с помощью конструкции if else:

```
<?php
    if ($var1 < $var2)
        $min = $var1;
    else
        $min = $var2;
```

На практике подобные действия оказались настолько распространенными, что был разработан **специальный условный оператор**, выполняющий данные действия - сокращенный способ присваивания значения переменной на основе выполнения того или иного условия.

Этот оператор использует три операнда (по этой причине его часто называют тернарным) и записывается с помощью двух знаков "**?**" и "**:**".

Он является единственным оператором в PHP, который использует более двух операндов.

С помощью тернарного оператора можно записать предыдущий пример следующим образом:

```
<?php
      $min = ($var1 < $var2) ? $var1 : $var2;
```

При этом он возвращает разные результаты, в зависимости от того, выполнено ли условие или нет.

В общем виде использование оператора выглядит следующим образом:

```
условие ? результат_если_true : результат_если_false
```

Тернарный оператор в примере нахождения модуля числа.

example_2. Тернарный оператор

```
<?php
      $x = -2;
      $mod = $x >= 0 ? $x : -$x;
      // вывод модуля числа переменной x
      echo '<h3>Модуль числа: ' . $mod, '</h3>';
?>
```

Оператор ??

Существует **короткая форма** (синтаксический сахар) **тернарного оператора** с **isset**().

```
$login = isset($login) ? $login : 'no-name';
```

Оператор **isset** () — возвратит **true**, если переменная $login существует, и её значение не **null** (будем изучать далее).

Короткая форма записи тернарного оператора обозначается **??** и называется **null coalescing operator** (оператор объединения с null):

```
$login = $login ?? 'no-name';
```

Общий вид использования оператора **??** выглядит следующим образом:

```
результат = значение_1 ?? значение_2
```

PHP выбирает из двух вариантов, если переменная есть, то используется её значение, **если переменной нет или она равна null** - то используется значение, указанное после **??**.

example_3. Оператор ??

```
<?php
    // инициируем логин
    // если закомментировать переменную скрипт не рухнет
    // $login = 'master';
    $login = $login ?? 'no-name';
    // вывод логина
    echo '<h3>Логин пользователя: ' . $login, '</h3>';
?>
```

Следует отметить, что в приведенном написании:

```
<?php
    if ($login ?? "no-name") {
        // ...
    }
```

оператор **??** не имеет смысла, поскольку нам не нужно возвращать значение предиката, а нужно лишь узнать, существует ли переменная $**login**.

Таким образом, в следующем демонстрационном примере все способы написания условия являются правильными. Вопрос в количестве строк кода.

example_4. Условные операторы

```php
<?php
     // инициируем логин
     // если закомментировать переменную скрипт не рухнет
     // $login = 'master';
     if (isset($login))
          $login = $login;
     else
          $login = "no-name";
     echo 'Логин в конструкции if: ', $login;
     echo '<p>';
     $login = isset($login) ? $login : 'no-name';
     echo 'Логин в тернарном операторе: ', $login;
     echo '<p>';
     $login = $login ?? 'no-name';
     echo 'Логин в операторе ??: ', $login;
?>
```

Задание № 29

==========

Используя **таблицу истинности** трёхвходового элемента "**Исключающее ИЛИ**", напишите программу, реализующую функционал строгой дизъюнкции.

Схема элемента:

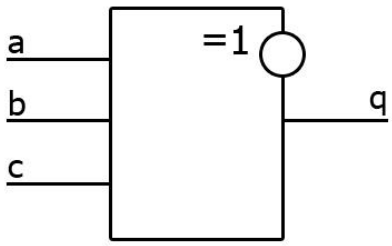


Таблица истинности элемента:

Вход			**Выход**
a	**b**	**c**	**q**
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Алгоритм работы:

Если **a = 0** и **b = 0** и **c = 0** то **q = 0**

Если **a = 0** и **b = 0** и **c = 1** то **q = 1**

...

Задание № 30

==========

Напишите программу: Проверка памяти:

- **вспомните отрывок** из поэмы АС Пушкина "Евгений Онегин" - "Мой дядя самых честных правил..." ;
- **по количеству воспроизведенных** строк программа должна определить уровень памяти.

Входные данные – **количество вспомненных строк**.

Выходные данные – предложенный **текст**.

Если вы вспомнили 2 строки:

Текст: Прямо беда с вами

Если 4:

Текст: Плохо.

Если 6:

Текст: Кажется, что вы где-то учились.

Если 8:

Текст: Вы среднестатистический человек.

Если 10:

Текст*:* Нормально.

Если 12:

Текст: Хорошо.

Если 14:

Текст: Отлично!

Иначе:

Мне некогда заниматься ерундой всякой. Какая-то дурацкая инструкция.

Задание № 31

==========

Рассчитайте **корни квадратного уравнения (x1, x2)**. При вычислении корней используйте функцию нахождения корня **sqrt** ().

Алгоритм нахождения корней квадратного уравнения:

Общий вид уравнения:

- **a * x^2 + b * x + c = 0**

Находим дискриминант (**D**):

- **$d = b^2 – 4 * a * c**

Если **D > 0**:

- **$x1 = (-$b + sqrt($d)) / 2 * $a;**
- **$x2 = (-$b + sqrt($d)) / 2 * $a;**

Если **D < 0**:

- Нет корней

Если **D = 0**:

- **$x = (-$b + sqrt($d)) / 2 * $a;**

Циклы

Введение

На втором месте по частоте использования, после конструкций условных операторов, находятся циклы. По частоте использования на втором, а по важности понимания **процессов**, происходящих при **обработке данных** - НА ПЕРВОМ!

> ***Важно***. **Нет другого способа** обработки массивов данных кроме организации **циклов**. Понимание алгоритмов перебора массивов данных циклами – **залог успешного программирования** в фреймворках и не только.

Циклы позволяют повторять определенное (и даже неопределенное - когда работа цикла зависит от условия) количество раз различные инструкции. Инструкции цикла называются **телом цикла**. Перебор **всех инструкций** цикла называется **итерацией**.

PHP поддерживает несколько видов циклов:

- цикл с **предусловием** (while),
- цикл с **постусловием** (do-while),
- цикл **со счетчиком** (for),
- специальный цикл **перебора массивов** (foreach).

> ***К сведению***. В подавляющем количестве случаев при работе с базами данных используются циклы **while** и **foreach**.

При программировании циклов есть возможность использования операторов **break** и **continue**. Первый из них прерывает **работу всего цикла**, а второй - **только текущей итерации**.

И хотя я написал, что наиболее часто используются два вида циклов, рассмотрим практику применения всех циклов PHP.

Цикл с предусловием while

Цикл с предусловием **while** работает по следующим принципам:

1. вычисляется **значение предиката**;
2. **если значение истинно**, выполняется **тело цикла**, в противном случае - переход на следующий **за циклом** оператор.

> ***Еще раз***. **Предикат -** логическое выражение, возвращающее значение **true** или **false**.

Синтаксис цикла с предусловием:

```
while (логическое_выражение)инструкция;
```

В данном случае **телом цикла** является **инструкция**. Обычно тело цикла состоит из более чем одной инструкции.

```
while (логическое_выражение){
     инструкция_1;
     инструкция_2;
     ...
     инструкция_n;
```

```
}
```

Пример цикла с предусловием while:

```
<?php
      $x=0;
      while ($x++<10) echo $x;
      // выводит 12345678910
```

Обратите внимание на последовательность выполнения операций условия $x++<10.

- сначала **проверяется условие**, а только потом **увеличивается** значение переменной,
- если мы поставим операцию инкремента перед переменной (++$x<10), то сначала будет **выполнено увеличение** переменной, а только затем - **сравнение**.

Этот же цикл можно записать по-другому:

example_1. Цикл с предусловием

```
<?php
      $i=0;
      while ($i<10){
            $i++; // увеличение счетчика (итератора) цикла
            echo $i;
      }
      // выводит 12345678910
?>
```

Если увеличить счетчик после выполнения оператора **echo**, получим строку "0123456789". В любом случае, имеем 10 итераций.

> ***Еще раз***. **Итерация** – это **однократное** выполнение всех инструкций внутри тела цикла. Переменную **$i** часто называют – **итератором цикла**.

Для демонстрации возможностей работы с циклами инициализируем **простой индексный массив** и переберем его элементы.

> ***Забегая вперед***. Встроенные функции рассматриваются отдельной темой, но рассмотреть возможности обработки массивов циклом **while** без использования встроенной функции **count** невозможно.
>
> Функция **count**() - возвращает количество элементов массива.

example_2. Перебор элементов простого массива

```
<?php
	// инициируем массив
	$album = array(1, "Atom Heart Mother", "10 октября 1970",
	"EMI", "LP, CD", "Золотой (USA)");
	// функция count возвращает количество элементов массива
	$count = count($album);
	// инициируем итератор цикла
	$i = 0;
	while($i < $count) {
		// выводим очередное значение массива
		echo "-- " . $album[$i], "<br />";
		$i++;
	}
?>
```

Инициализируем и переберем элементы **двумерного индексного массива**.

example_3. Перебор элементов двумерного массива

```
<?php
     // инициализируем массив
     $disc = array(
          array(1, "Atom Heart Mother", "10 октября 1970", "EMI",
          "LP, CD", "Золотой (USA)"),
          array(2, "Meddle", "30 октября 1971", "EMI", "Vinyl,
          CD", "Платиновый (USA)"),
          array(3, "Obscured by Clouds", "3 июня 1972",
          "Capitol", "LP, CD", "Золотой (USA)"),
          array(4, "The Dark Side of the Moon", "17 марта 1973",
          "EMI", "LP, CD", "Платиновый (USA)"),
          array(5, "Wish You Were Here", "15 сентября 1975",
          "Harvest", "CD, SACD",   "Платиновый (USA)")
     );
     // определяем размер массива
     $count = count($disc);
     // инициализируем итератор цикла
     $i = 0;
     while($i < $count) {
          echo "
               ID альбома: {$disc[$i][0]} <br />
               Название: {$disc[$i][1]} <br />
               Дата выпуска: {$disc[$i][2]} <br />
               Лейбл: {$disc[$i][3]} <br />
               Формат: {$disc[$i][4]} <br />
```

```
                Статус: {$disc[$i][5]} <p>
                <hr />
            ";
            // инкремент итератора
            $i++;
        }
?>
```

Подобно конструкции условного оператора if, можно группировать операторы внутри тела цикла while, используя следующий **альтернативный синтаксис**:

```
while (логическое_выражение):
    инструкция_1;
    ...
endwhile;
```

Пример использования альтернативного синтаксиса:

```
<?php
    $x = 1;
    while ($x <= 10):
        echo $x;
        $x++;
    endwhile;
```

Цикл с постусловием do while

В отличие от цикла while, этот цикл проверяет значение выражения **не до, а после каждого прохода (итерации) тела цикла**.

> ***Важно***. Таким образом, тело цикла выполняется **хотя бы один раз** (не зависимо от условия).

Синтаксис цикла с постусловием такой:

```
do {
      // тело цикла
      }
while (логическое_выражение);
```

После очередной итерации проверяется значение **предиката (логического выражения)**, если значение истинно, управление передается вновь на начало цикла, в противном случае цикл завершается.

Пример скрипта, показывающего работу цикла с постусловием do-while:

```
<?php
      $x = 1;
      do {
            echo $x;
      } while ($x++<10);
```

Рассмотренный сценарий выводит: 12345678910

Цикл do-while довольно редко используется при программировании web-приложений.

Цикл со счетчиком for

Цикл со счетчиком используется **для выполнения тела цикла определенное число раз**.

С помощью цикла for можно (и нужно) создавать конструкции, которые будут выполнять действия совсем не такие тривиальные, как простая переборка значения счетчика.

Синтаксис цикла for такой:

```
for (инициализирующие_команды; условие; команды_после_итерации){
    // тело цикла
}
```

Алгоритм цикла **for**:

- цикл for начинает свою работу с выполнения **инициализирующих команд**. Данные команды выполняются только **один раз**;
- после этого проверяется **условие** цикла, если оно истинно (true), то выполняется **тело цикла**;
- после того, как будет выполнен **последний оператор тела цикла**, выполняются **команды после итерации**;
- затем снова проверяется **условие** цикла. Если оно истинно (true), выполняется **тело цикла** и **команды после итерации**, и.т.д.

```
<?php
    for ($x=0; $x<10; $x++) echo $x;
    // выводит: 0123456789
```

или так:

```
<?php
    for ($x=0; $x++<10;) echo $x;
    // выводит 12345678910
```

В данном примере мы обеспечили увеличение счетчика **при проверке** логического выражения. В таком случае нам **не нужны команды**, выполняющиеся после итерации.

Если необходимо указать **несколько** инициализирующих команд, их можно **разделить запятыми**, пример:

```
<?php
	for ($x=0, $y=0; $x<10; $x++, $y++) echo $x;
	// выводит 0123456789
```

Еще один пример использования нескольких команд в цикле **for**:

example_4. Инструкция страдающим бессонницей

```
<?php
	for($i=1, $k=" слоник <br/>"; $i<200; $i++) {
		$str = $i . $k;
		echo $str;
	}
?>
```

Инициализируем **простой индексный массив** и переберем его элементы с использованием цикла **for**.

example_5. Перебор элементов простого массива циклом for

```
<?php
	// инициализируем массив
	$album = array(1, "Atom Heart Mother", "10 октября 1970",
	"EMI", "LP, CD", "Золотой (USA)");
	// определяем размер массива
	$count = count($album);
	// запускаем цикл
	for ($i=0; $i < $count; $i++) {
		// выводим значение массива
		echo $album[$i], "<br />";
```

```
    };
?>
```

Используя цикл **for** переберем элементы **двумерного индексного массива**.

example_6. Перебор элементов двумерного массива

```
<?php
    // инициализируем массив
    $disc = array(
        array(1, "Atom Heart Mother", "10 октября 1970", "EMI",
        "LP, CD", "Золотой (USA)"),
        array(2, "Meddle", "30 октября 1971", "EMI", "Vinyl,
        CD", "Платиновый (USA)"),
        array(3, "Obscured by Clouds", "3 июня 1972",
        "Capitol", "LP, CD", "Золотой (USA)"),
        array(4, "The Dark Side of the Moon", "17 марта 1973",
        "EMI", "LP, CD", "Платиновый (USA)"),
        array(5, "Wish You Were Here", "15 сентября 1975",
        "Harvest", "CD, SACD",   "Платиновый (USA)")
    );
    // определяем размер массива
    $count = count($disc);
    $html = "";
    // запускаем цикл
    for ($i=0; $i < $count; $i++) {
        // формируем html-строку
        $html .= "
            ID альбома: {$disc[$i][0]} <br />
```

```
                Название: {$disc[$i][1]} <br />
                Дата выпуска: {$disc[$i][2]} <br />
                Лейбл: {$disc[$i][3]} <br />
                Формат: {$disc[$i][4]} <br />
                Статус: {$disc[$i][5]} <p>
                <hr />
            ";
        };
        // выводим html-строку
        echo $html;
?>
```

Для цикла **for имеется альтернативный синтаксис**:

```
for(инициализирующие_команды; условие; команды_после_итерации):
        // тело цикла
endfor;
```

Рассмотренный пример (да и вообще любой цикл for) можно реализовать и через while, только это будет выглядеть не так изящно и лаконично.

Цикл перебора массивов foreach

В языке программирования PHP есть еще один специальный тип цикла - **foreach**.

> ***К сведению***. Цикл **foreach** специально предназначен для **перебора массивов**.

Синтаксис цикла foreach выглядит следующим образом:

```
foreach ($массив as $ключ=>$значение){
```

```
    // тело цикла
}
```

Здесь команды циклически выполняются **для каждого элемента массива**, при этом очередная пара **ключ=>значение** оказывается в переменных **$ключ** и **$значение**.

Пример работы цикла **foreach** с **ассоциативным** массивом.

example_7. Ассоциативный массив в цикле foreach

```
<?php
    // ассоциативный массив
    $user["name"] = "Татьяна";
    $user["surname"] = "М******";
    $user["age"] = 31;
    $user["email"] = "pretty-woman@gmail.com";
    $user["general_experience"] = 9;
    $user["edu_experience"] = 7;
    $user["city"] = "г. Печора";
    echo "<table border=1>
        <tr>
        <th>Ключ</th>
        <th>Значение</th>
        </tr>";
    // циклом foreach извлекаем пару ключ - значение ($key =>
    $value)
    foreach ($user as $key => $value) {
        echo "<tr><td> $key </td><td> $value </td></tr>";
    }
    echo "</table>";
```

```
?>
```

Пример работы цикла **foreach** с **индексным** массивом.

example_8. Индексный одномерный массив в цикле foreach

```
<?php
	// простой индексный массив
	$album = array(1, "Atom Heart Mother", "10 октября 1970",
	"EMI", "LP, CD", "Золотой (USA)");
	// циклом foreach извлекаем пару ключ - значение
	// и не важно какой массив индексный или ассоциативный
	foreach ($album as $key => $value) {
		echo "Ключ - $key, значение - $value <br />";
	};
?>
```

В предыдущих примерах в качестве **значения** ($value) выступала **скалярная переменная**.

> ***Важно***. В двумерных массивах **значением** является **вложенный массив**. Ничего сложного, но забывать об этом во время вывода **нельзя**.

example_9. Индексный двумерный массив в цикле foreach

```
<?php
	// двумерный индексный массив
	$disc = array(
		array(1, "Atom Heart Mother", "10 октября 1970", "EMI",
		"LP, CD", "Золотой (USA)"),
```

```
        array(2, "Meddle", "30 октября 1971", "EMI", "Vinyl,
        CD", "Платиновый (USA)"),
        array(3, "Obscured by Clouds", "3 июня 1972",
        "Capitol", "LP, CD", "Золотой (USA)"),
        array(4, "The Dark Side of the Moon", "17 марта 1973",
        "EMI", "LP, CD", "Платиновый (USA)"),
        array(5, "Wish You Were Here", "15 сентября 1975",
        "Harvest", "CD, SACD",   "Платиновый (USA)")
    );
    $html = "";
    /*
    в предыдущем примере (example_8.php) была конструкция -
    foreach ($album as $key => $value)
    почему $value ?
    - потому что значением массива было скалярное (строковое)
    значение,
    - например: 1, "Atom Heart Mother", "10 октября 1970" и т.д.
    почему в этом примере вместо $value использую $item?
    - потому что значение массива - ДРУГОЙ МАССИВ
    - например: array(1, "Atom Heart Mother", "10 октября 1970",
    "EMI", "LP, CD", "Золотой (USA)"),
    !! На самом деле программе все равно как будут названы
    переменные, но таким образом
    я показываю, что понимаю, какой тип переменной обрабатываю
    класс программиста в мелочах (кстати, я не претендую)
    */
    foreach ($disc as $key => $item) {
        // формируем строку для вывода
        $html .= "
```

```
                Ключ - $key <br />
                ID альбома: {$item[0]} <br />
                Название: {$item[1]} <br />
                Дата выпуска: {$item[2]} <br />
                Лейбл: {$item[3]} <br />
                Формат: {$item[4]} <br />
                Статус: {$item[5]} <p>
                <hr />
            ";
    };
    echo $html;
?>
```

У цикла **foreach** имеется и другая форма записи, которую следует применять, когда нас **не интересует значение ключа очередного элемента**. Выглядит она так:

```
foreach ($массив as $значение){
    // тело цикла;
}
```

В этом случае доступно лишь **значение очередного элемента массива**, но не его **ключ**.

Конструкция break

Очень часто для того, чтобы упростить логику какого-нибудь сложного цикла, удобно иметь возможность его прервать в ходе очередной итерации (к примеру, при выполнении какого-нибудь особенного

условия). Для этого и существует конструкция **break**, которая осуществляет **немедленный выход из цикла**.

Она может задаваться с одним необязательным параметром - числом, которое указывает, из какого вложенного цикла должен быть произведен выход.

По умолчанию используется 1, т. е. выход из текущего цикла, но иногда применяются и другие значения.

Синтаксис конструкции break:

```
// по умолчанию
break;
// для вложенных циклов (указывается номер прерываемого цикла)
break(номер_цикла);
```

Пример использования:

```
<?php
    $x=0;
    while ($x++<10) {
        if ($x==3) break;
        echo "<b>Итерация $x</b><br>";
    }
    // когда $x равен 3, цикл прерывается
```

Если нам нужно прервать работу определенного (вложенного) цикла, то нужно передать конструкции **break** параметр - **номер_цикла**, например, break(1).

Нумерация циклов выглядит следующим образом:

```
<?php
```

```
for (...) // третий цикл
{
    for (...) // второй цикл
    {
        for (...) // первый цикл
        {
            // ...
        }
    }
}
```

Конструкция continue

Конструкция continue так же, как и break, работает только в паре с **циклическими конструкциями**.

Она немедленно завершает **текущую итерацию цикла** и переходит к **новой**. Точно так же, как и для break, для continue можно указать уровень вложенности цикла, который будет продолжен по возврату управления.

В основном continue позволяет вам сэкономить количество фигурных скобок в коде и увеличить его удобочитаемость. Это чаще всего бывает нужно в циклах-фильтрах, когда требуется перебрать некоторое количество объектов и выбрать из них только те, которые удовлетворяют определенным условиям.

Приведу пример использования конструкции **continue**:

```
<?php
    $x=0;
    while ($x++<5) {
```

```
if ($x==3) continue;
     echo "<b>Итерация $x</b><br>";
}
// цикл прервется только на третьей итерации
```

Грамотное использование **break** и **continue** позволяет заметно улучшить читабельность кода и количество блоков else.

Перебор массивов циклами – важнейший навык разработчика, который отрабатывается исключительно на практических примерах. Время практиковаться.

Задание № 32

==========

В файле **teams.php** раздаточного материала вам предоставлен многомерный массив **$team**.

Используя цикл **foreach, выведите данные массива** в формате представленном ниже.

Формат вывода:

Группа: _____ (id = _____)
Страна: _____
Дата основания: _____
Стиль: _____
<hr/>
<p>

Задание № 33

==========

В файле **albums.php** раздаточного материала вам предоставлен **многомерный массив $album**.

Используя любой из циклов, выведите данные массива в **табличном виде**.

Пример вывода:

ID	Альбом	Дата выпуска	Страна	Идентификатор группы
...	...	...	...	...

Задание № 34

==========

В файле **tracks.php** раздаточного материала вам предоставлен **многомерный массив $track**.

Выведите треки музыкального сервиса в виде **нумерованного списка**.

Пример вывода:

Просмотр треков сервиса:

1. (id=1) Back in the Saddle
2. (id=2) Last Child
3. ...

Задание № 35

==========

В файле **tracks.php** раздаточного материала вам предоставлен **многомерный массив $track**.

Выберите треки альбома с **id_album = 6** (альбом: Back in Black (AC/DC)). Выведите отобранные данные в виде **маркированного списка**.

Пример вывода:

Треки альбома Back in Black (AC/DC):

- Hells Bells
- Shoot to Thrill
- ...

Задание № 36

==========

В файле **tracks.php** раздаточного материала вам предоставлен **многомерный массив $track**.

Выведите массив **$track** в браузер, используя при этом:

1. Цикл **do-while**,

   формат вывода: **маркированный** список.

   Пример: (id_трека) Название_трека

2. Цикл **for**,

   формат вывода: **нумерованный** список.

   Пример: : (id_трека) Название_трека (id альбома: ___)

3. Цикл **foreach**,

   формат вывода: **табличное** представление.

   Пример:

id	**Трек**	**Примечание**	**Id-альбома**

4. Цикл **while**,

   формат вывода: **табличное** представление.

   Пример:

id	**Трек**	**Примечание**	**Id-альбома**

Конструкции

- Конструкция выбора (переключатель)
- Конструкция включений include
- Конструкция включений require
- Конструкции однократного включения
- Включения удаленных файлов

Конструкция выбора (переключатель)

Оператор **switch** похож на ряд операторов if с одинаковым условием. Во многих случаях вам может понадобиться сравнивать одну и ту же **переменную** (или выражение) с **множеством различных значений** и выполнять различные участки кода в зависимости от того, какое значение принимает эта переменная (или выражение).

> ***К сведению***. Конструкция switch-case предназначена для **выбора действий** в зависимости от **значения** указанного выражения.

Следующий пример иллюстрируют два различных способа написать одно и то же. Один способ использует ряд операторов **if** и **elseif**, а другой - оператор **switch**.

example_1. Сравниваем конструкции

```
<?php
     $i = 1;
     // условная конструкция
     if ($i == 0) {
        echo "i равно 0";
     } elseif ($i == 1) {
```

```
        echo "i равно 1";
    } elseif ($i == 2) {
        echo "i равно 2";
    }
    echo "<p>";
    // конструкция выбора
    switch ($i) {
        case 0:
            echo "i равно 0";
            break;
        case 1:
            echo "i равно 1";
            break;
        case 2:
            echo "i равно 2";
            break;
    }
?>
```

Управляющая структура switch передает управление тому из помеченных case операторов, для которого **значение константного выражения совпадает со значением переключающего выражения**.

Если значение переключающего выражения не совпадает ни с одним из константных выражений, то выполняется переход к оператору, помеченному меткой **default**. В каждом переключателе может быть не более одной метки default, однако она может **отсутствовать** вообще.

Оператор **switch** допускает **сравнение** с типом **string**.

example_2. Переключатель switch

```php
<?php
     $entity = "album";
     switch ($entity) {
        case "team":
            echo "Вывод данных таблицы `team`";
            break;
        case "album":
            echo "Вывод данных таблицы `album`";
            break;
        case "track":
            echo "Вывод данных таблицы `track`";
            break;
     }
?>
```

Операторный список для case может быть пуст, тогда переключатель передает управление в **операторный список следующей конструкции case**.

> ***К сведению***. PHP продолжает исполнять операторы до **конца блока switch** либо до тех пор, пока не встретит оператор **break**.

Если вы **не напишете оператор break** в конце секции case, PHP будет продолжать исполнять команды **следующей секции case**.

Пример :

```php
<?php
    switch ($x) {
```

```
        case 0:
        case 1:
        case 2:
            echo "x меньше, чем 3, но не отрицателен";
            break;
        case 3:
            echo "x=3";
    }
```

Когда ни одно значение из набора не совпало со значением выражения, тогда **выполняется блок default** (если он указан).

Пример:

```
<?php
    $x=3;
    switch ($x) {
        case 0:
            echo "x=0";
            break;
        case 1:
            echo "x=1";
            break;
        case 2:
            echo "x=2";
            break;
        default:
            echo "x не равен 0, 1 или 2";
    }
```

Данный скрипт выводит "x не равен 0, 1 или 2", поскольку переменная $x=3.

Конструкция switch-case также имеет **альтернативный синтаксис** (в квадратных скобках, как правило, указывают необязательный елемент):

```
switch(выражение):
     case значение1: команды1; [break;]
     . . .
     case значениеN: командыN; [break;]
     [default: команды_по_умолчанию; [break]]
endswitch;
```

Конструкция включений include

Языковая конструкция **include включает** и **выполняет** указанный файл. Если подключаемый файл не найден, то мы увидим ошибку уровня Warning (**предупреждение**) и скрипт спокойно **продолжит** своё выполнение.

Пример использования:

```
include "file.php";
```

Оператор **include** берет весь **текст**/**код**/**разметку**, который существует в указанном файле, и копирует его в файл, который использует оператор **include**.

example_3/index.php. Включаем в сценарий файл example_4.php

```
<?php
     include "inc.php";
?>
```

example_3/inc.php.

```
<pre>
<h3>
Хотя и сладостен азарт по сразу двум идти дорогам,
нельзя одной колодой карт играть и с Дьяволом и с Богом…
То плоть загуляла, а духу не весело,
то дух воспаряет, а плоть позабыта,
и нету гармонии, нет равновесия —
то чешутся крылья, то ноют копыта.
</h3>
© Игорь Губерман
</pre>
```

Конструкция **include** позволяет **включать удаленные файлы**, если такая возможность включена в **конфигурационном** файле PHP.

Конструкция включений require

Принцип действия языковой конструкции **require аналогичен с языковой констукцией include**, она **включает** и **выполняет** указанный файл, за исключением того, что при ошибке он выдаст **фатальную ошибку**, при этом работа скрипта **остановится**.

Пример использования:

```
require "test.php";
```

> ***К сведению***. Операторы **include** и **require идентичны**, за исключением случаев **отказа**.

Подключение файлов очень полезно, если вы хотите подключить один и тот же **PHP**, **HTML** или обычный **текст** на **нескольких страницах** веб-сайта.

В качестве примера рассмотрим включения в вывод сценария различных шаблонных страниц с HTML-кодом.

example_4/header.html. Файл header.html

```
<!DOCTYPE html>
<html>
	<head>
		<title>Программируем в PHP</title>
	</head>
	<body bgcolor="lightgray">
		<header>
		<h1>Веб-программирование и дизайн</h1>
		</header>
```

example_4/footer.html. Файл footer.html

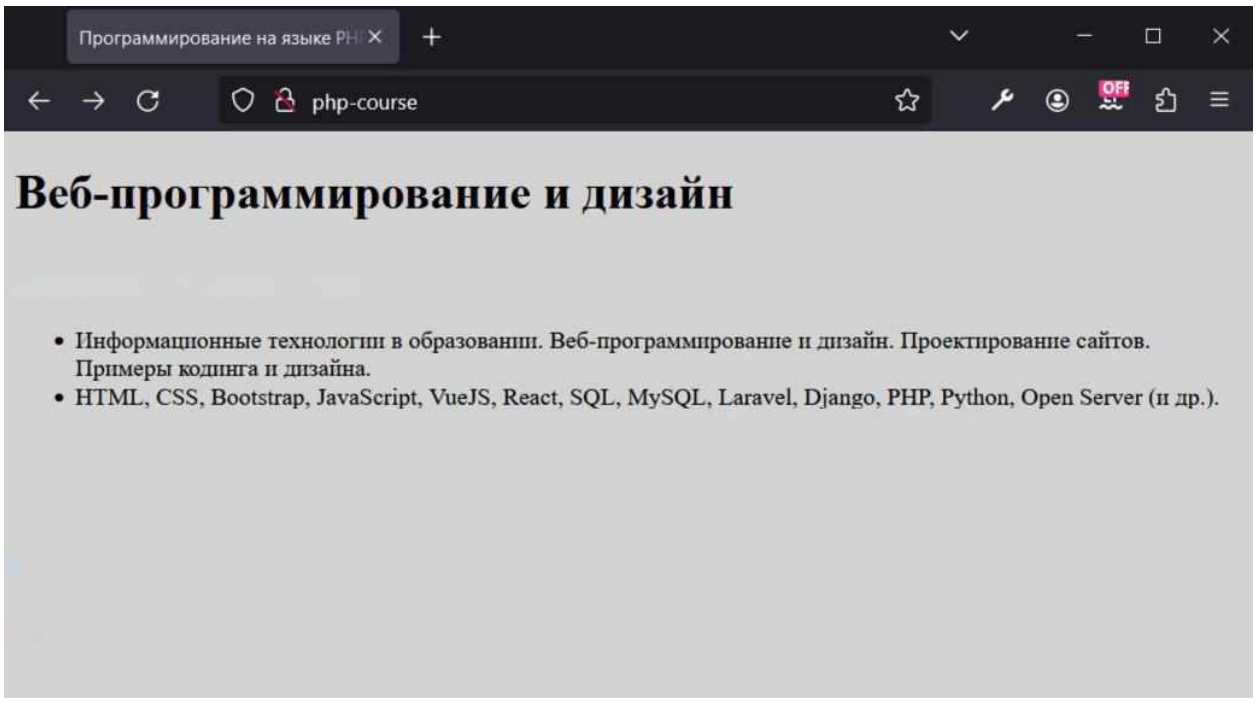
```
		<footer align="center">
		</footer>
	</body>
</html>
```

example_4/index.php. Файл index.php

```
<?php
    require "header.html";
?>

<ul>
    <li>Информационные технологии в образовании. Веб-
    программирование и дизайн. Проектирование сайтов. Примеры
    кодинга и дизайна.</li>
    <li>HTML, CSS, Bootstrap, JavaScript, VueJS, React, SQL,
    MySQL, Laravel, Django, PHP, Python, Open Server (и
    др.).</li>

</ul>
<?php
    require "footer.html";
?>
```



> ***Таким образом***. Конструкция require позволяет собирать сценарии PHP из нескольких **отдельных файлов**, которые могут быть как **html-страницами**, так **и php-скриптами**.

Конструкция require позволяет **включать удаленные файлы**, если такая возможность включена в **конфигурационном** файле PHP.

Конструкции однократного включения

В больших PHP сценариях инструкции include и require применяются **очень часто**. Поэтому становится довольно сложно контролировать, как бы случайно не включить один и тот же файл **несколько раз**, что чаще всего приводит к ошибке, которую сложно обнаружить.

В PHP предусмотрено решение данной проблемы.

> ***Важно***. Используя конструкции **однократного включения require_once** и **include_once**, можно быть уверенным, что один файл

> не будет включен дважды.

Работают конструкции однократного включения **require_once** и **include_once** так же, как и **require** и **include** соответственно. Разница в их работе лишь в том, что перед включением файла интерпретатор проверяет, **включен ли указанный файл ранее или нет**. Если да, то файл не будет включен вновь.

Конструкции однократных включений позволяют **включать удаленные файлы**, если такая возможность включена в **конфигурационном** файле PHP.

Включения удаленных файлов

PHP позволяет работать с **объектами URL, как с обычными файлами**. Упаковщики, доступные по умолчанию, служат для работы с удаленными файлами с использованием протокола FTP или HTTP.

Если "URL fopen-оболочки" включены в PHP (как в конфигурации по умолчанию), вы можете специфицировать файл, подключаемый с использованием URL (через HTTP), вместо локального пути. Если целевой сервер интерпретирует целевой файл как PHP-код, переменные могут передаваться в подключаемый файл с использованием URL-строки запроса, как в HTTP GET.

Строго говоря, это не то же самое, что подключение файла и наследование им области видимости переменных родительского файла; ведь скрипт работает на удалённом сервере, а результат затем подключается в локальный скрипт.

Задание № 37

==========

Вам предоставлен файл с кодом **динамического подключения** текстовых файлов в цикле **for**.

Запустите сценарий на исполнение. Протестируйте варианты:

1. файлы **подключаются** конструкцией **include**,
2. файлы **подключаются** инструкцией **require**.

Проанализируйте результат, сделайте выводы, запомните.

```
for($i=1; $i<=5; $i++) {
     // вариант 1
     include "$i.txt";
     // вариант 2
     // require "$i.txt";
     echo "<br />";
}
```

Задание № 38

==========

Выполнить **ревьюирование** (оптимизацию) сценариев решения предыдущих заданий:

- review-1/**index.php**.
- review-2/**index.php**.
- review-3/**index.php**.

Исключить инициализацию массивов из файлов сценариев. Вынести **определения** массивов в отдельный файл. **Подключить** соответствующий массив, используя **любую из конструкций включения** файлов.

Выполнить **вывод массива** в браузер. Формат вывода можно оставить без изменений или выполнить по своему усмотрению.

Задание № 39

==========

В файле **personnels.php** вам предоставлен многомерный массив работников образовательного учреждения - **$content**.

Подключите массив к сценарию. Выведите данные массива (запись), соответствующие введённому вами критерию. Критерий поиска может иметь вид:

```
$term = "surname";  // ключ, по которому будет выполняться поиск

$value = "Маркова";  // значение ключа для поиска
```

Поиск соответствия может быть организован следующим образом:

```
foreach ($content as $item) {
     if ($item[$term] == $value) {
          ...
     }
}
```

Вывод данных в браузер выполнить в формате:

id: _____
  // id_personnel

Фамилия: _____
  // surname

Имя: _____
  // name

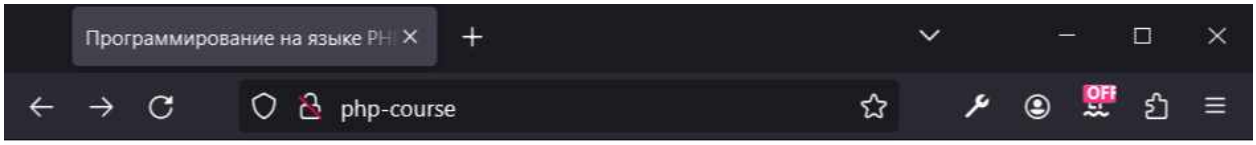
Отчество: _____
  // patronymic

Должность: _____
  // post

Категория: _____
  // category

Образование: _____
  // level_edu

Стаж работы в ОУ: _____ // experience_total



Управляющие конструкции

Конструкции

id: 4
Фамилия: Маркова
Имя: Елизавета
Отчество: Андреевна
Должность: Преподаватель
Категория: Первая
Образование: Высшее профессиональное
Стаж работы в ОУ: 19.11

Обработка связанных массивов

Элементы массива в PHP могут содержать значения **любого типа**, такие как **числа**, **строки**, **объекты**. Они также могут содержать и **другие массивы**, что фактически означает создание **многомерного или вложенного массива**.

В темах, посвященных многомерным (вложенным) массивам, мы разобрали, как их создать, как ими манипулировать, как организовать цикл по всему многомерному массиву в PHP.

> ***Главное***. Мы изучили, как **создать** и **перебрать** элементы **многомерного массива**.

Но, есть большой вопрос – кто сказал, что **несколько массивов**, пусть даже расположенные в разных файлах, не могут быть связанными? В теме **Связанные массивы** мы говорили об этом.

Эту тему изучим в виде выполнения нескольких практических заданий, выполняющих вывод в браузер данных связанных массивов.

Поэкспериментируем с **парой массивов** авторы – книги.

Прежде чем приступить, несколько слов о **данных**. Здесь и далее мы очень часто используем директорию **dump**. В такой директории будут храниться **файлы с данными**, используемыми для скриптов программ обучающих заданий.

> ***К сведению***. **Дамп** (англ. **dump** – сбрасывать) – файл, включающий в себя **содержимое базы данных**. Он содержит особые данные, благодаря которым можно легко **воссоздать копию БД**.

Файлы дампа базы данных могут быть созданы вручную либо экспортированы с помощью средств систем управления базами данных.

Файлы могут быть разных форматов, в нашем проекте мы используем файлы дампов с расширениями **.php** или **.sql**.

Задание 1 (файл example_1.php)

- Подключить массив $authors (dump/).
- Выполнить поиск в массиве и вывод в браузер автора по определенному идентификатору (id).

Задание 2 (файл example_2.php)

- Подключить массивы $authors, $books (dump/).
- Найти автора по определенному идентификатору (id).
- Найти книги искомого автора.
- Вывести книги с помощью функции print_r.

Задание 3 (файл example_3.php)

- Подключить массивы $authors, $books (dump/).
- Найти автора по определенному идентификатору (id).
- Найти книги искомого автора.
- Вывести книги в список.

Задание 4 (файл example_4.php)

- Подключить массивы $authors, $books (dump/).
- В строке запроса get-параметром передать идентификатор автора.
- Найти автора по полученному параметру идентификатора (id).
- Найти книги искомого автора.
- Вывести книги в список.

> ***К сведению***. В комментариях к коду скрипта получите первичное представление о **GET-параметре** и строке запроса. Подробному изучению этих вопросов посвящена одна из тем курса.

Задание № 40

==========

В файлах albums.php и tracks.php директории **dump** вам даны многомерные, связанные по первичному и внешнему ключам массивы **$albums** и **$tracks**.

Создайте файл сценария. Используя конструкцию **require**, **подключите файлы массивов раздаточного материала**.

Выведите данные массивов в виде вложенных списков в формате:

1. Название_альбома_1 (Страна)
    - трек_альбома_1
    - трек_альбома_2
    - и т.д.
2. Название_альбома_2 (Страна)
    - трек_альбома_1
    - трек_альбома_2
    - и т.д.
3. И т.д.

Для решения задачи используйте **вложенные циклы**.

P.S. Для решения задачи необходимо вспомнить назначение первичного и внешнего ключей массивов из темы: "Массивы", тема: "Связанные массивы".

Задание № 41

==========

В файлах albums.php и tracks.php директории **dump** вам даны многомерные, связанные по первичному и внешнему ключам массивы **$albums** и **$tracks**.

Создайте файл сценария. Используя конструкцию **require**, **подключите файлы массивов раздаточного материала**.

Используя **GET-параметр строки запроса**, передайте идентификатор альбома для вывода в браузер.

Выведите название альбома и его треки в виде **списка**:

Название_альбома (Страна)

- трек_альбома_1
- трек_альбома_2
- и т.д.

Для решения задачи используйте **вложенные циклы**.

P.S. Для решения задачи необходимо вспомнить назначение первичного и внешнего ключей массивов из темы: "Массивы", тема: "Связанные массивы".

Задание № 42

==========

В файлах albums.php и tracks.php директории **dump** вам даны многомерные, связанные по первичному и внешнему ключам массивы **$albums** и **$tracks**.

Создайте файл сценария. Используя конструкцию **require**, **подключите файлы массивов раздаточного материала**.

Используя **get-параметр строки запроса**, передайте **идентификатор альбома** для вывода в браузер.

Выведите:

- **список альбомов** и их **треки** в виде вложенных **списков**, если GET-параметр **не передается**.
- **искомый альбом** и его **треки**, если GET-параметр **передается**.

Для решения задачи используйте **вложенные циклы**.

P.S. Успешным будет считаться любое решение, приводящее к выполнению поставленной задачи. В качестве решения вам предложены два варианта.

- Файл **index_1.php** (неоптимальный) выполнен студентом первого года обучения.
- Файл **index_2.php** выполнен студентом второго года обучения, после требования оптимизации кода первого файла.

P.P.S. Для решения задачи необходимо вспомнить назначение первичного и внешнего ключей массивов из темы: "Массивы", тема: "Связанные массивы".