

Встроенные функции (1)

- Введение
- Функции работы с переменными
- Математические функции

Введение

Функции представляют собой **фрагмент кода** (блок кода), который можно выполнять многократно в любом месте данной программы. Функции позволяют разделять программу на меньшие функциональные части.

К сведению. Иногда функцию ещё называют **подпрограммой**.

Мы привыкли, что **обычной переменной** можно присвоить **число**, **строку** или **массив**, а затем получить его обратно, обратившись к значению по имени переменной. Функции устроены похожим образом. Это тоже своего рода переменная, только вместо строки или числа в ней **хранится блок кода**, который вызывается при использовании этой переменной.

Функция — очень мощный инструмент **повторного использования кода**. Чтобы упростить себе работу, мы можем оформить в виде функции некоторую часть кода, которая используется в сценарии несколько раз. Затем, вместо копирования и вставки этой части кода, будет достаточно только **вызывать эту функцию**, как если бы мы обращались к переменной.

Разделяют два типа функций — **встроенные** и **пользовательские**.

Встроенные функции — это функции, которые за нас уже написали создатели языка программирования, мы можем просто брать их и использовать.

К сведению. Примером встроенных функций могут служить рассмотренные ранее функции вывода **echo** и **print**.

В PHP существуют сотни готовых функций на все случаи жизни. И нет никакой нужды запоминать названия или аргументы встроенных функций, достаточно научиться **пользоваться справочником** (хотя с опытом программирования приходят и знания некоторых из них).

В нашем конспекте для **общего понимания сути вопроса** рассмотрим некоторые наиболее популярные функции.

Пользовательские функции не предусмотрены синтаксисом языка, и создаются программистом самостоятельно. Эти функции, как правило, используются только внутри одного проекта или, даже, сценария, в целях решения поставленной задачи.

Что необходимо знать про любые функции:

- после названия функции всегда ставятся **круглые скобки** (могут быть пустые или с аргументами, перечисленными через запятые);
- функция может принимать информацию из программы через **список аргументов**, разделенных запятыми (аргументы читаются слева направо);
- если в функцию передаются аргументы не того типа, на который она "рассчитывает" (например string вместо array), то будет выдаваться либо **null**, либо **ошибка**;
- для вызова функции достаточно написать ее **имя** и **список аргументов (фактических параметров)** в круглых скобках.

Пример вызова встроенных функций:

```
<?php
```

```
print("Hello, User!"); // функция с одним аргументом
```

```
htmlspecialchars($message); // функция с одним аргументом
```

```
unset($login, $pwd); // функция двух аргументов
```

Важно. Наберите представленные примеры самостоятельно, запомните **названия** и **описания** функций. Для первого прочтения этого достаточно. PHP содержит сотни разных функций, в случае выполнения самостоятельных задач учитесь пользоваться **справочниками**.

Функции работы с переменными

isset

isset () — проверяет, что переменная **существует**, и её значение не **null**.

Описание

```
isset($var);
```

Определяет, была ли установлена переменная, значением отличным от **null**.

Параметр

- **var** – проверяемая переменная.

Возвращаемые значения

Если переменная **существует** и её значение **не null**, то функция вернёт **true**, иначе — **false**.

example_1. Функция **isset**

```
<?php  
  
    // 1. шаг
```

```
// выполним запрос /example_1.php

// 2. шаг

// выполним запрос с добавлением к маршруту GET-параметра
// например:

// /example_1.php?var=user

// 3. шаг

// выполним запрос с GET-параметром без значения
// например:

// /example_1.php?var=

if (isset($_GET["var"])){

    echo "<h3>GET-переменная var существует и имеет
    значение: {$_GET['var']}</h3>";

}

else {

    echo "<h3>GET-переменной var не существует</h3>";

}

?>
```

empty

empty () — проверяет, что переменная **не существует** или у нее пустое значение.

Описание

```
empty($var);
```

Переменная не должна существовать или ее значение должно быть пустым.

Параметр

- **var** – проверяемая переменная.

Возвращаемые значения

Если переменная **не существует**, у нее пустое значение или false, то функция вернёт **true**, иначе — **false**.

Не генерирует предупреждение, если переменная не существует.

example_2. Функция empty

```
<?php

// 1. шаг

// выполним запрос /example_1.php

// 2. шаг

// выполним запрос с добавлением к маршруту GET-параметра

// например:

// /example_1.php?var=user

// 3. шаг

// выполним запрос с GET-параметром без значения

// например:

// /example_1.php?var=

if (empty($_GET["var"])){

    echo "<h3>GET-переменной var не существует или она
    пустая</h3>";

}

else {

    echo "<h3>GET-переменная var существует и имеет
    значение: {$_GET['var']}";

}

?>
```

unset

unset () — удаляет одну или несколько перечисленных переменных.

Описание

```
unset($vars);
```

Функция не возвращает значения после выполнения.

Параметр

- **vars** – удаляемые переменные.

example_3. Функция unset

```
<?php

// выполним запрос с добавлением к маршруту GET-параметра
// например:
// /example_3.php?user=admin

if (isset($_GET["user"])) {

    echo "

    <h3>GET-переменная user существует и имеет значение:
    {$_GET['user']}</h3>

    ";

} else {

    echo "<h3>GET-переменной user не существует</h3>";

}

// удалим переменную

unset($_GET["user"]);

if (empty($_GET["user"])) {

    echo "<h3>GET-переменной user не существует</h3>";

};
```

?>

Разница между isset и empty

Выражение	isset(\$var)	empty(\$var)
unset(\$var)	false	true
\$var = null	false	true
\$var = 0	true	true
\$var = ""	true	true
\$var = array()	true	true
\$var = false	true	true
\$var = true	true	false
\$var = 10	true	false

Еще раз. Начинающему разработчику довольно трудно разобраться с отличием функций. Поэтому, подводя краткий итог, скажу еще раз:

- `isset()` – **true** если **переменная есть** (и не равна null).
- `empty()` – **true** если **переменной нет** (или есть но пустая/false).

Некоторые функции проверки переменных

is_bool — проверяет, является ли переменная булевой.

is_int — проверяет, является ли переменная целым числом.

is_numeric — проверяет, является ли переменная числом или строкой.

is_array — проверяет, является ли переменная массивом.

is_object — проверяет, является ли переменная объектом.

is_null — проверяет, является ли значение переменной равным null.

```
if (is_array($vars)) {  
    // инструкции  
};
```

Математические функции

pow

pow () — возведение в степень.

Описание

```
pow($num, $exponent);
```

Возвращает **num**, возведённое в степень **exponent**.

Параметр

- **num** - основание
- **exponent** - показатель степени

Возвращаемые значения

Основание **num** возведённое в степень **exponent**. Если оба аргумента - неотрицательные целые и результат может быть представлен как целое, то возвращаемый результат будет типа **int**. В ином случае результат будет типа **float**.

example_4. Возведение в степень

```
<?php  
  
    // функция возведения в степень  
  
    echo pow(3, 2), "<p>"; // 9
```



```
$num = 5;  
  
echo pow($num, 3); // 125  
  
?>
```

sqrt

sqrt () — возвращает квадратный корень аргумента.

Описание

sqrt (\$num) ;

Возвращает квадратный корень из **num**.

Параметр

- **num** - аргумент для вычисления.

Возвращаемые значения

Квадратный корень из **num** или специальное значение NAN для отрицательных чисел.

example_5. Вычисление квадратного корня

```
<?php  
  
// функция извлечения корня  
  
echo sqrt(9), "<p>"; // 3  
  
$num = 10;  
  
echo sqrt($num); // 3.16227766 ...  
  
?>
```

rand

rand () — генерирует случайное число.

Описание

```
rand([$min, $max]);
```

При вызове без параметров **min** и **max**, возвращает псевдослучайное целое в диапазоне от 0 до максимально возможного. Например, если вам нужно случайное число между 5 и 15 (включительно), вызовите rand (5, 15).

Параметр

- **min** - наименьшее значение, которое может быть возвращено (по умолчанию: 0).
- **max** -наибольшее значение, которое может быть возвращено (по умолчанию: максимально возможное).

Возвращаемые значения

Псевдослучайное значение в диапазоне от **min** (или 0) до **max**.

example_6. Получение случайного числа

```
<?php

// генерируем случайные числа

$first = rand(1,6);

$second = rand(1,6);

$third = rand(1,6);

// выводим изображения кубиков согласно

// сгенерированного числа

echo "<img src='cube/$first.jpg' width='200px'>";

echo "<img src='cube/$second.jpg' width='200px'>";

echo "<img src='cube/$third.jpg' width='200px'>";

?>
```

Замечание. На некоторых платформах максимально возможное число всего лишь 32768. Чтобы расширить диапазон, используйте параметры **min** и **max**.

abs

abs () — модуль числа.

Описание

abs (\$num) ;

Возвращает абсолютное значение **num**.

Параметр

- **num** – числовое значение.

Возвращаемые значения

Возвращает абсолютное значение **num**. Если **num** имеет тип **float**, возвращаемое значение также будет иметь тип **float**, иначе - **integer**.

example_7. Модуль числа

```
<?php
    // выведем тип числа
    echo gettype(abs(-4.2)), "<p>"; // double (4.2)

    $num = -5;
    $abs = abs($num);
    echo $abs; // 5 (integer)
?>
```

pi

pi () — возвращает число Пи.

Описание

pi () ;

Возвращает число Пи с точностью, значение по умолчанию которой 14. Константа **M_PI** даёт идентичный результат.

Параметр

У этой функции нет параметров.

Возвращаемые значения

Приближенное значение числа Пи в виде числа с плавающей точкой (float).

example_8. Приближенное значение числа Пи

```
<?php

    // 7 класс, курс геометрии

    // отношение длины окружности к её диаметру ;)

    echo pi(); // 3.1415926535898

    echo "<p>";

    echo M_PI; // 3.1415926535898

?>
```

Задание 43

=====

Вам предоставлен файл **teams.php** с массивом групп **\$content**.
Организуйте вывод данных в браузер в зависимости от переданного в строке запроса **GET-параметра id**:

1. GET-параметр передает идентификатор группы (пример:
/index.php?id=5).

В таком случае: вывод информации о группе, **согласно** указанного в параметре **идентификатора**.

2. GET-параметр передается, но его значение не определено (пример:
/index.php?id=) , или GET-параметр не передается (пример:
/index.php).

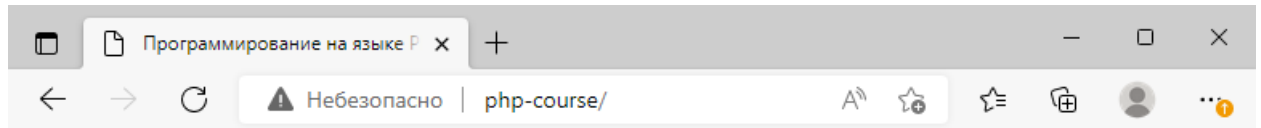
В таком случае: вывод информации о **всех группах**.

При решении постарайтесь использовать функции **isset()** или **empty()**, или и ту и другую.

Задание 44

=====

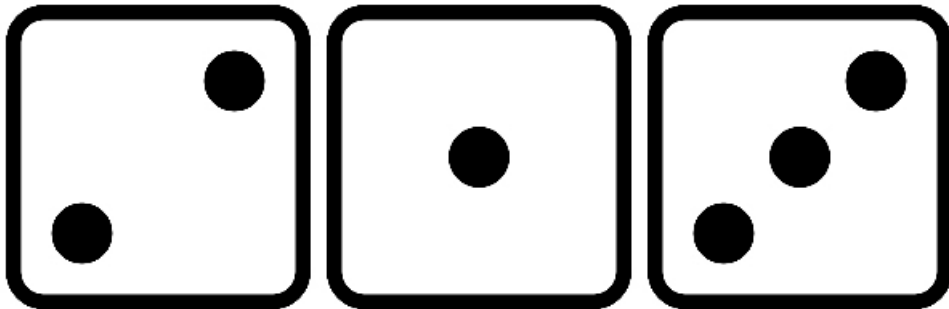
Написать сценарий, случайным образом выводящий изображения **трех** кубиков. Подсчитать и вывести в браузер **сумму набранных** очков.



Встроенные функции, часть 1

Поздравляем!

Неимоверными усилиями вам удалось набрать 6 баллов!



Задание 45

=====

Написать сценарий, случайным образом выводящий изображения **count** кубиков. Число **count** определять через соответствующую переменную **\$count**. Подсчитать и вывести в браузер сумму набранных очков.

Задание выполнить в два шага с помощью **циклов**:

1. **Инициализация** массива случайными числами.
2. **Вывод** изображений кубиков.

Задание 46

=====

Выполните ревьюирование сценария задания 3. Рассчитайте **корни квадратного уравнения (x1, x2)**. При вычислении корней используйте функции **pow()**, **sqrt()**.

Коэффициенты уравнения (a, b, c) передавать с использованием трех **GET-параметров**. Перед выполнением вычислений проверьте наличие коэффициентов в строке запроса.

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**:

- $x = -b / 2 * a;$

Встроенные функции (2)

- Функции обработки строк
- Функции форматного вывода

Функции обработки строк

Эти функции предназначены для выполнения различных манипуляций со строками.

addslashes

addslashes () — **экранирует** строку с помощью слешей.

Описание

addslashes (\$string) ;

Возвращает строку с обратным слешем перед символами, которые нужно **экранировать**, например, для последующего использования этой строки в **запросе к базе данных**.

Параметр

- **string** – экранируемая строка.

Экранируются следующие символы:

- **одинарная** кавычка (')
- **двойная** кавычка (")
- обратный **слеш** (\)
- **null** (байт null)

Возвращаемые значения

Возвращает экранируемую строку.

example_1. Экранирование символов

```
<?php

    $str = "Жанна д'Арк - национальная героиня Франции";

    // Жанна д\'Арк - национальная героиня Франции

    echo addslashes($str) . "<p>";

    $str = "Шиннед О'Коннор - ирландская певица и композитор";

    // Шиннед О\'Коннор - ирландская певица и композитор

    echo addslashes($str) . "<p>";

?>
```

stripslashes

stripslashes () — **удаляет** экранирование символов.

Описание

```
stripslashes($string);
```

Удаляет экранирование символов, произведенное функцией addslashes().

Параметр

- **string** – входная строка.

Функцию stripslashes() можно использовать, если экранирование символов не требуется. Например, данные не вставляются в базу данных, а просто **выводятся в браузер**.

Возвращаемые значения

Возвращает строку с **вырезанными** обратными слешами, (\' становится ' и т.п.) Двойные обратные слеша (\\) становятся одинарными (\).

example_2. Удаление экранирования

```
<?php

    $str = "Шарль Ожье де Бац де Кастельмор граф д'Артаньян -
    герой моего детства";

    // Шарль Ожье де Бац де Кастельмор граф д'Артаньян - герой
    моего детства

    echo stripslashes($str) . "<p>";

    $str = "It's my life (Bon Jovi)";

    $out = addslashes($str);

    // It's my life (Bon Jovi)

    echo stripslashes($out) . "<p>";

?>
```

htmlspecialchars

htmlspecialchars () — преобразует некоторые специальные символы в HTML-сущности.

Описание

htmlspecialchars(\$string);

Позволяет преобразовать HTML-теги в специальные сущности. Данная функция используется, главным образом, для преобразования данных, получаемых от пользователя через гостевую книгу, форму обратной связи и т.п. Функция **htmlspecialchars** **возвращает** строку, над которой проведены такие преобразования.

К символам подлежащим преобразованию относятся:

- & (амперсанд)
- < (меньше)
- > (больше)

- " (двойные кавычки)
- ' (одинарные кавычки)

Параметр

- **string** – конвертируемая строка.

Возвращаемые значения

Преобразованная строка **string**.

example_3. Тестируем htmlspecialchars

```
<?php

$str = "<h1>Жюль Амеде Барбе д'Оревилли — французский
писатель и публицист</h1>";

$out = htmlspecialchars(addslashes($str));

echo $out;

// вывод обычной ссылки

echo "<p><a href='https://vk.com/omsk_proger'
target='_blank'>Omsk-PROger</a><p>";

// вывод ссылки обработанной htmlspecialchars

echo htmlspecialchars("<a
href='https://vk.com/Omsk_proger'
target='_blank'>Omsk-PROger</a>");

?>
```

Если вам нужно преобразовать **все возможные сущности**, используйте `htmlentities()`.

htmlentities

htmlentities () — преобразует все возможные символы в соответствующие **HTML-сущности**.

Описание

```
htmlentities($string);
```

Эта функция идентична **htmlspecialchars()** за исключением того, что **htmlentities()** преобразует все символы в **соответствующие HTML-сущности** (для тех символов, для которых HTML-сущности существуют).

Если же вы хотите **раскодировать** строку (наоборот), используйте **html_entity_decode()**.

Параметр

- **string** – входная строка.

Возвращаемые значения

Возвращает преобразованную строку.

strip_tags

strip_tags () — удаляет теги HTML и PHP из строки.

Описание

```
strip_tags($string [, $allowed_tags]);
```

Функция пытается вернуть строку, из которой удалены все NULL-байты, HTML- и PHP-теги.

Параметр

- **string** - входная строка.
- **allowed_tags** - необязательный, может быть использован для указания тегов, которые не нужно удалять. Может указываться как строка (string) или как массив (array) с PHP 7.4.

Возвращаемые значения

Возвращает строку без тегов.

Важно. Из-за того, что **strip_tags()** не проверяет валидность HTML, то частичные или сломанные теги могут послужить удалением большего количества текста или данных, чем ожидалось.

Замечание. Имена тегов в HTML превышающие 1023 байта будут рассматриваться как не валидные независимо от параметра **allowed_tags**.

example_4. Инструкции желающим разбогатеть

```
<?php

// инструкции как стать богатым

$ul = "

    <ul>

    <li><a href='#>Как стать богатым и счастливым
человеком: Проверенные способы</a></li>

    <li><a href='#>10 законов богатства, которым следуют
все успешные миллиардеры</a></li>

    <li><a href='#>Проверенные способы, как
разбогатеть</a></li>

    <li><a href='#>Пошаговая инструкция: как стать богатым
и разбогатеть с нуля</a></li>

    <li><a href='#>Тренинги и упражнения, которые помогут
воспитать в себе миллионера</a></li>

    </ul>

";

echo "<h1>Мы подобрали ссылки специально для вас!</h1>";
```

```
// удалим теги <a>

echo strip_tags($ul, ["li", "ul"]);

// придется жить как жили ...

echo "<h2>Упс... неудачка вышла... ссылки удалились
:(</h2>";

?>
```

implode

implode () — объединяет элементы **массива в строку**.

Описание

```
implode($separator, $array);
```

Объединяет элементы массива **с помощью** строки **separator**.

Параметр

- **separator** – необязателен. По умолчанию равен **пустой строке**.
- **array** – массив **объединяемых** строк.

Возвращаемые значения

Возвращает строку, полученную объединением **элементов массива**, со вставкой строки между соседними элементами. Функция имеет псевдоним **join()**.

example_5. Объединяем элементы массива в строку

```
<?php

$array = array('php', 'javascript', 'css', 'html');

$str = implode(" #", $array);

// изучаем Web-технологии: #php #javascript #css #html

echo "Изучаем Web-технологии: #" . $str;
```

```
?>
```

explode

explode () — разбивает **строку в массив** с помощью разделителя.

Описание

```
explode($separator, $string [, $limit]);
```

Возвращает массив строк, полученных разбиением строки **string** с использованием **separator** в качестве **разделителя**.

Параметр

- **separator** – разделитель.
- **string** – входная строка.
- **limit** – необязательный,
 - если аргумент **limit** является положительным, возвращаемый массив будет содержать максимум **limit** элементов, при этом последний элемент будет содержать остаток строки **string**.
 - если параметр **limit** отрицателен, то будут возвращены все компоненты, кроме последних **-limit**.
 - если **limit** равен нулю, то он расценивается как 1.

Возвращаемые значения

Возвращает массив строк, созданный делением параметра string **по границам**, указанным параметром **separator**.

example_6. Разбиваем строку в массив

```
<?php

$str = "Изучаем Web-технологии: #laravel #php #java #sql
#python";

echo "<h3>$str</h3>";
```



```
$web = explode("#", $str);

// echo $web[0] . "<br />"; // Изучаем Web-технологии:

echo $web[1] . "<br />"; // laravel

echo $web[2] . "<br />"; // php

echo $web[3] . "<br />"; // java

echo $web[4] . "<br />"; // sql

echo $web[5]; // python

// echo "<pre>";

// print_r($web);

?>
```

trim

trim () — **удаляет пробелы** из начала и конца строки.

Описание

```
trim($string, $characters);
```

Функция **возвращает строку** с удаленными из начала и конца строки пробелами. Если второй параметр не передан, функция удаляет следующие символы (список не полон):

- " " - символ пробела.
- "\t" - символ табуляции.
- "\n" - символ перевода строки.

Параметр

- **string** – обрезаемая строка.
- **characters** – заданный **список символов для удаления**.

Возвращаемые значения

Обрезаемая строка.

example_7. Тестируем trim

```
<?php

    // в строках присутствуют пробелы

    $veni = "    Veni    ";
    $vidi = "    Vidi    ";
    $vici = "    Vici    ";

    echo $veni . $vidi . $vici;

    echo "<p>";

    echo trim($veni) . trim($vidi) . trim($vici);

?>
```

ltrim, rtrim

ltrim (), **rtrim ()** — удаляет пробелы из начала или конца строки.

Описание

```
ltrim($string, $characters);
```

```
rtrim($string, $characters);
```

ltrim — удаляет пробелы из начала строки.

rtrim — удаляет пробелы из конца строки.

Удаляет пробелы (или другие символы) из начала или конца строки.

Параметр

- **string** – входная строка.
- **characters** – заданный **список символов** для удаления.

Возвращаемые значения

Эти функции возвращают строку с удаленными из начала / конца строки пробелами. Если второй параметр не передан, удаляют те же символы что в trim()).

example_8. Тестируем rtrim

```
<?php

$str = "Изучаем Web-технологии: #laravel #php #java #sql
#python";

echo "<h3>$str</h3>";

$web = explode("#", $str);

echo rtrim($web[0]) . "<br />"; // Изучаем Web-технологии:
echo rtrim($web[1]) . "<br />"; // laravel
echo rtrim($web[2]) . "<br />"; // php
echo rtrim($web[3]) . "<br />"; // java
echo rtrim($web[4]) . "<br />"; // sql
echo rtrim($web[5]); // python

?>
```

strlen

strlen () — возвращает **длину** строки.

Описание

```
strlen($string);
```

Возвращает **длину** строки string.

Параметр

- **string** – строка, для которой изменяется длина.

Возвращаемые значения

Длина строки `string` в случае успешного выполнения, и **0**, если **`string`** пуста.

Важно. Функция `strlen` возвратит **количество байт**, а не число символов в строке.

```
<?php

$str = 'abcdef';

echo strlen($str); // 6

$str = ' ab cd ef ';

echo strlen($str); // 10
```

strpos

strpos () — возвращает позицию первого вхождения подстроки.

Описание

```
strpos ($string, $findme [, $offset]);
```

Ищет позицию первого вхождения подстроки **findme** в строку **string**.

Параметр

- **string** – строка в которой производится поиск.
- **findme** – искомая строка.
- **offset** – необязательный. Если указан, то поиск будет начат с указанного количества символов с начала строки. Если задано отрицательное значение, отсчёт позиции начала поиска будет произведён с конца строки.

Возвращаемые значения

Возвращает **позицию**, в которой находится искомая строка, относительно начала строки **string** (независимо от смещения **offset**). Возвращает **false**, если искомая строка не найдена.

Важно. Обратите внимание на то, что позиция строки отсчитывается от 0, а не от 1.

example_9. Тестируем strpos

```
<?php

// входная строка
$string = "#javascript #php #python #sql";

// искомая подстрока
$substr = "php";

$search = strpos($string, $substr);

if($search){

    echo "Совпадение найдено";

} else echo "Совпадений нет";

echo "<pre>";

var_dump($search);

?>
```

substr

substr () — возвращает подстроку.

Описание

substr(\$string, \$offset [, \$length])

Возвращает подстроку строки **string**, начинающейся с **offset** символа по счету и длиной **length** символов.

Параметр

- **string** – входная строка.
- **offset** –
 - если **offset** неотрицателен, возвращаемая подстрока начинается с позиции **offset** от начала строки, считая от нуля. Например, в строке 'abcdef', в позиции 0 находится символ 'a', в позиции 2 - символ 'c', и т.д.
 - если **offset** отрицательный, возвращаемая подстрока начинается с позиции, отстоящей на **offset** символов от конца строки **string**.
 - если **string** меньше **offset** символов, будет возвращена пустая строка.
- **length** – необязательный,
 - если **length** положительный, возвращаемая строка будет не длиннее **length** символов, начиная с параметра **offset** (в зависимости от длины **string**).
 - если **length** отрицательный, то будет отброшено указанное этим аргументом число символов с конца строки **string**.
 - если параметр **length** задан и равен 0, будет возвращена пустая строка.
 - если параметр **length** опущен или **null**, то будет возвращена подстрока, начинающаяся с позиции, указанной параметром **offset** и длящейся до конца строки.

Возвращаемые значения

Возвращает извлечённую часть параметра **string** или пустую строку.

example_10. Пример использования substr

```
<?php
echo substr("abcdef", 1), "<br />"; // возвращает "bcdef"
echo substr("abcdef", 1, 3), "<br />"; // возвращает "bcd"
```

```
echo substr("abcdef", 0, 4), "<br />"; // возвращает "abcd"

echo substr("abcdef", 0, 8), "<br />"; // возвращает
"abcdef"

echo substr("abcdef", -1), "<br />"; // возвращает "f"

echo substr("abcdef", -2), "<br />"; // возвращает "ef"

echo substr("abcdef", -3, 1), "<br />"; // возвращает "d"

echo substr("abcdef", 0, -1), "<br />"; // возвращает
"abcde"

echo substr("abcdef", 2, -1), "<br />"; // возвращает "cde"

echo substr("abcdef", 4, -4), "<br />"; // возвращает ""

echo substr("abcdef", -3, -1), "<br />"; // возвращает "de"

echo "<hr>";

// к отдельным символам можно обращаться с помощью фигурных
скобок

$string = 'abcdef';

echo $string[0], "<br />"; // выводит а

echo $string[3]; // выводит d

?>
```

substr_replace

substr_replace () — **заменяет** часть строки.

Описание

substr_replace(\$string, \$replace, \$offset [, \$length])

substr_replace() заменяет **часть строки string**, начинающуюся с символа с порядковым номером **offset** и (необязательной) длиной **length**, строкой **replace** и возвращает результат.

Параметр

- **string** – входная строка.
- **replace** – строка замены.
- **offset** –
 - если **offset** **положителен**, замена начинается с символа с порядковым номером **offset** строки **string**.
 - если **offset** **отрицателен**, замена начинается с символа с порядковым номером **offset**, считая от конца строки **string**.
- **length** – необязательный, по умолчанию равен **strlen(string)**,
 - если аргумент положителен, то он представляет собой длину заменяемой подстроки в строке **string**.
 - если этот аргумент отрицательный, он определяет количество символов от конца строки **string**, на которых заканчивается замена.
 - если **length** равен нулю, то это эквивалентно **вставке replace** в **string** на указанной позиции **offset**.

Возвращаемые значения

Заменяет часть строки **string** начинающуюся с символа с порядковым номером **start** и длиной **length**, строкой **replace** и **возвращает результат**.

example_11. Простой пример использования substr_replace

```
<?php

$var = 'ABCDE: / 12345 /';

echo "Оригинал: $var<hr />";

// обе следующих строки заменяют всю строку $var на
// значение второго параметра

echo substr_replace($var, 'abc', 0) . "<br />";

echo substr_replace($var, '123', 0, strlen($var)) .
"<br />";

// вставляет 'abc' в начало $var
```



```
echo substr_replace($var, 'abc', 0, 0) . "<br />";  
  
// обе следующих строки заменяют '12345' на 'abc'  
  
echo substr_replace($var, 'abc', 9, -2) . "<br />";  
  
echo substr_replace($var, 'abc', -7, -2) . "<br />";  
  
// удаляет ' 12345 ' из $var  
  
echo substr_replace($var, '', 9, -1) . "<br />";  
  
?>
```

Функции форматного вывода

sprintf

sprintf () — возвращает **отформатированную** строку.

Описание

```
sprintf($format, $values);
```

Возвращает строку, созданную с использованием строки формата **format**.

Параметр

- **format** – **обычные символы**, которые просто **выводятся без изменения и спецификаторы преобразования**, каждый из которых требует передачи своего параметра.
- **values** – подставляемые значения.

Спецификатор преобразования в самом простом случае имеет формат: **%specifier**.

где **specifier** может быть:

- **s** - аргумент рассматривается и печатается как строка (%s).

- **d** - аргумент рассматривается как целое число и печатается как целое число со знаком (%d).
- **f** - аргумент рассматривается как число с плавающей точкой (%f).
- ...

Например:

```
<?php
    $msg = sprintf("Добрый день, %s!", "Евгений");
    echo $msg; // Добрый день, Евгений!
```

Возвращаемые значения

Возвращает строку, **отформатированную** в соответствии с форматом **format**.

example_12. Функция sprintf

```
<?php
    // иницилируем массив
    $albums = [ ... ];
    $html = "";
    // перебираем массив
    foreach ($albums as $key => $item) {
        // формируем строку для вывода
        $html .= sprintf(
            "Номер - %d<br />
            ID альбома: %d<br />
            Название: %s<br />
            Дата выпуска: %s<br />
            Лейбл: %s<br />
            Формат: %s<br />
```

```
        Статус: %s<p>

        <hr />

        ",
        ++$key,
        $item['id'],
        $item['album_name'],
        $item['date'],
        $item['label'],
        $item['format'],
        $item['status']
    );
};

// выводим строку в браузер
echo $html;

?>
```

printf

printf () — выводит **отформатированную** строку.

Описание

```
printf($format, $values);
```

Выводит строку, **отформатированную** в соответствии с форматом **format**.

Параметр

- **format** – **обычные символы**, которые просто **выводятся без изменения и спецификаторы преобразования**, каждый из которых требует передачи своего параметра.

- **values** – подставляемые значения.

Спецификатор преобразования в самом простом случае имеет формат: **%specifier**.

где **specifier** может быть:

- **s** - аргумент рассматривается и печатается как строка (%s).
- **d** - аргумент рассматривается как целое число и печатается как целое число со знаком (%d).
- **f** - аргумент рассматривается как число с плавающей точкой (%f).
- ...

Например:

```
<?php
$name = "Татьяна"; // строковая переменная
$age = 16; // числовая переменная
// Добрый день, Татьяна! Вам больше 16 лет?
printf("Добрый день, %s! Вам больше %d лет?", $name, $age);
```

Возвращаемые значения

Возвращает длину выводимой строки.

example_13. Функция printf

```
<?php
// иницилируем массив
$albums = [ ... ];
// перебираем массив
foreach ($albums as $key => $item) {
    // выводим строку в браузер
    printf("
```

```

        Номер - %d<br />

        ID альбома: %d<br />

        Название: %s<br />

        Дата выпуска: %s<br />

        Лейбл: %s<br />

        Формат: %s<br />

        Статус: %s<p>

        <hr />

    ",
    ++$key,
    $item['id'],
    $item['album_name'],
    $item['date'],
    $item['label'],
    $item['format'],
    $item['status']
);

};

?>

```

Функции **sprintf** и **printf** чрезвычайно удобны при выводе форматированных строк и часто используются разработчиками.

Кроме того, иногда бывает удобно перед подстановкой переменных в шаблон выполнить над ними некоторые действия.

Например:

```

<?php
    printf(
        "Название: %s<br />",

```

```
    htmlspecialchars($item['title'])  
);
```

Задание 47

=====

В директории **dump** вам предоставлены файлы с массивами групп (**teams.php**), альбомов (**albums.php**), треков (**tracks.php**). Выполните вывод в браузер данных, согласно переданного в строке запроса **GET-параметра** вида **?search=entity::id**,

где:

- **entity** - сущность (массив),
- **id** - идентификатор записи сущности.

Например:

- ?search=teams::3
- ?search=albums::5
- ?search=tracks::10

Формат вывода - любой.

Задание 48

=====

В раздаточном файле **index.php** представлен скрипт вывода двумерного ассоциативного массива **\$albums**. В некоторых ключах массива хранятся значения, содержащие HTML-теги списка.

Ревьюируйте код скрипта:

- при выводе в браузер данных массива **\$albums** избавьтесь от HTML-тегов списка;
- для вывода используйте любую из функций **форматного вывода** (**printf** / **sprintf**).

P.S. Так как в задании не указывается, каким именно способом освободиться от тегов списка, решением задачи могут быть следующие действия:

- **преобразование** тегов списка в HTML-сущности, функция **htmlspecialchars** (index_1.php);
- **удаление** тегов списка, функция **strip_tags** (index_2.php).

Задание 49

=====

В раздаточном файле **index.php** организован вывод массива **\$albums** с использованием цикла **foreach**. Ревьюируйте код таким образом, чтобы значения вложенных массивов выводились в виде **строковых литералов**.

Для решения задачи используйте функцию **implode**.

Задание 50

=====

Ревьюируйте код файла **index.php** таким образом, чтобы строка вывода **статусов** альбома выводилась в виде маркированного списка. Для решения задачи используйте функцию **explode()**.

Образец вывода:

The Wall (id=7)

Дата выпуска: 30 ноября 1979

Лейбл: Harvest, EMI, Columbia, Capitol

Формат: LP, 8-track, кассета, CD

Статус:

- Платиновый (USA)
- Платиновый (GBR)
- Бриллиантовый (CAN)

Встроенные функции (3)

Функции для работы с массивами

count

count () — подсчитывает **количество элементов** в массиве.

Описание

Подсчитывает все элементы в массиве.

```
count($array [, $mode]);
```

Параметр

- **array** – массив.
- **mode** – необязательный параметр. Если mode установлен в **true**, count() будет рекурсивно подсчитывать количество элементов многомерного массива.

Возвращаемые значения

Возвращает количество элементов в массиве **array**.

```
<?php  
  
$arr[0] = 1;  
  
$arr[1] = 3;  
  
$arr[2] = 5;  
  
var_dump(count($arr)); // 3
```

array_key_exists

array_key_exists () — проверяет, присутствует ли в массиве указанный **ключ** или **индекс**.

Описание

```
array_key_exists($key, $array);
```

Функция `array_key_exists()` возвращает **true**, если в массиве **присутствует** указанный ключ **key**. Параметр **key** может быть любым значением, которое подходит для индекса массива.

Параметр

- **key** – проверяемое значение.
- **array** – массив с проверяемыми ключами.

Возвращаемые значения

Возвращает **true** в случае успешного выполнения или **false** в случае возникновения ошибки.

Замечание: `array_key_exists()` ищет ключи только на первом уровне массива. Внутренние ключи в многомерных массивах найдены не будут.

example_1. Поиск ключа в массиве

```
<?php

$person = array( ... );

// проверим наличие ключа site в массиве

if (array_key_exists("site", $person)) {

    echo "В массиве найден ключ `site`";

};

?>
```

in_array

`in_array ()` — проверяет, присутствует ли в массиве значение.

Описание

```
in_array($needle, $haystack, $strict = false);
```

Ищет в **haystack** значение **needle**. Если **strict** не установлен, то при поиске будет использовано нестрогое сравнение.

Параметр

- **needle** – искомое значение. Если needle - строка, сравнение будет произведено с учётом регистра.
- **haystack** – массив.
- **strict** – если третий параметр strict установлен в true, тогда функция также проверит соответствие типов параметра needle и соответствующего значения массива haystack.

Возвращаемые значения

Возвращает **true**, если needle был найден в массиве, и **false** в противном случае.

example_2. Поиск значения в массиве

```
<?php

$person = array('Бородина', 'Ксения', 'Алексеевна',
    'Преподаватель', 'Соответствие занимаемой должности',
    'Высшее профессиональное');

// критерий поиска

$value = "Преподаватель";

// результат поиска

var_dump(in_array($value, $person));

?>
```

array_search

array_search() — осуществляет поиск указанного значения в массиве и возвращает ключ первого найденного элемента в случае успешного выполнения.

Описание

```
array_search($needle, $haystack, $strict = false);
```

Ищет в **haystack** значение **needle**.

Параметры

- **needle** – искомое значение. Если needle является строкой, сравнение происходит с учётом регистра.
- **haystack** – массив.
- **strict** – если третий параметр strict установлен в true, то функция array_search() будет искать идентичные элементы в haystack. Это означает, что также будут проверяться типы needle в haystack, а объекты должны быть одним и тем же экземпляром.

Возвращаемые значения

Возвращает **ключ** для **needle**, если он был найден в массиве, иначе **false**.

example_3. Поиск значения в массиве и возврат ключа

```
<?php

$person = array( ... );

// проверим наличие значения в массиве

if ($key = array_search("Преподаватели", $person)) {

    echo "В массиве найден ключ $key";

};

?>
```

array_map

array_map () — применяет **callback**-функцию ко всем элементам указанных массивов.

К сведению. Данную функцию подробно рассмотрим в "Анонимные функции".

Описание

```
array_map(callback, $array);
```

Возвращает массив, содержащий элементы всех указанных массивов после их **обработки функцией обратного вызова**. Количество параметров, передаваемых функции обратного вызова, должно совпадать с количеством массивов, переданным функции **array_map()**.

Параметр

- **callback** – функция, применяемая к каждому элементу в каждом массиве.
- **array** – массив, к каждому элементу которого применяется функция.

Возвращаемые значения

Возвращает массив, содержащий результаты применения **функции обратного вызова** к соответствующему элементу **массива**, используемого в качестве аргумента для **функции обратного вызова**.

array_push

array_push () — добавит один или несколько элементов в **конец массива**.

Описание

```
array_push($array, $values);
```

Использует **array** как стек, и добавляет переданные значения в конец массива **array**. Длина **array** увеличивается на количество переданных значений.

Параметр

- **array** – входной массив.
- **values** – значения, добавляемые в конец массива **array**.

Возвращаемые значения

Возвращает новое количество элементов в массиве.

Замечание. Вместо использования **array_push()** для добавления в массив одного элемента, лучше использовать `$array[] = ...`, потому что в этом случае не происходит вызова функции.

example_4. Добавление элемента в конец массива

```
<?php

    $tags = "#javascript #php #python #sql";

    // разбиваем строку в массив

    $arr = explode(" ", $tags);

    echo "<pre>";

    var_dump($arr);

    // добавим элементы в конец массива

    array_push($arr, "#perl", "#mysql");

    echo "<pre>";

    var_dump($arr);

?>
```


array_unshift

`array_unshift()` — добавит один или несколько элементов в **начало массива**.

Описание

```
array_unshift($array, $values);
```

Добавляет переданные в качестве аргументов элементы в начало массива **array**.

Параметр

- **array** – входной массив.
- **values** – значения, добавляемые в начало массива **array**.

Обратите внимание, что список элементов добавляется целиком, то есть порядок элементов сохраняется. Все числовые ключи будут изменены таким образом, что нумерация массива будет начинаться с нуля, в то время как строковые ключи останутся прежними.

Возвращаемые значения

Возвращает новое количество элементов в **array**.

example_5. Добавление элемента в начало массива

```
<?php

$tags = "#javascript #php #python #sql";

// разбиваем строку в массив

$arr = explode(" ", $tags);

echo "<pre>";

var_dump($arr);

// добавляем элементы в начало массива

array_unshift($arr, "#perl", "#mysql");
```

```
echo "<pre>";

var_dump($arr);

?>
```

array_shift

array_shift () — **извлекает первый элемент** массива.

Описание

```
array_shift($array);
```

Извлекает первое значение массива **array** и возвращает его, сокращая размер **array** на один элемент. Все числовые ключи будут изменены таким образом, что нумерация массива начнётся с нуля, в то время как строковые ключи останутся прежними.

Параметр

- **array** – входной массив.

Возвращаемые значения

Возвращает извлекаемое значение или **null**, если **array** пуст или не является массивом.

example_6. Извлечение первого элемента массива

```
<?php

$tags = "#javascript #php #python #sql";

// разбиваем строку в массив

$arr = explode(" ", $tags);

// добавляем элементы в начало массива

array_unshift($arr, "#perl", "#mysql");

echo "<pre>";
```

```
var_dump($arr);

echo "Первый элемент массива - ";

echo array_shift($arr);

?>
```

array_pop

array_pop () — извлекает последний элемент массива.

Описание

Извлекает и возвращает последнее значение параметра **array**, уменьшая размер **array** на один элемент.

```
array_pop($array);
```

Параметр

- **array** – входной массив.

Возвращаемые значения

Возвращает значение последнего элемента массива **array**.

Если **array** пуст (или не является массивом), будет возвращён **null**.

example_7. Извлечение последнего элемента массива

```
<?php

$tags = "#javascript #php #python #sql";

// разбиваем строку в массив

$arr = explode(" ", $tags);

// добавляем элементы в конец массива

array_push($arr, "#perl", "#mysql");

echo "<pre>";
```

```
var_dump($arr);  
  
echo "Последний элемент массива - ";  
  
echo array_pop($arr);  
  
?>
```

array_merge

array_merge () — объединяет один или большее количество массивов.

Описание

```
array_merge(...$arrays);
```

Объединяет элементы одного или большего количества массивов таким образом, что значения одного массива присоединяются к концу предыдущего. Результатом работы функции является новый массив.

Если входные массивы имеют одинаковые строковые ключи, тогда каждое последующее значение будет заменять предыдущее. Однако, если массивы имеют одинаковые числовые ключи, значение, упомянутое последним, не заменит исходное значение, а будет добавлено в конец массива.

В результирующем массиве значения исходного массива с числовыми ключами будут перенумерованы в возрастающем порядке, начиная с нуля.

Параметр

- **arrays** - объединяемые массивы.

Возвращаемые значения

Возвращает **результатирующий массив**. Если вызывается без аргументов, возвращает **пустой массив**.

example_8. Объединение массивов

```
<?php

    // массив 1

    $arr1 = ["javascript", "css", "bootstrap", "figma"];

    // массив 2

    $arr2 = ["php", "python", "sql", "mysql"];

    // объединяем массивы

    $arr_merge = array_merge($arr2, $arr1);

    echo "<pre>";

    var_dump($arr_merge);

    // выведем массив строкой

    $tags = implode(" #", $arr_merge);

    echo "#" . $tags;

?>
```

current

current () — возвращает **текущий элемент** массива.

Описание

```
current($array);
```

У каждого массива имеется **внутренний указатель** на его **текущий элемент**, который инициализируется первым элементом, добавленным в массив.

Параметр

- **array** – массив.

Возвращаемые значения

Функция возвращает **значение элемента массива**, на который указывает его **внутренний указатель**. Если внутренний указатель находится за пределами списка элементов или массив пуст, возвращает **false**.

example_9. Текущий элемент массива

```
<?php

    $tags = "#javascript #php #python #sql #perl #mysql";

    $arr = explode(" ", $tags);

    echo "<pre>";

    var_dump($arr);

    echo "Текущий элемент массива - ", current($arr);

?>
```

next

next () — **передвигает внутренний указатель** массива на одну позицию **вперёд**.

Описание

next (\$array) ;

Ведёт себя подобно **current()**, но с одним отличием. Перед тем, как вернуть значение элемента массива, эта функция передвигает его внутренний указатель на одну позицию вперёд. Другими словами, она **возвращает следующий элемент массива и сдвигает его внутренний указатель на одну позицию**.

Параметр

- **array** – массив, изменяемый данной функцией.

Возвращаемые значения

Возвращает значение элемента массива, находящегося на позиции, следующей за позицией, в которой находится его внутренний указатель или **false**, если достигнут конец массива.

example_10. Перемещение внутреннего указателя вперед

```
<?php

    $tags = "#javascript #php #python #sql #perl #mysql";

    $arr = explode(" ", $tags);

    echo "<pre>";

    var_dump($arr);

    next($arr);

    echo "Текущий указатель - ", current($arr), "<br />";

    next($arr);

    echo "Текущий указатель - ", current($arr), "<br />";

?>
```

prev

prev () — **передвигает внутренний указатель** массива на одну позицию **назад**.

Описание

prev(\$array) ;

Ведёт себя подобно next(), за исключением того, что она передвигает внутренний указатель массива на одну позицию назад, а не вперёд.

Параметр

- **array** – входной массив.

Возвращаемые значения

Возвращает значение элемента массива, находящегося на позиции, предыдущей по отношению к позиции, в которой находится его внутренний указатель или **false**, если достигнут конец массива.

Внимание

Если массив содержит пустые или равные 0 элементы, функция возвратит **false** для этих элементов.

example_11. Перемещение внутреннего указателя назад

```
<?php

    $tags = "#javascript #php #python #sql #perl #mysql";

    $arr = explode(" ", $tags);

    echo "<pre>";

    var_dump($arr);

    next($arr);

    echo "Текущий указатель - ", current($arr), "<br />";

    prev($arr);

    echo "Текущий указатель - ", current($arr), "<br />";

?>
```

end

end () — устанавливает внутренний указатель массива на его **последний** элемент.

Описание

```
end($array);
```

Устанавливает внутренний указатель **array** на последний элемент и возвращает его значение.

Параметр

- **array** – массив.

Возвращаемые значения

Возвращает значение последнего элемента или **false** для пустого массива.

```
<?php  
  
$arr = array();  
  
$arr[1] = 1;  
  
$arr[0] = 0;  
  
echo end($a); // 1
```

key

key () — выбирает ключ из массива.

Описание

Возвращает индекс текущего элемента массива.

```
key($array);
```

Параметр

- **array** – массив.

Возвращаемые значения

Функция возвращает ключ того элемента массива, на который в данный момент указывает внутренний указатель массива. Она не сдвигает указатель ни в каком направлении. Если внутренний указатель указывает вне границ массива или массив пуст, возвратит **null**.

reset

reset () — устанавливает внутренний указатель массива на его **первый** элемент.

Описание

```
reset ($array);
```

Перемещает внутренний указатель массива **array** к его первому элементу и возвращает значение **первого элемента** массива.

Параметр

- **array** – массив.

Возвращаемые значения

Возвращает значение **первого элемента** массива, или **false**, если массив пуст.

example_12. Установка указателя на первый элемент

```
<?php

$web = array("javascript", "php", "ruby", "angular",
"mysql");

// сместим указатель в конец массива

end($web);

// выведем индекс текущего элемента

echo "Последний индекс - ", key($web), "<p>";

// сбросим указатель в начало массива

reset($web);

// выведем индекс текущего элемента

echo "Первый индекс - ", key($web);

?>
```

sort

sort () — **сортирует** массив по возрастанию.

Описание

```
sort($array);
```

Функция сортирует массив. После завершения работы функции элементы массива будут расположены в порядке возрастания.

Параметр

- **array** – входной массив.

Возвращаемые значения

Возвращает **true** в случае успешного завершения или **false** в случае возникновения ошибки.

Замечание. Функция назначает новые ключи для элементов **array**. Все ранее назначенные значения ключей будут удалены, вернее переназначены. Если оба сравниваемых значения эквивалентны, они сохраняют свой первоначальный порядок.

Замечание. Сбрасывает внутренний указатель массива на первый элемент.

example_13. Сортировка массива

```
<?php

echo "<h3>Исходный массив</h3>";

$web = array("javascript", "php", "ruby", "angular",
"mysql");

echo "<pre>";
```

```
var_dump($web);  
  
echo "</pre>";  
  
echo "<h3>Отсортированный массив</h3>";  
  
sort($web);  
  
echo "<pre>";  
  
var_dump($web);  
  
?>
```

rsort

rsort() — **сортирует** массив в обратном порядке (r - reverse).

Описание

Эта функция сортирует массив в обратном порядке (от большего к меньшему).

```
rsort($array);
```

Параметр

- **array** – входной массив.

Возвращаемые значения

Возвращает **true** в случае успешного завершения или **false** в случае возникновения ошибки.

ksort

ksort() — сортирует массив **по ключу** в порядке возрастания (k - key).

Описание

```
ksort($array);
```

Сортирует массив по ключам в порядке возрастания, сохраняя отношения между ключами и значениями. Функция полезна, в основном, для работы с ассоциативными массивами.

Параметр

- **array** – входной массив.

Возвращаемые значения

Возвращает **true** в случае успешного завершения или **false** в случае возникновения ошибки.

example_14. Сортировка массива по ключу

```
<?php

$person = array(

    'id_personnel' => 3,

    'surname' => 'Рыбкина',

    'name' => 'Ольга',

    'patronymic' => 'Витальевна',

    'post' => 'Преподаватель',

    ...

);

echo "<h3>Исходный массив</h3>";

echo "<pre>";

var_dump($person);

echo "</pre>";

ksort($person);

echo "<h3>Отсортированный по ключу массив</h3>";

echo "<pre>";

var_dump($person);
```

```
echo "</pre>";
```

```
?>
```

krsort

krsort () — сортирует массив **по ключу** в обратном порядке (k - key, r - reverse).

Описание

```
krsort($array);
```

Сортирует массив по ключам в обратном порядке, сохраняя отношения между ключами и значениями. Функция полезна, в основном, для работы с ассоциативными массивами.

Параметр

- **array** - входной массив.

Возвращаемые значения

Возвращает **true** в случае успешного завершения или **false** в случае возникновения ошибки.

Задание 51

=====

В раздаточном файле **person.php** вам предложен массив **person**.
Напишите скрипт, выполняющий проверку наличия в массиве ключа **category**. Если ключа нет, добавьте его в массив, со значением по умолчанию равным: "Соответствие занимаемой должности".

- Выведите данные о **категории** преподавателя.
- Используя конструкцию **var_dump**, выведите **массив** в браузер.

Задание 52

=====

В массиве **albums** файла **albums.php** в некоторых вложенных массивах отсутствуют ключи-идентификаторы записей **id**. Используя цикл **foreach**, переберите вложенные массивы, и если ключ-идентификатор отсутствует, добавьте новый. Значения ключам присвойте, используя правила простого счета (1,2,3,4...).

P.S. Задача несколько сложнее, чем кажется на первый взгляд. Цель - перебрать элементы массива, и внести поправки в случае необходимости. Но в конструкции:

```
foreach ($albums as $item){...}
```

изменения переменной **item** никак не сказывается на исходном массиве **albums**. Выход - изменяем **item** и записываем в новый массив.

Более элегантный способ - использование функции **array_map()**, рассматриваемой в "Анонимные функции".

Задание 53

=====

Ревьюируйте цикл **while** файла **index.php** таким образом, чтобы не использовать переменную цикла **i**.

Для организации алгоритма перебора значений массива примените функции:

- **current()** - проверка условия выхода из цикла.
- **next()** - перемещение внутреннего указателя массива.

Задание 54

=====

Ревьюируйте код файла **index.php** таким образом, чтобы не использовать переменную цикла **i**. Осуществите перебор значений массива от последнего к первому.

Для организации алгоритма перебора значений массива используйте:

- функцию **end()** - перемещение указателя в конец массива;
- цикл **do - while**;
- функцию **current()** - проверка условия выхода из цикла;
- функцию **prev()** - перемещение внутреннего указателя массива.

Задание 55

=====

Ревьюируйте код файла **index.php** таким образом, чтобы не использовать переменную цикла **i**. При этом для решения задачи используйте:

- цикл **for()**;
- функцию **current()** - проверка условия выхода из цикла;
- функцию **next()** - перемещение внутреннего указателя массива.

P.S. Аргументы цикла **for()** - **не являются** обязательными. Цикл может работать без них, при этом алгоритм перебора определяется в теле цикла разработчиком.

Пользовательские функции

- Введение
- Создание (определение) функции
- Передача параметров
- Необязательные параметры
- Возврат значений
- В качестве эпилога

Введение

Помимо встроенных функций, в PHP часто возникает необходимость создания **пользовательских** функций, выполняющих определенные задачи по реализации функционала проектируемых систем.

В любом языке программирования существуют пользовательские функции. **Функция** - это специальным образом оформленный **именованный** фрагмент кода, к которому можно **многократно обратиться из любого места внутри программы**.

Важно.

- **Функция не будет** выполняться сразу при загрузке страницы.
- **Функция будет** выполняться при ее **вызове** из тела программы.

Некоторые особенности пользовательских функций в PHP:

- доступны параметры по умолчанию. Есть возможность вызывать одну и ту же функцию с переменным числом параметров;
- пользовательские функции могут возвращать любой тип;
- есть возможность изменять переменные, переданные в качестве аргумента.

Создание (определение) функции

Код, в котором создаётся функция, называется **определением функции**.

Вот шаблон определения простой функции:

```
<?php
    function имяФункции()
    {
        // код функции (еще говорят - тело функции)
    }
```

Важно. Имена функций **не чувствительны** к регистру.

Место в программе, где упоминается имя функции, называется **вызовом функции**.

Так как функцию часто называют **подпрограммой**, то программный код, откуда вызывается функция, называют **внешней** (или говорят – вызывающей) **программой**.

```
<?php
    // инструкции внешней программы
    // вызов ранее определенной функции (подпрограммы)
    имяФункции();
    // ...
```

Определение собственных функций значительно упрощает написание и поддержку программ. Функции позволяют объединять сложные (составные) операции в блок кода, который может располагаться вне основной логики программы. В таком **случае файл с функцией (функциями) необходимо подключить**.

Например, отправка запроса на аутентификацию пользователя по логину и паролю - это достаточно сложный процесс, включающий в себя

взаимодействие с внешними системами. Но благодаря возможности определять функции, вся сложность может быть скрыта за простой функцией:

```
// один маленький вызов — и много логики внутри
// сколько за этим бессонных ночей знает только разработчик
authorize($login, $password);
```

Ниже приведен пример определения и вызова простой функции приветствия.

example_1. Пример функции

```
<?php

    // определение (создание) функции

    function outGreeting() {

        echo "<h2>Ave, Caesar, morituri te salutant!</h2>";

    }

    // вызов функции

    outGreeting();

?>
```

Передача параметров

Функции без параметров встречаются редко. С помощью параметров мы можем передавать в функцию некоторые данные. Функции принимают эти данные, как-то их используют и возвращают результат обратно **внешней** программе.

Важно. Информация может передаваться функциям через **параметры**. **Параметр** — это просто переменная.

Параметры определяются в скобках после названия функции как обычные переменные, отделенные друг от друга запятой.

Передача одного параметра

Параметры можно называть как угодно, их имена имеют смысл исключительно в теле функции. При наименовании параметров применяются те же правила, что и для переменных.

Важно. Аргументы часто путают с параметрами. Предлагаю пользоваться следующим определением:

- **Параметр** — **переменная**, создаваемая при определении функции.
- **Аргумент** — **значение**, которое передается в функцию при её вызове.

В дальнейшем при вызове функции, ее параметру необходимо передать некоторое значение (аргумент), **сколько функция определяет параметров, столько необходимо передать аргументов** (значений). Внутри самой функции мы можем использовать аргумент так же, как обычные переменные.

Важно. Если мы не передадим значение для параметра - столкнемся с ошибкой.

Познакомимся с определением и вызовом функции, принимающей на вход один параметр.

example_2. Функция строкового параметра

```
<?php  
  
    // определение (создание) функции
```

```

// $name - параметр

function outMessage($name) {

    echo "<h3>$name, сдаётся мне, твой друг хочет обидеть
    нас!</h3>";

}

// вызов функции

outMessage("Билли"); // аргумент - Билли

outMessage("Винни-Пух"); // аргумент - Винни-Пух

outMessage("Александр Иванович"); // аргумент - Александр
Иванович

outMessage("Имярек"); // аргумент - Имярек

?>

```

Еще один пример использования функции одного параметра в расчете площади круга:

example_3. Функция числового параметра

```

<?php

// определение функции

function circleArea($radius) {

    // определим константу числа пи

    define("PI", 3.1415);

    $result = PI * $radius * $radius;

    // выведем результат в браузер

    echo $result;

}

// вызов функции

circleArea(5);

?>

```


Совет. В одних языках программирования принято использовать термины **параметры - аргументы**, в других языках **формальные - фактические параметры**. Не стоит, особенно на первых порах, придавать этому большое значение. А вот научиться использовать нечто, передаваемое в скобках (;-)) - **ОЧЕНЬ ВАЖНО**.

Передача нескольких параметров

Вы можете добавить **любое количество параметров**, просто разделив их запятой.

example_4. Функция двух параметров

```
<?php

// определение (создание) функции

function outCitate($name, $patronymic) {

    echo "<h3>Мне странно, $name $patronymic: вы, кажется,
    человек, известный ученостью, а говорите, как
    недоросль...</h3>";

}

// вызов функции

outCitate("Иван", "Иванович");

outCitate("Иван", "Никифорович");

?>
```

Здесь функция outCitate определяет **два параметра**, соответственно при вызове функции нам надо передать в функцию **два значения**.

Значения отделяются запятой и передаются параметрам **по позиции**. Это значит, что первое значение передается первому параметру, второе значение передается второму параметру и так далее.

Важно. В каком порядке параметры стоят при определении функции, **в таком же порядке** они должны передаваться при вызове.

Передача параметров переменной длины

PHP поддерживает списки **параметров переменной длины** для функций, определяемых пользователем с помощью добавления многоточия (...).

Аргументы в этом случае в теле функции будут доступны в **виде массива**.

Например:

example_5. Функция параметров переменной длины

```
<?php

// при определении функции говорим, что
// количество аргументов заранее неизвестно

function outHobby(...$args) {

    // проверяем тип данных и значения аргумента

    echo "<pre>";

    var_dump($args);

    echo "</pre>";

    // выведем аргумент-массив в цикле

    foreach ($args as $value){

        echo "<li> $value </li>";

    }

};

// вызов функции

outHobby("Спорт", "Программирование", "Censored",
```

```
"Censored");
```

```
?>
```

Передача массива

Так же как скалярная переменная (или любое количество скалярных переменных) на вход функции может быть подана переменная **типа массив**. Никаких проблем с обработкой массива в теле функции не будет.

example_6. Функция с параметром типа массив

```
<?php

// передаем на вход некий массив
// пусть имя параметра-массива будет $arr

function outHobby($arr) {

    // проверяем тип данных и значения аргумента
    echo "<pre>";

    var_dump($arr);

    echo "</pre>";

    // выведем массив-аргумент в цикле
    foreach ($arr as $key => $val) {

        echo "Увлечение " . ++$key . " - " . $val .
        "<br/>";

    }

};

$hobby = ["Спорт", "Программирование", "Censored",
"Censored", "Censored"];

// вызов функции
```

```
outHobby ($hobby) ;
```

```
?>
```

Передача **параметров переменной длины** и передача **массива** это совсем не одно и то же:

- в первом случае передаются **несколько аргументов**. **Тип данных** аргумента может быть **любой** (в т.ч. массив);
- во втором случае передается **один аргумент**. **Тип данных** – **массив**.

Более подробно практика применения параметров переменной длины рассмотрена в файлах директории **example_7**.

Важно. Умение правильно **передать** и **использовать** аргументы внутри функции – залог уверенной работы разработчика web-приложений.

Необязательные параметры

Выше, при определении функции мы были обязаны передать **для всех параметров** функции значения. Например, если функция определяет два параметра, соответственно нам надо передать в ее вызов два значения.

Однако, PHP позволяет сделать **параметры необязательными**. У такого параметра должно быть **значение по умолчанию**.

```
<?php
```

```
function displayInfo ($name, $age = 18) {  
    echo "<div>Имя: $name <br />";  
    echo "Возраст: $age</div>";  
}
```

Первый параметр указан привычно — просто название переменной. Это делает аргумент **обязательным**.

Второй параметр указан со значением в формате **параметр = значение**. Точно так же, как при создании переменной. Это указание делает аргумент **необязательным**, и задаёт ему значение **по умолчанию**.

Важно. Параметров по умолчанию может быть **любое количество**, но все они должны быть в **конце списка обязательных параметров**.

Такие строчки кода синтаксически некорректны:

```
function getParentFor($childName = 'Jon', $who){...}
function calculate($a, $b = 90, $c){...}
```

example_8. Параметры по умолчанию

```
<?php

// проверьте наличие в текущей директории папки images с
изображениями:

// -> user.jpg, noimage.jpg, hyppo.jpg

// объявляем функцию двух аргументов

// аргумент $img - имеет значение по умолчанию

function outMessage($name, $img="noimage") {

    echo "<h3>Уважаемый $name, мы рады приветствовать вас
на сайте!</h3>";

    echo "<img src='images/{ $img }.jpg' />";

};

// вызываем функцию с двумя аргументами

outMessage("Alex", "user");
```

```
outMessage("Friend", "hyppo");  
  
// вызываем функцию с одним аргументом  
  
outMessage("Bob");
```

?>

Возврат значений

Функция может возвращать некоторое значение - число, строку и т.д., некоторый результат своей работы. Для возвращения значения функция использует оператор **return**, после которого указывается **возвращаемое значение**.

Важно. Непосредственный вывод данных на экран из функции (операторами echo или print) - скорее **обучающий** элемент. В реальном приложении функции на экран, как правило, ничего не выводят. Функции **возвращают данные**, которые потребляются **другими функциями**.

Например, определим функцию, возвращающую сумму двух чисел:

```
<?php  
  
function add($a, $b) {  
    return $a + $b; // возвратим результат функции  
}  
  
// переменной присвоим результат возвращаемый функцией  
  
$result = add(5, 6);  
  
echo $result; // 11
```

Функция add() принимает два параметра и возвращает сумму их значений: \$a + \$b.

Поскольку функция **возвращает значение**, его можно присвоить переменной:

```
$result = add(5, 6);  
echo $result;
```

Либо использовать **напрямую**:

```
echo add(4, 8);
```

Если после инструкции **return** в функции идут другие инструкции, то они **выполняться не будут**:

```
function add($a, $b)  
{  
    $sum = $a + $b;  
    return $sum;           // завершение функции  
    echo "sum = $sum";    // эта строка не будет выполняться  
}
```

Важно. При вызове return исполнение пользовательской функции **прерывается**, а конструкция return **возвращает** определенное значение.

Возврат приводит к завершению выполнения функции и **передаче управления** обратно к той строке кода, в которой данная функция была вызвана (внешней программе).

Перепишем функцию из example_3, определяющую площадь круга, но теперь сделаем это так, чтобы функция не выводила, а **возвращала результат своей работы**.

example_9. Оператор return

```
<?php  
  
    // определение функции  
  
    function getCircleArea($radius) {
```

```
// задаем константу числа пи

define("PI", 3.1415);

$result = PI * $radius * $radius;

// отдадим результат во внешнюю программу

return $result;

}

// вызов функции, вывод результата в браузер

echo getCircleArea(5);

?>
```

К сведению. В реальности, даже если функция не использует оператор **return**, она все равно **возвращает значение**, только в этом случае это значение - **Null**.

example_10. Возврат значений

```
<?php

function add($a, $b){

    $sum = $a + $b;

    // есть оператор echo, но нет return

    echo "Сумма = $sum <br />";

}

// вызываем функцию, в которой нет return

$result = add(5, 6);

// проверяем, что возвращает такая функция

if($result === null)

    echo "result павен Null";
```



```
else

    echo "result не равен Null";

?>
```

Функция может возвращать **любой тип данных**. в том числе это могут быть списки и объекты.

В коде файлов директории **example_11** подробно продемонстрированы **некоторые** типы данных, возвращаемых функциями.

example_11/index_1.php. Функция возвращает NULL

```
<?php

function getInfoPerson($id = null) {

    // по переданному идентификатору $id функция должна
    // получить информацию по персоне,

    // скорее всего информация будет получаться из базы
    // данных, представим, что массив $person - полученная
    // информация

    $person = [

        "name" => "Анастасия",

        "surname" => "Ягужинская",

        "patronymic" => "Павловна",

        "category" => "Первая",

        "education" => [

            "Московский политехнический университет",

            "Российский государственный социальный
            университет",

            "ГБПОУ Урюпинский агропромышленный техникум"

        ],

        "disciplines" => [
```

```

        "Информатика",
        "Основы программирования",
        "Операционные системы"

    ]

];

// 1. возвращаем null
}

// переменная будет содержать то, что вы ей вернете
$var = getInfoPerson();

// 1.
echo "<h2>Возвращаемое значение - null</h2>";
echo "<pre>";
var_dump ($var);
echo "</pre>";
?>

```

example_11/index_2.php. Функция возвращает строку

```

<?php

function getInfoPerson($id = null) {

    // ...

    // 2. возвращаем строку

    return $person["surname"];

}

// переменная будет содержать то, что вы ей вернете
$var = getInfoPerson();

// 2.
echo "<h2>Возвращаемое значение - строка</h2>";

```

```
echo "<pre>";

var_dump ($var);

echo "</pre>";

// тестовый вывод

echo $var;

?>
```

example_11/index_3.php. Функция возвращает массив

```
<?php

function getInfoPerson($id = null) {

    // ...

    // 3. возвращаем массив

    return $person["education"];

}

// переменная будет содержать то, что вы ей вернете

$var = getInfoPerson();

// 3.

echo "<h2>Возвращаемое значение - массив</h2>";

echo "<pre>";

var_dump ($var);

echo "</pre>";

// тестовый вывод

echo $var[1];

?>
```

example_11/index_4.php. Функция возвращает многомерный массив

```
<?php
```

```

function getInfoPerson($id = null) {

    // ...

    // 4. возвращаем многомерный массив

    return $person;

}

// переменная будет содержать то, что вы ей вернете

$var = getInfoPerson();

// 4.

echo "<h2>Возвращаемое значение - многомерный массив</h2>";

echo "<pre>";

var_dump ($var);

echo "</pre>";

// тестовый вывод

echo $var["education"][2];

?>

```

example_11/index_5.php. Функция возвращает объект

```

<?php

function getInfoPerson($id = null) {

    // ...

    // 5. возвращаем объект

    return (object) $person;

}

// переменная будет содержать то, что вы ей вернете

$var = getInfoPerson();

// 5.

echo "<h2>Возвращаемое значение - объект</h2>";

```

```
echo "<pre>";

var_dump ($var);

echo "</pre>";

// тестовый вывод

echo $var->name . "<br />";

echo $var->education[0] . "<br />";

?>
```

В качестве эпилога

Конструкция **return** возвращает значения, преимущественно из **пользовательских функций**, как параметры функционального запроса.

Если конструкция **return** будет вызвана из **глобальной области определения** (вне пользовательских функций), то скрипт также завершит свою работу, а **return** возвратит определенное значение.

Для демонстрации возможности проанализируйте код файла **example_12.php**.

example_12.

```
<ol>

    <li>Зеленая миля:

        <ul>
```

```
<li>режиссёр: Фрэнк Дарабонт</li>

<li>год: 1999</li>

<li>страна: США</li>

</ul>

</li>

<!-- ... -->

</ol>

<?php
    // прекратим выполнение скрипта

    return;

?>

<h1>Цитаты русских писателей</h1>

<div id="cite">

    <p>Замечательный день сегодня. То ли чай пойти выпить, то ли
    повеситься (А.Чехов) .

    <p>А она — подошла к столу и выпила, залпом, ещё сто
    пятьдесят, ибо она была совершенна, а совершенству нет
    предела (В.Ерофеев) .

    <p>Хорошо идти, когда зовут. Ужасно — когда не зовут. Однако
    лучше всего, когда зовут, а ты не идёшь (С.Довлатов) .

</div>
```

Задание 56

=====

Используя решение задания 32 **Темы Управляющие конструкции - Циклы**, перепишите вывод данных массива **\$team** с использованием **пользовательской функции**.

Алгоритм решения:

1. **Подключите** файл с массивом.
2. **Определите** функцию вывода данных массива. При определении функции укажите параметр массив - **\$arr**.

Формат вывода оставьте без изменений:

Группа: _____ (id = _____)

Страна: _____

Дата основания: _____

Стиль: _____

<hr/>

<p>

3. **Вызовите** функцию вывода.

Задание 57

=====

Вам предоставлен файл **album.php** с массивом **\$album**. Напишите программу вывода данных массива **\$album** с использованием **пользовательской функции**. Вывод в браузер из функции осуществить с использованием оператора **return**.

Фрагмент кода вывода может быть таким:

```
// включение файла album.php
require "album.php";

function fnOutAlbum($arr) {
    // формируем строку для вывода
    // ...

    return $out;
}

// вывод альбомов из массива album
echo fnOutAlbum($album);
```

Формат вывода оставить без изменений:

ID	Альбом	Дата выпуска	Страна
...

Задание 58

=====

Вам предоставлен файл **track.php** с массивом **\$track**. Напишите **пользовательскую функцию** вывода треков по **заданному идентификатору альбома**. Идентификатор альбома передайте в виде **аргумента функции**.

Определение функции вынесите в **отдельный** файл.

Вывод данных организовать в виде **табличного** представления:

ID трека	Название трека	Примечание	Альбом
...

Фрагмент кода вывода может быть таким:

```
// включение файла track.php
include "track.php";

// подключение файла с функцией
include "function.php";

// идентификатор альбома
$id = 10;

// вывод треков указанного альбома ($id) из массива $track
fnOutTrack($track, $id);
```

Задание 59

=====

В качестве раздаточного материала вам предлагаются файлы и каталоги:

1. Каталог **dump**. В каталоге находятся **файлы с массивами**, имитирующими **данные по преподавателю**, полученные из базы данных.
2. Файл **function.php**. В файле определена функция **fnGetData()**, подключающая массивы из директории **dump** и отдающая данные во внешнюю **программу**.
3. Файл **index.php**. В файле определён **шаблон** сценария создаваемой программы (рис.1).

Задача.

- Ознакомьтесь с массивами данных из директории **dump**.
- Изучите способ формирования данных функцией **fnGetData()**.
- Напишите определение функций:
 - `getPersonData()`,
 - `getPersonEdu()`,
 - `getPersonCours()`,

осуществляющих вывод данных в браузер согласно **рисунка 2**.

```
1  <!DOCTYPE html>
2  <html>
3  <body>
4      <?php
5          // подключаем функцию fnGetData()
6          require 'function.php';
7          // получаем возвращаемый функцией массив
8          $data = fnGetData();
9
10         // echo "<pre>";
11         // var_dump($data);
12         // echo "</pre>";
13
14         // забираем данные по категории
15         $person = $data["personnel"];
16         $courses = $data["courses"];
17         $educations = $data["educations"];
18
19         function getPersonData($data) {
20             // ...
21
22         };
23
24         function getPersonEdu($data) {
25             // ...
26
27         };
28
29         function getPersonCours($data) {
30             // ...
31
32         }
33
34         // выводим персональные данные
35         echo getPersonData($person);
36         // выводим данные об образовании
37         echo getPersonEdu($educations);
38         // выводим данные о курсах
39         echo getPersonCours($courses);
40     ?>
41 </body>
42 </html>
```

Line 19, Column 40

Tab Size: 4

PHP

Рис.1. Шаблон файла сценария программы

php-course/task_5_6.php

← → ↻ ⚠ Не защищено | php-course/task_5_6.php

🔍 📄 ☆ ⚙ 0 Обновить

Бородина Ксения Алексеевна

getPersonData ()

Категория: Соответствие занимаемой должности

Образование:

getPersonEdu ()

Обучение	Институт	Квалификация	Специализация
2005 - 2010	Московский государственный институт электронной техники (технический университет)	Факультет инфокоммуникационных технологий	Программное обеспечение радиоэлектронных систем
1993 - 1998	Санкт-Петербургский государственный университет (СПбГУ)	Информационные системы и технологии	Безопасность киберфизических систем

Курсы:

getPersonCours ()

Наименование	Продолжительность	Цена
Fullstack разработчик на Python	220	110.000
Профессия веб-разработчик	100	90.000
Профессия Android-разработчик	150	135.000

Рис.2. Вывод данных в браузер

Задание 60

=====

файлах **dump** расположены **массивы**, в которых содержится информация по одному преподавателю. Усложним задачу.

Теперь файлы категории dump содержат **массивы информации** по многим преподавателям. Изучите структуру предлагаемых массивов. Повторите вывод данных о преподавателе задания 5_17. Идентификатор искомого преподавателя передавать через **строку запроса** в **GET-параметре id**.

Если пользователь **не указал** GET-параметр, вывести данные по преподавателю с **идентификатором 1**.

Область видимости переменных

- Введение
- Переменные в блоках циклов и условных конструкций
- Локальные переменные
- Статические переменные
- Глобальные переменные
- Константы
- Передача аргументов по ссылке и по значению
- Время жизни переменной
- Unset внутри функции
- Некоторые итоги

Введение

Область видимости переменной — это **контекст**, в котором эта **переменная определена** (видима). В большинстве случаев все переменные PHP имеют только одну область видимости.

Важно. Единая область видимости охватывает также **включаемые** (include) и **требуемые** (require) файлы.

Например:

```
<?php  
  
    $var = 1;  
  
    include 'file.inc';
```

Здесь **переменная \$var будет доступна внутри включённого скрипта file.inc.**

Переменные в блоках циклов и условных конструкций

Блоки циклов и условных конструкций **не образуют отдельной области видимости**, и переменные, определенные в этих блоках, мы можем использовать **вне этих блоков**:

```
<?php

$condition = true;

if($condition){

    $name = "Venus";

}

echo $name; // $name доступна за пределами блока if
```

или

```
<?php

$i = 1;

while($i<10) {

    $i = $i +1;

}

echo $i; // $i доступна за пределами цикла
```

Локальные переменные

Определение (тело) пользовательской функции задаёт **локальную область видимости** данной функции.

Важно. Любая используемая **внутри функции** переменная по умолчанию ограничена **локальной областью видимости функции**.

К таким переменным можно обратиться только **изнутри** данной функции. Например:

```
<?php

function showName() {

    $name = "Venus";

    echo '$name внутри функции: ', $name, "<p>";

}

showName();

// так написать нельзя

echo '$name снаружи функции: ', $name;

// переменная $name существует только внутри функции
```

В данном случае в функции showName() определена локальная переменная **\$name**. Соответственно обратиться к этой переменной мы можем **только внутри функции showName()**, но не вне ее.

То же самое относится и к параметрам функции: **вне функции ее параметры также не существуют**.

```
<?php

function getUser($par1, $par2){

    $par1 = "admin";

    $par2 = "12345";

}

$login = "";

$pwd = "";

getUser($login, $pwd);

echo "Логин = $par1; Пароль = $par2";
```


Статические переменные

Статические переменные похожи на локальные. Их особенность в том, что после завершения работы функции их **значение сохраняется**.

Важно. При каждом новом вызове - **функция использует ранее сохраненное значение статической переменной**.

Чтобы указать, что переменная будет статической, к ней добавляется ключевое слово **static**.

example_1. Статическая переменная

```
<?php

function getCounter() {

    // объявление статической переменной

    static $counter = 0;

    $counter++;

    echo $counter;

    echo "<br />";

}

// первый вызов
getCounter(); // counter=1

// второй вызов
getCounter(); // counter=2

// третий вызов
getCounter(); // counter=3

?>
```

При трех последовательных вызовах функции **getCounter()** переменная **\$counter** будет **увеличиваться на единицу**.

Если бы переменная **\$counter** была обычной нестатической, то при каждом вызове функция **getCounter()** выводила бы 1 (можно поэкспериментировать).

Как правило, статические переменные служат для создания различных счетчиков, как в примере выше.

Глобальные переменные

Глобальные переменные по умолчанию недоступны внутри функции. Например:

```
<?php

function hello(){

    echo "Hello, " . $name;

}

$name = "Tom";

hello(); // Hello,
```

Данный код **не будет работать**, а интерпретатор PHP известит нас, что переменная **\$name не определена**.

Тем не менее, мы можем обратиться внутри функции к глобальной переменной. Для этого необходимо использовать ключевое слово **global**.

```
<?php

function hello() {

    global $name;

    echo "Hello, " . $name;

}

$name = "Tom";

hello(); // Hello, Tom
```

Для получения доступа к глобальной переменной в функции **с помощью оператора `global`** объявляется **переменная с тем же именем**:

```
global $name;
```

После этого к глобальной переменной `$name` можно будет обращаться внутри функции. При этом мы можем не только **получить** ее значение, но и **изменить** его.

example_2. Глобальная переменная

```
<?php

    function changeName () {

        global $name;

        $name = "Tomas";

    };

    $name = "Tom";

    echo $name, "<br />"; // Tom

    changeName ();

    echo $name; // Tomas

?>
```

Константы

Константы — это именованные значения, которые не могут быть изменены во время выполнения программы. Они отличаются от переменных тем, что их значение остаётся постоянным на протяжении всего скрипта

Важно. Константы в PHP доступны во всех областях видимости, включая **глобальную** и **локальную** области видимости.

Константы могут содержать только **скалярные значения** (строки, числа или булевы значения) или **массивы**.

Для определения константы в PHP используется функция **define()**.

Константы в PHP необходимо определить перед их использованием.

example_3. Константа в глобальной области видимости

```
<?php

// объявление константы в глобальной области видимости
define('MYCONST', "Глядя на мир, нельзя не удивляться!");

// объявление глобальной переменной
$cite = "Нельзя объять необъятное!";

function outVariable() {

    // вывод константы

    echo '<h3>' . MYCONST . '</h3>';

    // вывод переменной

    echo '<h3>' . $cite . '</h3>';

};

outVariable();

?>
```

example_4. Константа в локальной области видимости

```
<?php

// функция определения константы

function outVariable() {

    // объявление константы в локальной области видимости

    define('MYCONST', "Глядя на мир, нельзя не удивляться!");

};
```

```
// вызов функции

outVariable();

// вывод константы, определенной в функции

echo '<h3>' . MYCONST . '</h3>';

?>
```

Передача аргументов по ссылке и по значению

По умолчанию аргументы функции передаются **по значению**. Это означает, что если значение аргумента внутри функции изменяется, оно **не изменится вне функции**.

example_5. Передача аргумента по значению

```
<?php

// передаем аргумент по значению

function inc($par) {

    $par = $par + 10;

    echo "Локальная переменная = " . $par; // 15

    echo "<p>";

}

$var = 5;

// вызываем функцию

inc($var);

echo "Глобальная переменная = " . $var; // 5

?>
```

Чтобы функция могла изменять свои аргументы, они должны **передаваться по ссылке**. Чтобы аргумент функции передавался по

ссылке, необходимо добавить амперсанд (&) к имени параметра в определении функции.

К сведению. На самом деле запомнить это правило не трудно. **По значению** - значит, передается копия значения переменной. **По ссылке** - передается ссылка на саму переменную.

В следующем листинге приведен пример передачи аргумента по ссылке.

example_6. Передача аргумента по ссылке

```
<?php

    // передаем аргумент по ссылке

    function inc(&$par) {

        $par = $par + 10;

        echo "Локальная переменная = " . $par; // 15

        echo "<p>";

    }

    $var = 5;

    // вызываем функцию

    inc($var);

    echo "Глобальная переменная = " . $var; // 15

?>
```

В вызове функции знак ссылки указывать не нужно, он есть только в определении функции. Этого достаточно для корректной передачи аргументов по ссылке.

При передаче в качестве аргументов функции массивов необходимо придерживаться тех же правил. Ниже представлены примеры передачи массива по значению и по ссылке.

example_7. Массив как аргумент по значению

```
<?php

// передаем аргумент-массив по значению

function fnChangingArray($arr) {

    $arr[0] = "javascript";

    $arr[1] = "css";

    $arr[2] = "vue.js";

}

$array = array("php", "laravel", "mysql");

echo "<pre>";

print_r($array);

// вызываем функцию

fnChangingArray($array);

echo "<pre>";

print_r($array);

?>
```

example_8. Массив как аргумент по ссылке

```
<?php

// передаем аргумент-массив по ссылке

function fnChangingArray(&$arr) {

    $arr[0] = "javascript";

    $arr[1] = "css";

    $arr[2] = "vue.js";

}

$array = array("php", "laravel", "mysql");

echo "<pre>";
```

```
print_r($array);

// вызываем функцию
fnChangingArray($array);

echo "<p>";

print_r($array);

?>
```

Важно. Исключение в языке программирования PHP составляют объекты. **Объекты всегда передаются по ссылке.**

Пример передачи объекта в виде аргумента функции представлен в листинге. Если действия с объектом вам непонятны, ничего страшного, главное убедиться, что свойства объекта (role, pwd) меняются. Объекты будем изучать далее.

example_9. Объект как аргумент по значению

```
<?php

// создаем класс
class UserClass {

    public $login = "master";

    public $role = "user";

    public $pwd = "12345";

}

// передаем объект в качестве аргумента по значению
function fnChangingObj($obj) {

    global $role, $pwd;

    $obj->role = $role;

    $obj->pwd = $pwd;
```



```
}

// создаем экземпляр класса

$user = new UserClass();

echo "<pre>";

print_r($user);

$role = "admin";

$pwd = "qwerty";

fnChangingObj($user);

echo "<p>";

print_r($user);

?>
```

К сведению. На самом деле с объектами все **немного сложнее**. И знак амперсанда (&) в сигнатуре функции все таки имеет значение. Подробнее этот вопрос будет рассмотрен в теме, посвященной объектам.

Время жизни переменной

Временем жизни переменной называется интервал выполнения программы, в течение которого она существует.

Важно. Интервал выполнения программы в котором существует переменная есть **время жизни** этой переменной.

Поскольку локальные переменные имеют своей областью видимости функцию, то время жизни локальной переменной определяется временем выполнения функции, в которой она объявлена. Это означает,

что в разных функциях совершенно независимо друг от друга могут использоваться переменные с одинаковыми именами.

Локальная переменная при каждом вызове функции **инициализируется заново**, поэтому функция-счетчик, в приведенном ниже примере всегда будет возвращать значение 1:

```
<?php

function counter() {

    $counter = 0;

    return ++$counter;

}
```

Для того, чтобы локальная переменная сохраняла свое предыдущее значение при новых вызовах функции, ее можно объявить статической при помощи ключевого слова **static**:

```
<?php

function counter() {

    static $counter = 0;

    return ++$counter;

}
```

Важно. Временем жизни статических переменных является **время выполнения сценария**.

Т. е., если пользователь перезагружает страницу, что приводит к новому выполнению сценария, переменная **\$counter** в этом случае инициализируется заново.

Unset внутри функции

В "Встроенные функции 1" мы рассмотрели функцию **unset()** — удаляющую одну или несколько перечисленных переменных.

```
<?php

$var = 'global variable';

var_dump($var);

echo "<p>";

unset($var);

var_dump($var);
```

Поведение **unset()** **внутри пользовательской функции** может отличаться, в зависимости от того, какой тип имеет переменная, которую необходимо удалить.

Если переменная, объявленная **глобальной**, удаляется внутри функции, то будет удалена только **локальная переменная**. Переменная в области видимости вызова функции сохранит то же значение, что и до вызова **unset()**.

example 10. Поведение unset при удалении глобальной переменной

```
<?php

function fnUnsetVar() {

    // объявляем переменную глобальной
    global $var;

    // удаляем глобальную переменную внутри функции
    unset($var);

    var_dump($var); // Warning: Undefined variable $var
    echo "<p>";

}
```

```

// иницилируем глобальную переменную

$var = 'global variable';

fnUnsetVar();

// выводим глобальную переменную

// во внешней программе ничего с переменной не произошло

var_dump($var);

?>

```

Если переменная, которая передаётся **по ссылке**, удаляется внутри функции, то будет удалена только **локальная переменная**.

Переменная в области видимости вызова функции сохранит то же значение, что и до вызова **unset()**.

example_11. Поведение unset при удалении переменной по ссылке

```

<?php

function fnUnsetVar(&$par) {

    // удаляем переменную переданную по ссылке

    unset($par);

    var_dump($par); // Warning: Undefined variable $par

    echo "<p>";

}

// иницилируем глобальную переменную

$var = 'global variable';

fnUnsetVar($var);

// выводим глобальную переменную

// во внешней программе ничего с переменной не произошло

var_dump($var);

?>

```

Некоторые итоги

Подведем некоторые итоги чрезвычайно важно, посвященно функциям.

Функции это **именованный блок кода**, который можно многократно вызывать в различных частях программы. Это своего рода "переменная", в которой хранится блок кода, который **выполняется** при использовании этой "переменной".

Функции позволяют разделять программу на меньшие **функциональные** части. Это очень мощный инструмент **повторного использования кода**. Чтобы упростить себе работу, мы можем оформить в виде функции некоторую часть кода, которая используется в сценарии несколько раз.

Функция не выполняется сразу при загрузке страницы. Функция будет выполняться при ее вызове из тела программы.

Разделяют два типа функций — **встроенные** и **пользовательские**.

- **Встроенные функции** — это функции, которые за нас уже написали создатели языка программирования, и мы можем просто брать их и использовать. В PHP существуют тысячи готовых функций.
- **Пользовательские функции** — это функции, которые программист создает самостоятельно.

Функции могут принимать на вход некоторую информацию из внешней программы. **Информация** передается в функцию через **параметры**. При этом следует различать следующие понятия:

- **Параметр** — **переменная**, создаваемая при определении функции.

- **Аргумент — значение**, которое передается в функцию при её вызове.

Существует два **основных** способа передать функции входные параметры:

1. сделать переменную **глобальной**,
2. передать переменную **параметром** функции:
 - a. по значению,
 - b. по ссылке.

Замечание. Мы не берем в рассмотрение **суперглобальные переменные массива \$GLOBALS**, изучением которых займемся в отдельном .

Закрепим знания о каждом способе еще раз в отдельных примерах.

example_12/index.php. Использование глобальных переменных

```
<?php

// включаем массив $team

// проверьте наличие файла team.php в директории

require "team.php";

// определяем функцию

function search() {

    // делаем переменные глобальными

    global $team, $name;

    foreach ($team as $key => $item) {

        if ($item["name"] == $name) {

            return $item;

        }

    }

}
```

```
}

// определяем критерий поиска

$name = "Aerosmith";

// вызываем функцию поиска

$data = search();

// выводим результат

echo "<pre>";

var_dump($data);

?>
```

example_13/index.php. Использование аргументов функции

```
<?php

// включаем массив $team

// проверьте наличие файла team.php в директории

require "team.php";

// определяем функцию с двумя параметрами

function search($arr, $term) {

    foreach ($arr as $key => $item) {

        if ($item["name"] == $term) {

            return $item;

        }

    }

}

// определяем критерий поиска

$name = "Aerosmith";

// вызываем функцию поиска

$data = search($team, $name);
```

```
// выводим результат
```

```
echo "<pre>";
```

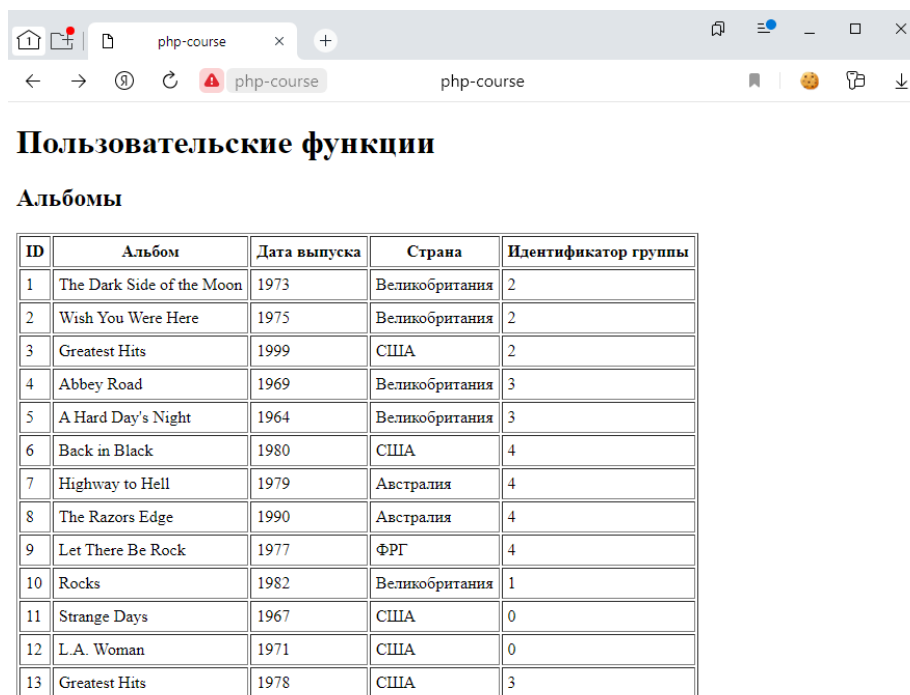
```
var_dump($data);
```

```
?>
```


Задание 61

=====

В директории раздаточного материала вам предоставлен файл **index.php**. Скрипт файла выполняет вывод в браузер таблицы **Альбомы** (рис.1).

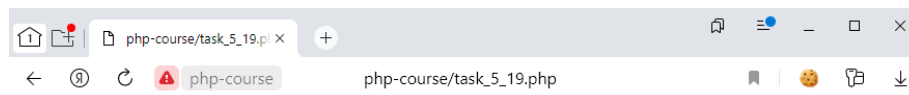


The screenshot shows a web browser window with the address bar displaying 'php-course'. The page title is 'Пользовательские функции'. Below the title is a section header 'Альбомы'. The main content is a table with 5 columns: ID, Альбом, Дата выпуска, Страна, and Идентификатор группы. The table contains 13 rows of data.

ID	Альбом	Дата выпуска	Страна	Идентификатор группы
1	The Dark Side of the Moon	1973	Великобритания	2
2	Wish You Were Here	1975	Великобритания	2
3	Greatest Hits	1999	США	2
4	Abbey Road	1969	Великобритания	3
5	A Hard Day's Night	1964	Великобритания	3
6	Back in Black	1980	США	4
7	Highway to Hell	1979	Австралия	4
8	The Razors Edge	1990	Австралия	4
9	Let There Be Rock	1977	ФРГ	4
10	Rocks	1982	Великобритания	1
11	Strange Days	1967	США	0
12	L.A. Woman	1971	США	0
13	Greatest Hits	1978	США	3

Рис.1.

Модернизируйте вывод таблицы **Альбом** таким образом, чтобы вместо столбца **Идентификатор группы** выводилось **Наименование группы** (рис.2). Массивы **\$album**, **\$team**, представляющие соответствующие таблицы, прилагаются.



Область видимости переменных

Альбомы

ID	Альбом	Дата выпуска	Страна	Наименование группы
1	The Dark Side of the Moon	1973	Великобритания	Pink Floyd
2	Wish You Were Here	1975	Великобритания	Pink Floyd
3	Greatest Hits	1999	США	Pink Floyd
4	Abbey Road	1969	Великобритания	The Beatles
5	A Hard Day's Night	1964	Великобритания	The Beatles
6	Back in Black	1980	США	AC/DC
7	Highway to Hell	1979	Австралия	AC/DC
8	The Razors Edge	1990	Австралия	AC/DC
9	Let There Be Rock	1977	ФРГ	AC/DC
10	Rocks	1982	Великобритания	Aerosmith
11	Strange Days	1967	США	---
12	L.A. Woman	1971	США	---
13	Greatest Hits	1978	США	The Beatles

Рис.2.

Для перевода **идентификатора группы в название** используйте вторую функцию. Для передачи данных в функцию используйте глобальную переменную.

Код вызова функции может быть таким, как представлено на **фрагменте** ниже:

```
function fnOutAlbum() {  
    global $album;  
  
    // перебираем массив $album для формирования таблицы  
    foreach ($album as $key => $item) {  
        // вызываем функцию поиска группы по ее идентификатору  
        $name = fnGetTeamName($item['id_team']);  
  
        $str .= "  
        <tr>  
  
        <td>{$item['id_album']}
```

```
        <td>{$item['title']}</td>
        <td>{$item['date']}</td>
        <td>{$item['country']}</td>
        <td>$name</td>
    </tr>
    ";
}; // end foreach
// ...
```

Задание 62

=====

Ревьюируйте решение **задания 1**.

- **Определение** функций вынесите в **отдельный** файл. Итого, в файле основного скрипта необходимо будет подключать **три внешних файла**.
- **Подключаемые** массивы внутрь функций передавайте НЕ через **глобальную** переменную (**global**), а с использованием **параметров функции**.

Таким образом, результатом ревьюирования станет намного более простой для восприятия алгоритм работы основного сценария **task/index.php** первого задания.

```
<?php

    // включение файла album.php
    require "album.php";

    // включение файла team.php
    require "team.php";

    // включение файла с функциями
    require "fun.php";

    // вывод альбомов из массива album
    echo fnOutAlbum($album, $team);

?>
```

Раздаточный материал:

- Файлы массивов **album.php**, **team.php**.
- Файл основного сценария **index.php**.

- Файл определения функций **fun.php** (требуется кодировка определения функций).

Задание 63

=====

В файле **personnel.php** раздаточного материала инициирован двумерный ассоциативный массив **\$personnel**. Передавая скрипту **index.php** следующие GET-параметры:

- **num** - порядок вложенного массива преподавателя,
- **term** - терм поиска (ключ вложенного массива, значение которого нужно поменять),
- **val** - новое значение для ключа массива,

мы планируем выполнить следующие действия:

1. Вывести в браузер вложенный массив (преподавателя) с указанным номером **num**.
2. Поменять значение **val** указанного ключа **term** для выбранного массива.

Строка запроса может выглядеть следующим образом:

```
// выбрать вложенный массив с индексом = 1,  
// ключу "surname" присвоить новое значение "Иванов"  
?num=1&term=surname&val=Иванов  
  
// или так  
?num=2&term=post&val=Мастер
```

Массив выбирается и выводится, но значение ключа не меняется.

Выполните **отладку сценария** для реализации намеченных планов.

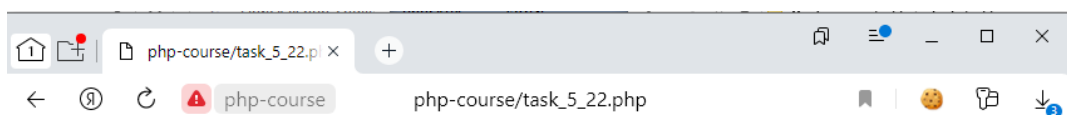
Задание 64

=====

В файле **personnel.php** раздаточного материала инициирован двумерный ассоциативный массив **\$personnel**. В файле **index.php** представлен скрипт, выводящий массив **\$personnel** в виде **таблицы**. Параллельно выводу таблицы происходит сбор **статистических данных** по преподавателям, **не имеющих информации об адресе персонального сайта или электронной почты**.

Вывод статистики выполняется с использованием конструкции **var_dump**. Скрипт показывает **неверную выборку статистики** (в списке должно быть три преподавателя).

1. Найдите "**ошибку**", исправьте скрипт.
2. Напишите функцию вывода статистических данных в виде **таблицы**.



Область видимости переменных

Список преподавателей

№	Фамилия Имя Отчество	Должность	Категория
1	Бородина Ксения Алексеевна	Преподаватель	Соответствие занимаемой должности
2	Кошкина Вера Николаевна	Преподаватель	Соответствие занимаемой должности
3	Рыбкина Ольга Витальевна	Преподаватель	Соответствие занимаемой должности
4	Маркова Елизавета Андреевна	Преподаватель	Первая
5	Ильчук Виталина Георгиевна	Преподаватель	Высшая

А вот этих мы лишим премии...

№	Фамилия Имя Отчество	Сайт	E-mail
1	Бородина Ксения Алексеевна	borodina.wordpress.com	
2	Кошкина Вера Николаевна		koshkina_v_n@mail.ru
3	Маркова Елизавета Андреевна		markova.el@gmail.com

Анонимные функции

- Введение в анонимные функции
- Замыкания (closures)
- Стрелочные функции
- Функции обратного вызова
- Подведем итоги

Введение в анонимные функции

Анонимные функции позволяют создавать функции, не имеющие определенных имен. Синтаксис определения анонимной функции не отличается от синтаксиса определения обычной функции за тем исключением, что она не имеет имени.

Шаблон определения анонимной функции:

```
<?php  
  
function () {  
  
    // тело функции  
  
}
```

Анонимные функции наиболее полезны в качестве **значений параметров** других функций, но могут иметь и множество других применений.

Анонимные функции могут быть использованы в качестве **значений переменных**. В таком случае присвоение переменной использует тот же синтаксис, что и любое другое присвоение, включая завершающую точку с запятой.

Для вызова подобной функции применяется имя представляющей ее переменной.

example_1. Пример анонимной функции

```
<?php

    // присвоение переменной анонимной функции

    $greet = function() {

        echo "Привет, давайте изучать PHP вместе!";

    };

    $greet();

?>
```

Фактически подобная **переменная** применяется как **стандартная функция**.

Также анонимные функции могут **возвращать** некоторое значение:

```
<?php

    $greet = function () {

        return "Привет, давайте изучать PHP вместе!";

    };

    echo $greet();
```

Анонимные функции могут принимать обычные **параметры**.

example_2. Передача параметров анонимной функции

```
<?php

    // присвоение переменной анонимной функции

    // функции передаем параметр

    $greet = function($par) {

        printf("Привет, давайте изучать %s вместе!</p>", $par);

    };

    // иницилируем значения аргументов анонимной функции
```

```
$lang1 = "PHP";

$lang2 = "JavaScript";

// вызываем функцию

$greet($lang1);

$greet($lang2);

?>
```

Параметры анонимных функций могут передаваться как **по значению**, так и **по ссылке**.

example_3. Передача параметра по ссылке

```
<?php

// присвоение переменной анонимной функции

// функции передаем параметр по ссылке

$greet = function(&$par) {

    $par = "Виноделие";

    printf("Привет, давайте изучать %s вместе!</p>", $par);

};

// иницируем значения аргументов анонимной функции

$lang1 = "PHP";

$lang2 = "JavaScript";

// вызываем функцию

$greet($lang1);

// $lang1 передавали в функцию

echo '$lang1 = ' . $lang1 . "<p>";

// $lang2 - нет

echo '$lang2 = ' . $lang2;

// передайте параметр $par анонимной функции по значению и
```

```
протестируйте еще раз
```

```
?>
```

Кроме того, как и в обычных функциях, в анонимных функциях может использоваться ключевое слово **global**.

example_4. Использование глобальных переменных

```
<?php

// иницилируем переменную $role

$role = "Администратор";

$test = function() {

    // делаем переменную глобальной

    global $role;

    var_dump("Вы зашли как " . $role . " сайта.");

};

$test();

?>
```

Замыкания (closures)

Замыкания в PHP представляют анонимную функцию, которая может **наследовать переменные** из родительской области видимости. Любая подобная переменная должна быть объявлена в конструкции **use**.

example_5. Наследование глобальных переменных

```
<?php

$role = "Администратор";

// наследуем переменную role

$example = function () use($role) {
```

```
        var_dump("Вы зашли как " . $role . " сайта.");  
  
    };  
  
    $example();  
  
?>
```

Выражение **use** получает **внешние переменные родительской области видимости**, которые анонимная функция собирается использовать.

Внешние переменные в замыканиях могут передаваться как по **значению**, так и по **ссылке**.

example_6. Наследование по ссылке

```
<?php  
  
    $role = "Администратор";  
  
    echo '$role = ' . $role . "<p>";  
  
    // наследуем переменную role  
  
    $example = function () use(&$role) {  
  
        $role = "Модератор";  
  
    };  
  
    $example();  
  
    echo '$role = ' . $role . "<p>";  
  
?>
```

К сведению. Наследование переменных из глобальной области видимости в замыканиях **не то же самое**, что использование **глобальных** переменных.

Разница между **наследованием** переменной из глобальной видимости и использованием ключевого слова **global** наглядно продемонстрирована в коде листинга файла example_7.php.

example_7. Наследуемые и глобальные переменные

```
<?php

// иницилируем переменную $role

$role = "Администратор";

// определяем функцию, наследуем переменную role

// !!! обратите внимание -

// test_use была определена, когда $role = Администратор

$test_use = function () use($role) {

    var_dump("Вы зашли как " . $role . " сайта.");

};

// определяем функцию, делаем переменную role глобальной

$test_global = function () {

    global $role;

    var_dump("Вы зашли как " . $role . " сайта.");

};

// переопределим переменную $role

$role = "Модератор";

$test_use(); // Вы зашли как Администратор сайта

echo "<p>";

$test_global(); // Вы зашли как Модератор сайта

?>
```

Важно. Значение унаследованной переменной задано там, где

функция **определена**, но не там, где **вызвана**.

Подобным образом функция-замыкание может захватывать и **большее количество внешних переменных**.

example_8. Наследуемых переменных может быть много

```
<?php

    // иницилируем переменные

    $phone = "2-12-85-06";

    $role = "Администратор";

    // передаем параметры, наследуем переменные

    $message = function ($login, $pwd) use ($role, $phone) {

        echo "<pre>";

        var_dump($login, $pwd, $role, $phone);

    };

    // вызываем функцию с аргументами

    $message("master", "12345");

?>
```

Ну, и естественно, можно сочетать все эти способы работы с переменными в анонимных функциях

example_9. Одержимым шпиономанией

```
<?php

    // иницилируем переменные

    $phone = "2-12-85-06";

    $role = "Администратор";

    // передаем параметры, наследуем переменные
```

```
$message = function ($login, $pwd) use ($role) {  
  
    // получаем доступ  
  
    global $phone;  
  
    echo "<pre>";  
  
    var_dump($login, $pwd, $role, $phone);  
  
};  
  
// вызываем функцию с аргументами  
  
$message("master", "12345");  
  
?>
```

Объявление **типа возвращаемого значения** функции может быть помещено после конструкции **use**.

```
<?php  
  
$user = "Batman";  
  
$example = function () use ($user): string {  
  
    return "Привет mr. $user";  
  
};  
  
echo $example();
```

Стрелочные функции

Стрелочные функции (arrow function) появились в PHP 7.4, как более лаконичный синтаксис для анонимных функций, которые возвращают некоторое значение. И при этом стрелочные функции **автоматически** имеют доступ к переменным из **внешнего** окружения.

Стрелочная функция определяется с помощью оператора **fn**:

```
fn (параметры) => действия;
```

example_10. Стрелочная функция

```
<?php

    // инициализация переменных

    $a = 8;

    $b = 10;

    // определение стрелочной функции

    $fun = fn() => $a + $b;

    // вызов функции-стрелки

    echo $fun(); // 18

?>
```

Вот некоторые особенности использования стрелочных функций в PHP:

- становятся доступными в PHP 7.4;
- начинаются с ключевого слова **fn**;
- могут иметь только **одно** выражение, которое являет собой **return** выражение, при этом ключевое слово **return** указывать **не** нужно;
- функциям могут задаваться возвращаемые типы.

Ниже представлен пример использования анонимной функции и аналогичной ей стрелочной.

example_11. Анонимная функция против стрелочной

```
<?php

    $y = 5;

    // определение и присвоение переменной анонимной функции

    $fun1 = function ($x) use ($y) {

        return $x + $y;

    };

    // вызов анонимной функции
```



```
var_dump ($fun1(4)); // 9

// определение и присвоение переменной стрелочной функции

$fun2 = fn($x) => $x + $y;

echo "<p>";

// вызов стрелочной функции

var_dump ($fun2(4)); // 9

?>
```

Разработчик с опытом программирования в JavaScript, может попробовать описать стрелочную функцию указывая несколько выражений.

```
$fun = fn() => {

    // ...

    return 1;

}
```

Но, в PHP так делать нельзя.

Важно. Стрелочная функция описывается в одну строку, использовать несколько строк для её описания **нельзя**.

Стрелочные функции используют привязку переменных **по значению**. Привязка по значению означает, что невозможно изменить какие-либо значения из внешней области. Вместо этого можно использовать анонимные функции для привязок по ссылкам.

example_12. Привязка переменных в стрелочных функциях

```
<?php

$x = 5;

// ничего не изменит
```

```
$fn = fn() => $x++;  
  
$fn();  
  
echo $x; // 5  
  
?>
```

Функции обратного вызова

Распространенным случаем применения анонимных функций является передача их параметрам **других функций**. Такие анонимные функции называют **функциями обратного вызова** или коллбеками (callback function).

Важно. Функция, переданная в качестве параметра другой функции, называется **функцией обратного вызова** (callback).

Функции обратного вызова достаточно объемный материал для изучения при программировании пользовательских функций.

Рассмотрим принцип действия callback-функций на примере работы стандартной функции **array_map** (Встроенные функции 3).

Функция **array_map** служит для **итеративной** обработки элементов массива. Callback-функция применяется к **каждому элементу массива** и в качестве результата выдается обработанный массив.

```
array_map(callback, $array);
```

Параметр

- **callback** - функция, применяемая к каждому элементу в каждом массиве.
- **array** - массив, к каждому элементу которого применяется функция.

example_13. Функция array_map

```
<?php

    // иницилируем исходный массив

    $nums = array(1, 2, 3, 4, 5);

    // в качестве параметра передаем анонимную функцию
    // функция возводит в квадрат переданный параметр

    $result = array_map(function ($num) {

        return $num * $num;

        // без return результирующий массив будет пустой

    }, $nums);

    // выводим результат

    echo "<pre>";

    print_r($result); // 1,4,9,16,25

?>
```

В коде листинга example_14 приведен другой вариант использования callback-функции.

example_14. Определение callback-функции

```
<?php

    // иницилируем исходный массив

    $nums = array(1, 2, 3, 4, 5);

    // определяем функцию

    $fnSquare = function ($num) {

        return $num * $num;

    };

    // в качестве параметра передаем callback-функцию

    // функция возводит в квадрат переданный параметр
```

```
$result = array_map($fnSquare, $nums);

// выводим результат

echo "<pre>";

print_r($result); // 1,4,9,16,25

?>
```

Ну, и не забываем, анонимная функция может, как **наследовать**, так и получать **доступ** к переменным глобального уровня видимости.

example_15. Наследование в анонимных функциях

```
<?php

// иницилируем исходный массив

$tags = array("php", "laravel", "wordpress", "mysql",
"free");

// внешняя переменная

$grid = "#";

// определяем callback-функцию

$fnSquare = function ($key) use($grid) {

    return $grid . $key;

};

// в качестве параметра передаем callback-функцию

$result = array_map($fnSquare, $tags);

// выводим результат

echo "<pre>";

print_r($result); // конечный массив

?>
```

Подведем итоги

Функция - это специальным образом оформленный именованный фрагмент кода, к которому можно многократно обратиться из любого места внутри программы. Функция не будет выполняться сразу при загрузке страницы. Функция будет выполняться при ее **вызове** из тела программы.

Информация может передаваться внутрь функции:

- через **параметры** (могут передаваться по **значению** и по **ссылке**);
- через ключевое слово **global**.

Определение пользовательской функции задаёт **локальную** область видимости данной функции. Любая используемая внутри функции переменная по умолчанию ограничена локальной областью видимости функции. К таким переменным можно обратиться только **изнутри данной функции**.

Функция может принимать в качестве параметров и возвращать в качестве результата своей работы любой используемый в PHP тип данных.

Если функция не использует оператор **return**, она все равно возвращает значение, только в этом случае это значение - **Null**.

PHP позволяет создавать **анонимные функции**, такие функции не имеют определенных имен. Синтаксис определения анонимной функции не отличается от синтаксиса определения обычной функции за тем исключением, что она не имеет имени.

Анонимные функции наиболее полезны в качестве **значений параметров** других функций.

Анонимные функции, которые могут наследовать переменные из родительской области видимости, называют **замыканиями**.

Анонимная функция может принимать **внешние переменные**:

- через параметры (по значению, по ссылке);
- через наследование (по значению, по ссылке);
- глобально.

Стрелочная функция - краткая форма записи анонимной функции (имеет особенность - прямой доступ к глобальным переменным)

Функции обратного вызова - анонимные функции, переданные в качестве **параметра** другой функции.

Задание 65

=====

В директории раздаточного материала представлен файл **index.php**. Вывод данных о преподавателях осуществляется с использованием **стандартной функции fnOutPersonnel ()**.

Ревьюируйте код сценария таким образом, чтобы вывод данных осуществлялся с помощью **анонимной функции** принимающей входной массив в качестве параметра.

Задание 66

=====

В директории раздаточного материала представлен файл **index.php**. Вывод данных о преподавателях осуществляется с использованием **стандартной функции fnOutPersonnel ()**.

Ревьюируйте код сценария таким образом, чтобы вывод данных осуществлялся с помощью **замыкания**, наследующего входной массив в выражении **use**.

Задание 67

=====

Проанализируйте сценарий файла **index.php** раздаточного материала. Код сценария, используя функцию **array_map**, выводит альбомы, имеющие в числе прочих статус "**Серебряный**".



Анонимные функции

```
array(10) {  
  [0]=>  
  NULL  
  [1]=>  
  NULL  
  [2]=>  
  array(6) {  
    ["id"]=>  
    string(1) "3"  
    ["album_name"]=>  
    string(18) "Obscured by Clouds"  
    ["date"]=>  
    string(15) "3 июня 1972"  
    ["label"]=>  
    string(21) "EMI, Harvest, Capitol"  
    ["format"]=>  
    string(22) "LP, кассета, CD"  
    ["status"]=>  
    string(48) "Золотой (USA), Серебряный (GBR)"  
  }  
  [3]=>  
  NULL  
  [4]=>  
  NULL  
  [5]=>  
  NULL  
  [6]=>  
  NULL  
  [7]=>  
  NULL  
  [8]=>  
  NULL  
  [9]=>  
  NULL  
}
```

Проблема вывода заключается в том, что функция **array_map** осуществляет перебор по всем значениям исходного массива. И если **callback-функция** не вернула результат, результатом будет значение **NULL**. Это не совсем то, что хотелось бы видеть в конечном массиве.

Выходом может быть заполнение **глобального** по отношению к **array_map** массива. Ревьюируйте сценарий, таким образом, чтобы результирующий массив не содержал избыточные NULL данные. Используйте глобальный массив.

Глобальный массив передавайте в **array_map**:

- Используя **use**.
- Используя **global**.

Задание 68

=====

В файле **index.php** раздаточного материала представлен скрипт вывода двумерного ассоциативного массива **\$albums**. Обратите внимание, значения некоторых ключей массива хранят HTML-теги списка.

Ревьюируйте код скрипта:

- Для вывода данных используйте возможности функции **array_map**.
- При выводе в браузер данных массива **\$albums** избавьтесь от HTML-тегов списка. Для удаления тегов используйте функцию **strip_tags()**.