
# РАЗРАБОТКА WEB-ПРИЛОЖЕНИЙ НА PHP&MYSQL

## УРОК 3

### Основные функции языка

# ВВЕДЕНИЕ

В прошлый раз, когда мы познакомились с принципами написания функций в **PHP**, было сказано, что в **PHP** имеется богатый выбор стандартных функций. Пришло время познакомиться с некоторыми – самыми необходимыми – из них:

- Функции для работы с массивами;
- Математические функции;
- Функции для работы со строками.
- Функции для работы с датой и временем;

# 1 ФУНКЦИИ ДЛЯ РАБОТЫ С МАССИВАМИ

## 1.1 Сортировка массивов

Начнем с самого простого — сортировки массивов. В **PHP** для этого существует много функций. Можно сортировать и ассоциативные массивы и списки в любом порядке.

### 1 Сортировка списка sort() / rsort()

```
bool sort (array &array [, int sort_flags])
bool rsort (array &array [, int sort_flags])
```

Это основные функции сортировки, при помощи которых хорошо сортировать **списки** (т.е. массивы с исключительно числовыми индексами (см. урок 2)). Функция `sort()` сортирует значения списка по возрастанию, а `rsort()` – по убыванию. Нечисловые элементы сортируются по **ASCII**-кодам в алфавитном порядке. Например:

**Пример 1.1.1**

```
<?php
$A=array("One", "Two", "Three", "Four");
sort($A);
for($i=0; $i<count($A); $i++) echo "$i:$A[$i] ";
// выводит "0:Four 1:One 2:Three 3:Two"
?>
```

Поскольку любой ассоциативный массив воспринимается этими функциями как список, то, после упорядочивания последовательность ключей превращается в 0, 1, 2, … а значения перераспределяются. Т.е. связи между парами ключ=>значение теряются, поэтому сортировать ассоциативные массивы этими функциями нельзя.

Необязательный параметр sort_flags позволяет устанавливать опции сортировки:

- SORT_REGULAR – сравнивать элементы, не меняя типов
- SORT_NUMERIC – сравнивать элементы как числа
- SORT_STRING – сравнивать элементы как строки
- SORT_LOCALE_STRING – сравнивать элементы, основываясь на текущей локали.

### 2 Сортировка массива по значениям (asort() / arsort())

```
bool asort (array &array [, int sort_flags])
bool arsort (array &array [, int sort_flags])
```

Функция asort() сортирует по возрастанию, но оставляет связь между ключами и значениями в ассоциативном массиве. Например:

**Пример 1.1.2**

```
<?php
$A=array("a"=>"Zero","b"=>"Null","c"=>"Empty","d"=>"Space");
asort($A);
foreach($A as $k=>$v) echo "$k=>$v ";
// выводит "c=>Empty b=>Null d=>Space a=>Zero"
?>
```

Функция arsort() сортирует по убыванию.

### 3 Сортировка по ключам (ksort() / krsort())

```
bool ksort (array &array [, int sort_flags])
bool krsort (array &array [, int sort_flags])
```

Эти функции сортируют не по значениям, а по ключам, сохраняя их связь со значениями. ksort() – по возрастанию, а krsort() – по убыванию. Например:

**Пример 1.1.3**

```
<?php
$langs = array("d"=>"coldfusion", "a"=>"php", "b"=>"perl",
"c"=>"python");
ksort($langs);
reset($langs);
while (list($key, $val) = each($langs)) {
echo "$key = $val\n";
}
?>
/*
a = php
b = perl
c = python
d = coldfusion
*/
```

### 4 Функция natsort()

Данная функция позволяет избавиться при сортировке строковых данных, содержащих числа, от следующего эффекта:

```
doc1
doc10
doc2
doc20
doc21
```
Отсортировано по возрастанию

Функция `natsort()` применяет натуральный алгоритм сортировки.

```
bool natsort (array &array);
```

### 5 Сортировка при помощи пользовательской функции

Довольно часто нам приходится сортировать что-то по более сложному критерию, чем просто по алфавиту. Допустим, существует необходимость отсортировать массив, в котором хранятся названия дней недели на русском языке.

Учитывая, что это просто строки, в этом случае нам стоит воспользоваться функцией `usort()`, написав предварительно функцию сравнения с двумя параметрами, как того требует `usort()`.

```
bool usort (array array, callback cmp_function)
```

`cmp_function` – это пользовательская функция сравнения, кото-

рая должна сравнивать два значения между собой и возвращать:

-1, если первое значение меньше второго,

0, если значения равны

1, если первое значение больше второго

Эта функция призвана перебирать попарно элементы массива, переданного в первом параметре – array.

**Пример 1.1.4**

```
<?php
$a=array("Понедельник", "Пятница", "Среда", "Четверг", "Сре-
да"); //Исходный массив
usort($a, "weekday_ru_sort");
//сортирует массив собственная функция
foreach ($a as $key=>$value) {
//перебор и вывод отсортированного массива
echo $key.": ".$value."<br />";
}
function weekday_ru_sort($a, $b) // собственно сама функция
{
$weekdays_ru=array(
    1=>"Понедельник", 2=>"Вторник",
    3=>"Среда", 4=>"Четверг",
    5=>"Пятница", 6=>"Суббота",
    7=>"Воскресенье",
);

$a_key=0;
$b_key=0;
foreach ($weekdays_ru as $key=>$value)
{
    if ($a==$value) $a_key=$key;
    if ($b==$value) $b_key=$key;
}
if ($a_key==$b_key) return 0;
elseif ($a_key<$b_key) return -1;
else return 1;
}
?>
```

В результате будет текст:

```
0: Понедельник
1: Среда
2: Среда
3: Четверг
4: Пятница
```

Сортировка происходит по значениям, при этом связь между значениями и ключами не сохраняется по аналогии с `sort()`.

Наряду с `usort()`, пользовательскую сортировку выполняют также функции `uasort()` и `uksort()`.

Функцией `uksort()` связи между ключами и значениями сохраняются, но сортировка происходит по ключам.

Функция `uasort()` очень похожа на `uksort()`, с той разницей, что пользовательской функции сортировки "подсовываются" не ключи, а очередные значения из массива. При этом также сохраняются связи значений и ключей.

## 1.2 Перемещение по массиву и поиск в массиве.

### 1 Проверка существования значения или ключа.

Функция `in_array()` проверяет, существует ли в массиве указанный элемент.

```
bool in_array(mixed value, array search [, bool strict])
```

`value` – это то, что мы ищем, а `search` – это имя исследуемого массива. Если указать необязательный параметр `strict` как `true`, то при поиске будет учитываться и тип искомого элемента.

**Пример 1.2.1**

```
<?php
$a = array('1.10', 12.4, 1.13);
if (in_array('12.4', $a, true)) {
    echo "'12.4' найдено по строгой проверке";
    }
if (in_array(1.13, $a, true)) {
    echo "1.13 найдено по строгой проверке";
    }
?>
//Будет выведено:
//1.13 найдено по строгой проверке
```

Функция `array_key_exists()` проверяет существование указанного ключа.

```
bool array_key_exists (mixed key, array search)
```

`key` – искомый ключ, `search` – исследуемый массив.

### 2 Поиск в массиве

Поиск в массиве осуществляется при помощи функции `array_search()`, которая возвращает ключ элемента, если он найден и `null` в случае неудачи.

```
mixed array_search (mixed value, array search [, bool strict])
```

Аргументы и принцип действия функции похожи на `in_array()`.

### 3 Перемещение по массиву

Для перебора массива мы используем конструкцию `foreach`, с которой мы познакомились в прошлом уроке. Однако для перемещения по массиву можно пользоваться и некоторыми другими функциями. В некоторых случаях это гораздо рациональнее.

Массив, по сути, это набор элементов, при переборе которого существует указатель `pointer`, определяющий элемент, с которым можно работать в данный момент.

Основные функции для «ручного» обхода массива имеют довольно «говорящие» названия: `reset()`, `end()`, `prev()`, `next()`, `current()`.

**Пример 1.2.2**

```
<?php
$a=array("PHP", "C", "C++", "Java", "Ruby");
echo "языки программирования: ";
reset($a);
while (current($a)) {
    echo current($a)." ";
    next($a);
}
?>
```

Для того, чтобы пакетно обработать весь массив при помощи какой-либо функции, т.е. применить какие-либо действия каждому элементу массива, нужно использовать функцию `array_walk()`:

bool **array_walk**(array **&arr**, callback **func**[, mixed **userdata**])

К каждому элементу массива arr будет применена функция с именем func. Функция должна принимать два параметра: первый – это значение элемента массива, а второй – его ключ. При необходимости, кроме значения и ключа элемента, функция func может использовать некоторые данные, передаваемые в неё при помощи userdata. Вернет true или false – в случае ошибки.

**Пример 1.2.3**

```
<?php
$fruits=array("d"=>"lemon",   "a"=>"orange",   "b"=>"banana",
"c"=>"apple");
function test_alter(&$item1, $key, $prefix) {
    $item1="$prefix: $item1";
}
function test_print($item2, $key) {
    echo "$key. $item2<br />\n";
}
echo "Before ...:\n";
array_walk($fruits, 'test_print');
array_walk($fruits, 'test_alter', 'fruit');
echo "... and after:\n";
array_walk($fruits, 'test_print');
?>
/* Вывод:
Before ...:
d. lemon
a. orange
b. banana
c. apple
... and after:
d. fruit: lemon
a. fruit: orange
b. fruit: banana
c. fruit: apple
*/
```

### 4 Подсчет элементов в массиве

В массивах можно подсчитать не только количество входящих в него элементов, но и количество вхождений некоторого значения, и

сумму всех элементов.

Функция `count()` считает количество элементов в массиве:

```
int count (mixed var[, int mode])
```

`var` – подсчитываемый массив. `mode` – флажок для рекурсивного обхода многоуровневого массива (значение `1`). По-умолчанию – значение `0`.

Функция `array_count_values()` подсчитывает количество вхождений определенного значения в массив и возвращает массив с количеством вхождений:

```
array array_cont_values (array input)
```

**Пример 1.2.4**

```
<?php
$array=array (1, "hello", 1, "world", "hello");
print_r(array_count_values ($array));
?>
/* Результат
Array
(
   [1] => 2
   [hello] => 2
   [world] => 1
) */
```

Для подсчёта суммы всех элементов массива служит функция `array_sum()`:

```
number array_sum (array ar)
```

## 1.3 Слияние и разделение массивов

При помощи функции `array_chunk()` можно разрезать массив на мерные кусочки, размер которых передаётся в функцию в качестве аргумента:

```
array array_chunk (array input, int size[, bool keys])
```

`input` – разрезаемый массив, `size` – размер «кусочков», `keys` – флажок, включающий опцию сохранения ключей.

Функция `array_merge()` сливает массивы:

```
array array_merge(array ar1[, array ar2[, array ar3 …]])
```

Для слияния массивы необязательно должны иметь одинаковую структуру:

**Пример 1.3.1**

```
<?php
$a1=array("color"=>"red", 2, 4);
$a2=array("a", "b", "color"=>"green", "shape"=>"circle", 4);
$result = array_merge($a1, $a2);
echo "<pre>";
print_r($result);
echo "</pre>";
?>
/*
Array
(
    [color] => green
    [0] => 2
    [1] => 4
    [2] => a
    [3] => b
    [shape] => circle
    [4] => 4
)
*/
```

Аргументами этой функции могут быть только массивы.

Функция `array_combine()` соединяет два массива, причем элементы одного из них становятся ключами, а элементы второго - значениями:

```
array array_combine(array keys, array values)
```

Количество элементов в массивах должно совпадать.

**Пример 1.3.2**

```
<?php
$a=array('fio', 'role', 'department');
$b=array('Ivanenko Helen', 'chief', 'sales department');
$c=array_combine($a, $b);
print_r($c);
?>
/* Выводит:
```

```
Array
(
    [fio] => Ivanenko Helen
    [role] => chief
    [department] => sales department
)*/
```

Функция `array_slice()` возвращает часть ассоциативного массива, начиная с пары `ключ=>значения` с номером от начала и определенной длиной (если последний параметр не задан, то до конца массива).

```
array array_slice(array Arr, int offset [, int len])
```

Параметры `offset` и `len` – это смещение и длина соответственно. Они могут быть отрицательными (в этом случае отсчет осуществляется от конца массива):

**Пример 1.3.3**

```
$input=array("a", "b", "c", "d", "e");
$output=array_slice($input, 2); // "c", "d", "e"
$output=array_slice($input, 2, -1); // "c", "d"
$output=array_slice($input, -2, 1); // "d"
$output=array_slice($input, 0, 3); // "a", "b", "c"
```

## 1.4 Вставка/удаление элементов массива

В этом разделе описаны 4 функции с аналогами которых вы должны быть знакомы из курса **JavaScript**.

```
int array_push (array &Ar, mixed vr1[, mixed vr2, …])
```

Эта функция дописывает в конец массива `Ar` новые элементы `vr1`, `vr2` и т.д. с новыми числовыми индексами. Если нужно добавить один элемент, то лучше воспользоваться `[]`. Функция возвращает новое число элементов в массиве.

```
int array_unshift (array &Ar, mixed vr1[, mixed vr2, …])
```

Функция дописывает перечисленные элементы не в конец, а в начало массива, сохраняя их порядок, т.е. как бы внедряя их слева. Функция переназначает всем элементам массива новые числовые индексы, увеличивая их на число добавленных элементов. Функция возвращает новый размер массива.

**Пример 1.4.1**

```
<?php
$A=array(10, "a"=>20, 30);
array_unshift($A, "!", "?");
//теперь $A===array(0=>"!", 1=>"?", 2=>10, "a"=>20, 3=>30);
?>
```

```
mixed array_pop(array &Arr)
```

Функция выбирает последний элемент из списка и возвращает его, удалив после этого его из `Arr`. Если список `Arr` был пуст, функция возвращает пустую строку.

```
mixed array_shift(array &Arr)
```

Эта функция извлекает первый элемент массива `Arr` и возвращает его. Она корректирует все числовые индексы у всех оставшихся элементов.

**Пример 1.4.2**

```
<?php
$stack=array("orange", "banana", "apple", "raspberry");
$fruit=array_shift($stack);
print_r($stack);
?>
/* В результате в массиве $stack останется 3 элемента:
Array
(
    [0] => banana
    [1] => apple
    [2] => raspberry
)
*/
```

## 1.5 Другие полезные функции для работы с массивами

Функция `array_reverse()` переворачивает массив, меняя местами «хвост» и «голову». Связь между ключами и значениями при этом сохраняется.

Функция `array_flip()` меняет местами его ключи и значения. При этом, исходный массив не изменяется, а результирующий массив просто возвращается. Если массив имел несколько одинаковых значений, то в итоге ключом станет последнее встреченное в масси-

ве.

Функция `array_unique()` возвращает массив уникальных значений с ключами, встреченными в переданном в параметре массиве.

**Пример 1.4.4**

```
<?php
$input=array("a"=>"green",   "red",   "b"=>"green",   "blue",
"red");
$result=array_unique($input);
// теперь $result===array("a"=>"green", "red", "blue");
?>
```

Функция `compact()` создает массив из переменных, передаваемых в параметрах.

```
array compact (mixed vnl [, mixed vn2, …])
```

Имена переменных становятся ключами, а их значения - значениями элементов массива.

**Пример 1.4.5**

```
<?php
$a="Test string";
$b="Some text";
$A=compact("a","b");
// теперь $A===array("a"=>"Test string", "b"=>"Some text")
?>
```

Функция `extract()` противоположна `compact()`. Т.е. получает в параметрах массив и превращает каждую его пару `ключ=>значение` в переменную текущего контекста.

```
void extract(array Arr[, int type] [, string prefix])
```

Параметр `type` определяет действия в том случае, если в текущем контексте уже существует переменная, подлежащая созданию. Он может быть равен одной из следующих констант:

- `EXTR_OVERWRITE` – перезаписывать (по умолчанию);
- `EXTR_SKIP` – не перезаписывать, если существует;
- `EXTR_PREFIX_SAME` – в случае совпадения имен создавать переменную с именем, начинающимся префиксом из параметра `prefix`.
- `EXTR_PREFIX_ALL` – все создаваемые переменные оснащать префиксом `prefix`.

# 2 МАТЕМАТИЧЕСКИЕ ФУНКЦИИ

**PHP** обладает всеми необходимыми возможностями в плане математических вычислений, хотя они и используются нечасто. Все таки **web** – это не та сфера, где математические вычисления необходимы.

Рассмотрим наиболее полезные функции.

## 2.1 Встроенные константы

Существует ряд предопределенных математических констант (полный список см. в **документации**). Ниже список наиболее употребимых.

| Константа | Значение | Описание |
|---|---|---|
| M_PI | 3.14159265358979323846 | Pi |
| M_E | 2.71828182845904523536 | e |
| M_PI_2 | 1.57079632679489661923 | pi/2 |
| M_PI_4 | 0.78539816339744830962 | pi/4 |
| M_SQRT2 | 1.41421356237309504880 | sqrt(2) |
| M_SQRT3 | 1.73205080756887729352 | sqrt(3) [4.0.2] |

## 2.2 Функции округления

```
double round(double val)
```

Округляет по правилам арифметики

```
int ceil(float number)
```

«Округляет» в большую сторону, т.е. до ближайшего большего целого.

```
int floor(float number)
```

То же самое, только – в меньшую сторону.

## 2.3 Случайные числа

Применение случайных чисел в **web**-программировании достаточно широкое. Как для декоративных целей (показ случайного изображения или баннера), так и для целей безопасности (генерация каких-то случайных паролей или других элементов).

Для генерации случайных чисел лучше всего использовать следующие три функции:

```
int mt_rand(int min=0, int max=RAND_MAX)
void mt_srand(int seed)
int mt_getrandmax()
```

Первая из них возвращает вполне равномерное случайное число в пределах от `min` до `max`. Как видно, по-умолчанию, числа генерируются в пределах от нуля до максимального возможного случайного числа, задаваемого константой `RAND_MAX`.

Вторая – настраивает генератор случайных чисел на новую последовательность. Хотя числа, генерируемые `mt_rand()`, достаточно равновероятны, но, тем не менее, последовательность сгенерированных чисел будет одинакова, если сценарий вызвать несколько раз подряд. Функция `mt_srand()` выбирает новую последовательность на основе параметра `seed`. В качестве параметра можно взять временной параметр, поскольку он, меняясь, позволит поддерживать истинную случайность естественным образом. Например:

**Пример 2.3.1**

```
<?php
mt_srand(time()+(double)microtime()*1000000);
$randval=mt_rand();
?>
```

В этом случае последовательность устанавливается на основе времени запуска сценария (в микросекундах).

Функции работы с датой и временем, использованные в данном примере, будут рассмотрены в следующем уроке.

И, наконец, третья – возвращает максимальное число, которое может быть сгенерировано функцией `mt_rand()` – иными словами, константу `RAND_MAX`.

## 2.4 Перевод в различные системы счисления

```
string base_convert(string number, int frombase, int tobase)
```

Переводит число `number` в строковом представлении в системе счисления по основанию `frombase` в систему по основанию `tobase`. Работает с системами счисления только от `2` до `36` включительно. В строке `number` цифры обозначают сами себя, буква `a` соответствует `11`, `b` — `12`, и т. д. до `z`, которая обозначает `36`. Например:

**Пример 2.4.1**

```
<?php
echo base_convert("FF",16,2);
?>
```

Будет выведено `11111111` (`8` единичек), потому что это представление шестнадцатеричного числа `FF` в двоичной системе счисления.

```
int bindec(string binary_string)
```

Двоичное число `binary_string` преобразуется в десятичное.

```
string decbin(int number)
```

Наоборот – из двоичного в десятичное. Максимальное число, которое еще может быть преобразовано, равно `2147483647`, которое выглядит как `31` единичка в двоичной системе.

## 2.5 Минимум и максимум

```
mixed min (mixed arg1 [int arg2, ..., int argn])
mixed max(mixed arg1 [int arg2, ..., int argn])
```

Эти функции возвращают наименьшее (или наибольшее) из чисел, заданных в ее аргументах.

Если указан только первый параметр, то он обязательно должен быть массивом, то возвращается максимальный (или минимальный) элемент этого массива. В противном случае все аргументы рассматриваются как числа с плавающей точкой.

Если хотя бы одно из переданных чисел – с плавающей точкой, то и результат будет с плавающей точкой, иначе – целым.

## 2.6 Другие математические функции

```
float sqrt(float arg)
```

Возвращает квадратный корень из аргумента. При попытке извлечения корня из отрицательного числа генерируется предупреждение, но работа программы не прекращается!

```
float log(float arg)
```

Возвращает натуральный логарифм аргумента. В случае недопустимого числа также генерирует предупреждение, но, как и `sqrt()`, не завершает программу.

```
float exp(float arg)
```

Возвращает `e` (`2,718281828`) в степени `arg`.

```
float pow(float base, float exp)
```

Возвращает `base` в степени `exp`.

```
float acos(float arg)
float asin(float arg)
float atan (float arg)
float sin(float arg)
float cos(float arg)
float tan(float arg)
```

Названия этих функций говорят, в общем-то, сами за себя. Следует напомнить только, что расчёты ведутся в радианах.

# 3 ФУНКЦИИ ДЛЯ РАБОТЫ СО СТРОКАМИ.

Для чего создаются **web**-приложения? Для создания возможности общения людей при его помощи. А это связано, в первую очередь, с тем, что скрипты должны обрабатывать огромное количество строковой информации, полученной в виде пользовательского ввода в **web**-формы. А пользователи, это такие удивительные люди, которые не всегда вводят в формы то, что от них требовалось. Например, самая большая клавиша на клавиатуре, как известно – пробел. Вот и нажимают её пользователи чаще всего и, как правило, в самых ненужных местах. ☺

Как результат, разработчик должен иметь мощный инструментарий для обработки со строк. Именно об этом и поговорим в этой главе.

Вообще следует сказать, что в **PHP** существует два основных подхода к процессу обработки строк. Первый – это встроенные строковые функции, а второй – это использование **регулярных выражений**. Сегодня речь пойдет о первом способе, а второй способ рассмотрим в 6-м уроке.

## *3.1 Вывод строк на экран. Конкатенация*

Все что выводится в браузер так или иначе представляет собой строку, поэтому чаще всего мы сталкиваемся с выводом строк и необходимостью их слияния, т.е. конкатенации.

С самой простой функцией вывода мы уже знакомы (см. 1-й урок). Это – `echo`. Следует отметить, что это, в принципе, и не функция, а конструкция языка, поэтому записывается без скобок.

Альтернативно можно использовать функции:

```
string sprintf(string format [, mixed args, …])
void printf(string format [, mixed args, …])
```

Эти функции конструируют строку, заменяя в строке `format` некоторые специальные символьные последовательности (см. таблицу ниже) на значения, которые берутся из переменных, передаваемых в необязательных параметрах `args`.

Специальные символьные последовательности – команды форматирования – предваряются знаком %. В нашем распоряжении имеются следующие команды:

| Символ | Соответствие |
|---|---|
| b | бинарное число |
| c | символ с указанным в аргументе ASCII-кодом |
| d | целое число |
| f | число с плавающей точкой |
| o | восьмеричное целое число |
| s | строка символов |
| x | шестнадцатеричное число с маленькими буквами a—f; |
| X | шестнадцатеричное число с большими буквами A—F |

Если же нужно поместить в текст % как обычный символ, необходимо его удвоить:

**Пример 3.1.1**

```
<?php
echo sprintf ("The percentage was %d%%", $percentage);
?>
```

Первая из представленных функций (sprintf) возвращается отформатированную строку, а вторая (printf) – выводит её непосредственно в браузер, как echo.

**Пример 3.1.2**

```
<?php
$name="John";
$age=26;
echo sprintf("My name is %s. I'm %d years old",$name, $age);
?>
```

В отличие от **JavaScript**, в **PHP** для конкатенации (сложения) строк существует специальный оператор "." (точка), отличный от оператора арифметического сложения "+".

Правда так было не всегда. В версиях **PHP** старше третьей для конкатенации, как и для сложения чисел, использовался арифметический оператор "+", что приводило к путанице.

Оператор "." всегда воспринимает свои операнды как строки и возвращает строку. В случае если один из операндов не может быть

представлен как строка, т.е. если это массив или объект, то он воспринимается как строки `array` и `object` соответственно.

Вообще говоря, это правило применимо и не только при сцеплении строк, но и при передаче такого операнда в какую-нибудь стандартную функцию, которой требуется строка.

В наиболее общих случаях, например, когда переменные, хранящие строковые значения, нужно вывести в браузер, перемежая обычным текстом (в том числе и html-разметкой), лучше всего воспользоваться функцией `echo`. Если, к примеру, у нас в `$day` хранится текущее число, в `$month` — название месяца и в `$year` — год, то вывести строку вида "Сегодня 8 мая 2010 года" можно так:

**Пример 3.1.3**

```
<?php echo "Сегодня $day $month $year года"; ?>
```

Такое замечательное поведение конструкции `echo` в **PHP** достигается благодаря тому, что все переменные начинаются с `$`.

## 3.2 Базовые функции. Поиск и замена

```
int strlen(string st)
```

Возвращает длину строки, т.е. число символов в ней. Обработает даже пустую строку. Вернет при этом нуль, конечно же.

```
int strpos(string where, string what, int fromwhere=0)
```

Возвращает позицию первого вхождения подстроки `what` в строке `where`. Необязательный параметр `fromwhere` можно задавать, если поиск нужно вести не с начала строки, а с какой-то другой позиции. В случае неудачи, функция возвращает `false`.

```
int strrpos (string where, char what)
```

Работа этой функции напоминает предыдущую, но есть два отличия:

- ищется последнее, а не первое вхождение.
- искомый параметр (`what`) – это символ, а не строка. Даже если передана будет строка, то браться во внимание будет только первый символ.

```
string substr(string str, int from [,int length])
```

Предыдущие функции отвечали на вопрос: где находится под-

строка? Эта функция отвечает на вопрос: что находится в позиции такой-то? Она возвращает участок строки `str`, начиная с позиции `start` длиной `length`. Если `length` не задана, то подразумевается подстрока от `start` до конца строки `str`. Если `start` больше, чем длина строки, или же значение `length` равно нулю, то возвращается пустая строка. Если `start` – отрицательное число, то индекс подстроки будет отсчитываться с конца `str` (например, `-1` означает "начиная с последнего символа строки").

**Пример 3.2.1**

```
<?php
$rest = substr("abcdef", -1); // возвращает "f"
$rest = substr("abcdef", -2); // возвращает "ef"
$rest = substr("abcdef", -3, 1); // возвращает "d"
?>
```

Параметр `length` тоже может быть отрицательным. Это означает, что последний символ возвращенной подстроки – это символ из `str` с индексом `length`, определяемым от конца строки.

**Пример 3.2.2**

```
<?php
$rest = substr("abcdef", 0, -1); // возвращает "abcde"
$rest = substr("abcdef", 2, -1); // возвращает "cde"
$rest = substr("abcdef", 4, -4); // возвращает ""
$rest = substr("abcdef", -3, -1); // возвращает "de"
?>
```

```
string str_replace(string from, string to, string str [, int count])
```

Как следует из названия, функция позволяет осуществить глобальную замену в строке. Она заменит либо все вхождения строки `from` в строке `str` на строку `to`, либо ограничится `count` заменами, если этот параметр передан. Исходная строка не изменяется. Результат будет возвращен этой функцией.

**Пример 3.2.3**

```
<?php
// присваивает <body text='black'>
$btag=str_replace("%body%", "black", "<bodytext='%body%'>");
?>
```

```
int strcmp(string str1, string str2)
```

Посимвольно сравнивает две строки и возвращает `0` при их совпадении; `-1`, если строка `str1` лексикографически меньше `str2`; и `1`, если, наоборот, `str1` "больше" `str2`. Так как сравнение идет побайтово, то оно регистрозависимо.

```
string chr(int code)
```

Возвращает строку из одного символа с кодом `code`. Эта функция полезна для вставки каких-либо непечатаемых символов в строку — например, нулевого кода или символа прогона страницы, а также при работе с бинарными файлами.

```
int ord (char ch)
```

Эта функция, наоборот, возвращает код символа переданного в в `ch`. Например, `ord(chr($n))` всегда равно `$n`.

## 3.3 Функции отрезания пробелов

Под пробельными символами побудем понимать:

- пробел `" "`
- символ перевода строки `\n`
- символ возврата каретки `\r`
- и символ табуляции `\t`.

Рассмотрим функции, позволяющие удалять пробельные символы.

```
string trim(string st)
```

Удаляет все пробелы.

```
string ltrim(string st)
string rtrim(string st)
```

То же, что и `trim()`, только удаляет только ведущие или только концевые пробелы соответственно.

## 3.4 Функции для работы с HTML

Гипертекстовая разметка – это, по сути, главный продукт любого **PHP**-сценария. По этой причине, имеет смысл рассмотреть ряд функций, работающих именно с html-разметкой.

```
string nl2br(string string)
```

Позволяет оптимизировать некий многострочный текст к выводу в браузер за счет замены в нём всех символов новой строки \n на <br>\n. Исходная строка не изменяется. Обратите внимание на то, что символы \r, которые присутствуют в конце строки текстовых файлов **Windows**, этой функцией никак не учитываются, а потому остаются на старом месте.

```
string strip_tags (string str [, string allowable_tags])
```

Эта функция удаляет из строки все тэги и возвращает результат. В параметре allowable_tags можно передать тэги, которые не следует удалять из строки. Они должны перечисляться вплотную друг к другу без разделителей. Вот пример:

**Пример 3.4.1**

```
<?php
$st="
<b>Жирный текст</b>
<tt>Моноширинный текст</tt>
<a href='http: //www.google.com'>Ссылка</a>";
echo "Исходный текст: $st";
echo "<br>После удаления тэгов: ".strip_tags($st, "<a><b>")
."<br>";
?>
```

Запустив этот пример, мы сможем заметить, что тэги <a> и <b> не были удалены (равно как и их парные закрывающие), в то время как <tt> исчез.

```
string Htmlentities(string str [int quote_style [, string charset]])
```

Позволяет безопасно подготовить к выводу в браузер строк с любым содержимым, поскольку преобразует специальные символы (<, >, &, одинарные и двойные кавычки, например) в соответствующие **HTML**-сущности. Параметр quote_style задаёт стиль преобразования кавычек. Параметр charset позволяет указать кодировку. Возможные значения параметра quote_st:

| Константа | Описание |
|---|---|
| ENT_COMPAT | Преобразуются только двойные кавычки |
| ENT_QUOTES | Преобразуются и двойные и одинарные кавычки |
| ENT_NOQUOTES | И двойные и одинарные кавычки остаются без изменений |

**Пример 3.4.2**

```
<?php
$str = "A 'quote' is <b>bold</b>";
// выводит: A 'quote' is &lt;b&gt;bold&lt;/b&gt;
echo htmlentities($str);
// выводит: A &#039;quote&#039; is &lt;b&gt;bold&lt;/b&gt;
echo htmlentities($str, ENT_QUOTES);
?>
```

```
string HtmlSpecialChars(string str)
```

Эта функция делает почти то же ,что и предыдущая, но она преобразует в строке только те символы, которые участвуют в разметке (см. таблица).

| Символ | Преобразование |
|---|---|
| & | &amp; |
| " | &quot; |
| ' | &#039; |
| < | &lt; |
| > | &gt; |

Чаще всего, эта функция применяется для вывода атрибутов web-форм, чувствительных к кавычкам и прочим «разметочным» символам, а также для вывода примеров разметки в каких-либо on-line пособиях, обсуждениях на форумах и т.д.

## 3.5 Конвертация кодировок

Кодировки очень актуальны для нас – носителей языка, основанного на кириллице. Существует, как известно, несколько кодировок кириллицы (см. таблицу). Часто встречается ситуация, когда нам требуется преобразовать строку из одной кодировки кириллицы в другую.

```
string convert_cyr_string(string str, char from, char to)
```

Функция переводит строку `str` из одной кириллической кодировки `from` в другую кириллическую кодировку `to`. Поскольку латиница во всех кириллических кодировках выглядит одинаково, то это

имеет смысл только для текста на самом деле содержащего кириллические символы. Разумеется, кодировка `from` должна совпадать с истинной кодировкой строки, иначе результат получится неверным. Значения `from` и `to` — один символ, определяющий кодировку согласно таблицы:

| Символ | Кодировка |
|---|---|
| k | KOI8-R |
| w | Windows-1251 |
| i | ISO-8859-5 |
| a | x-cp866 |
| m | x-mac-cyrillic |

Для преобразования кодировок вообще, а не только кириллических, служит функция:

```
string iconv (string in_chs, string out_chs, string str)
```

Исходная кодировка – `in_chs`, результирующая – `out_chs`, строка для преобразования – `str`.

**Пример 3.5.1**

```
<?php
echo iconv("ISO-8859-1", "UTF-8", "This is a test.");
?>
```

## 3.6 Хэш-функции

В целях безопасности, очень часто текст приходится кодировать или шифровать. В рамках курса «PHP», конечно же, мы не будем подробно изучать криптографию, но познакомиться с некоторыми функциями, предназначенными для этого, стоит.

Очень распространено использование алгоритма шифрации корпорации **RSA Data Security** под названием "**MD5 Message-Digest Algorithm**". Он позволяет получить практически уникальную строку для любой, к которой он применен. Такая строка называется – хэш-код.

```
string md5 (string st)
```

Возвращает хэш-код строки `st`.

Алгоритм **MD5** очень широко применяется для проверки паро-

лей на истинность. Хранятся не сами пароли, а их **MD5**-коды (хэши). При попытке какого-либо пользователя войти в систему вычисляется хэш-код только что введенного им пароля и сравнивается с тем, который записан в базе данных. Если коды совпадут, значит пароль введен верно, а если нет – то попытка входа будет отклонена.

**Пример 3.6.1**

```
<?php
$str = 'apple';
if (md5($str) === '1f3870be274f6c49b3e31a0c6728957f') {
echo "Would you like a green or red apple?";
exit; }
?>
```

Обсуждая алгоритм **MD5**, нужно также упомянуть и то, что хэширование выполняется довольно медленно, причем замедление это искусственно. Это сделано специально чтобы противостоять распространенной атаке – быстрому перебору паролей.

```
int crc32(string str)
```

Вычисляет 32-битную контрольную сумму строки `str`. То есть, результат ее работы — 32-битное (4-байтовое) целое число. Эта функция работает гораздо быстрее `md5()`, но в то же время выдает гораздо менее надежные "хэш-коды" для строки.

## 3.7 Другие функции работы со строками

```
string strtolower (string str)
string strtoupper (string str)
```

Переводит строку в нижний или верхний регистр. Обе эти функции, к сожалению, не обрабатывают кириллицу.

```
string str_repeat (string st, string number)
```

Функция "повторяет" строку `st number` раз и возвращает объединенный (конкатенированный) результат.

**Пример 3.7.1**

```
<?php
echo str_repeat("test !", 3); //выводит test ! test ! test !
?>
```

```
array explode (string sep, string string [, int limit ])
```

Возвращает массив строк, полученных разбиением строки `string` с использованием `sep` в качестве разделителя. Массив может быть ограничен максимумом в `limit` элементов, при этом последний элемент будет содержать остаток строки `string`.

Если `sep` – пустая строка (""), функция возвращает `FALSE`. Если `sep` не содержится в `string`, то `explode()` возвращает массив, содержащий один элемент `string`.

**Пример 3.7.2**

```php
<?php
$pizza="piece1 piece2 piece3 piece4 piece5 piece6";
$pieces=explode(" ", $pizza);
echo $pieces[0]; //piece1
echo $pieces[1]; //piece2
?>
```

```
string implode(string glue, array pieces)
```

Возвращает строку, полученную объединением элементов массива `pieces`, со вставкой строки `glue` между соседними элементами.

**Пример 3.7.3**

```php
<?php
$array = array('lastname', 'email', 'phone');
$comma_separated=implode(",", $array);
echo $comma_separated; //lastname,email,phone
?>
```

# 4 ФУНКЦИИ ДЛЯ РАБОТЫ С КАЛЕНДАРЁМ

Огромное количество сервисов **web**-приложений основано на работе с датой и временем. В **PHP** присутствует полный набор средств, предназначенных для работы с датами и временем в различных форматах. В основе всего исчисления лежит так называемый формат **Unix timestamp**, который представляет собой количество секунд, прошедшее с **GMT** 00:00:00 1 января 1970 года. Эта дата представляет собой условное начало эпохи **Unix** и применяется не только в PHP, но и в других языках **web**-программирования.

```
int time()
```

Возвращает **Unix timestamp** текущего момента (см. выше)

```
int mktime([int h[,int m[,int s[,int mn[,int d[,int y]]]]]])
```

Функция возвращает значение **Unix timestamp**, соответствующее указанной дате.

Если опускать необязательные параметры (их названия, в общем-то, достаточно красноречивы, чтобы их комментировать), то на их место будут подставляться значения, соответствующие текущему моменту.

**Пример 4.1.1**

```
<?php
echo date("d-m-Y", mktime(0, 0, 0, 12, 32, 2009))."<br />";
echo date("d-m-Y", mktime(0, 0, 0, 13, 1, 2009))."<br />";
echo date("d-m-Y", mktime(0, 0, 0, 1, 1, 2010))."<br />";
echo date("d-m-Y", mktime(0, 0, 0, 1, 1, 10))."<br />";
?>
```

Каждый из этих вызовов конструкции `echo` выведет одно и то же: «01-01-2010».

Данная функция обладает как-бы «перемоткой» даты, автоматически добавляя «лишние» дни, месяцы, часы и т.д. в случае, если соответствующие параметры имеют некорректные числовые значения. Например, последний день любого месяца можно вычислить как "нулевой" день следующего месяца. Оба приведенных ниже

примера выведут "Последний день в феврале 2000 г: 29".

**Пример 4.1.2**

```
<?php
$ld=mktime(0, 0, 0, 3, 0, 2000);
echo strftime("Последний день в феврале 2000 г:: %d", $ld);
$ld=mktime(0, 0, 0, 4, -31, 2000);
echo strftime("Последний день в феврале 2000 г:: %d", $ld);
?>
```

Функции date() и strftime() рассмотрены ниже.

Работа с датами, по сути, заключается в преобразовании **Unix timestamp** в «человекопонятный» формат. Для этого существуют две функции.

```
string date(string format [,int timestamp])
```

Форматирует строку в соответствии с параметром format на основе параметра timestamp (если последний не задан — то на основе текущей даты). Действие этой функции похоже на действие функции sprintf(), т.е. строка формата может содержать обычный текст, перемежаемый одним или несколькими символами форматирования:

| Буква формата | Описание | Пример |
|---|---|---|
| d | Двухцифровой день месяца | 04, 27 |
| D | Три буквы наименования дня недели (англ.) | Mon, Tue |
| m | Двухцифровой номер месяца | 01, 12 |
| M | Три буквы наименования месяца (англ.) | Jul, Dec |
| Y | Четырехцифровой год | 2009 |
| y | Двухцифровой год | 79 |
| H | Часы в 24-часовом формате | 00, 23 |
| i | Двухцифровые минуты | 06, 59 |
| s | Двухцифровые секунды | 01, 59 |
| z | Номер дня в году | 144 |
| w | Номер дня недели (0 – воскресенье) | 6 |
| l | Текстовое полное название дня недели | Friday |
| S | Английский числовой суффикс | nd, th |
| A | am или pm | |

С полным перечнем символов можно ознакомиться на **странице**

**документации**. Неформатирующие символы в строке остаются без изменений.

Вот пример применения функции `date()`:

**Пример 4.1.3**

```
<?php
echo date("l dS of F Y h:i:s A");
// Tuesday 27th 2010f April 2010 07:02:31 PM
echo date("Сегодня d.m.Y");
// Сегодня 27.04.2010
?>
```

```
array getdate(int timestamp)
```

Возвращает ассоциативный массив, содержащий данные об указанном в виде **Unix timestamp** времени. В массив будут помещены следующие ключи и значения:

| Элемент массива | Описание |
|---|---|
| seconds | Секунды от 0 до 59 |
| minutes | минуты от 0 до 59 |
| hours | Часы от 0 до 23 |
| mday | День месяца от 1 до 31 |
| wday | Номер дня недели от 0 до 6 |
| mon | Номер месяца от 1 до 12 |
| year | Четырехцифровой год |
| yday | Номер дня в году от 0 до 365 |
| weekday | полное название дня недели от Sunday до Saturday |
| month | полное название месяца от January до December |

Пример:

**Пример 4.1.4**

```
<?php
$today=getdate();
echo "<pre>";
print_r($today);
echo "</pre>";
/*
Array
(
```

```
    [seconds] => 38
    [minutes] => 4
    [hours] => 19
    [mday] => 27
    [wday] => 2
    [mon] => 4
    [year] => 2010
    [yday] => 116
    [weekday] => Tuesday
    [month] => April
    [0] => 1272384278
)
*/
?>
```

Понятно, что это – альтернатива функции `date()`.

```
int checkdate(int month, int day, int year)
```

Функция нужна для проверки корректности даты, которая передана ей в виде трех аргументов. Условия корректности следующие:

- год должен быть между `1900` и `32767` включительно;
- месяц должен лежать в диапазоне от `1` до `12`;
- число должно быть допустимым для указанного месяца и года с учетом високосности.

## ДОМАШНЕЕ ЗАДАНИЕ.

Разработать простой арифметический калькулятор, который сможет работать со строками. Например, можно написать:

`(18/6+3)*4`. В результате должен быть выведен результат: `24`.

Калькулятор должен понимать арифметические действия: `+`, `-`, `*`, `/` и разбирать содержание круглых скобок для определения приоритета действий. Эта информация должна быть выведена на обозрение пользователя.

Интерфейс скрипта разработать на своё усмотрение.

## ИТОГИ.

Данный урок представляет собой, по сути, справочник по основным функциям ядра **PHP**, которые предназначены для решения самых распространённых задач: работы со строками, массивами, воспроизведения математических вычислений и обработки дат и времени.

Все изученные в уроке функции можно представить в виде такой таблицы:

| Функция | Описание |
|---|---|
| **Функции работы с массивами** | |
| array array_combine(array keys, array values) | Комбинирование двух массивов |
| array array_cont_values (array input) | Количество вхождений элемента в массиве |
| array array_chunk (array input, int size[, bool keys]) | Разрезание массива на мерные кусочки |
| array array_flip(array array) | Смена местами ключей и значений |
| bool array_key_exists (mixed key, array search) | Проверка существования указанного ключа в массиве. |
| list array_keys(array Arr [,mixed searchVal]) | Получение списка ключей массива |
| array array_merge(array ar1[, array ar2[, array ar3 …]]) | Слияние массивов |
| mixed array_pop(array &Arr) | Удаление последнего элемента массива |
| int array_push (array &Ar, mixed vr1[, mixed vr2, …]) | Добавление элементов в конец массива |
| array array_reverse ( array array [, bool preserve_keys]) | Обращение массива |
| mixed array_search (mixed needle, array stack [,bool strict]) | Поиск в массиве |
| mixed array_shift(array &Arr) | Удаление первого элемента массива |
| void shuffle(array array) | Перемешивание списка |
| array array_slice(array Arr, int offset [, int len]) | Получение части массива |
| number array_sum (array ar) | Сумма всех элементов массива |
| array array_unique(array array) | Получение массива уникальных значений |
| int array_unshift (array &Ar, mixed vr1[, mixed vr2, …]) | Добавление элементов в начало массива |
| list array_values (array input) | Получение списка значений массива |
| bool array_walk(array &arr, callback func[, mixed userdata]) | Применение какой-либо функции к каждому элементу массива |
| bool arsort (array &array [, int sort_flags]) | Сортировка элементов массива в порядке убывания с сохранением связи с ключами |
| bool asort (array &array [, int sort_flags]) | Сортировка элементов массива в порядке возрастания с сохранением связи с ключами |
| array compact ( mixed varname [, mixed ...]) | Изготовление массива из переданных значений |
| int count (mixed var[, int mode]) | Количество элементов в массиве |
| mixed current(array a) | Текущий элемент массива |
| mixed end(array a) | Последний элемент массива |
| void extract(array Arr[, int type] [, string prefix]) | Преобразование массива в набор переменных |
| bool in_array(mixed needle, array stack [,bool strict]) | Проверка существования указанного элемента в массиве. |
| bool krsort (array &array [, int sort_flags]) | Сортировка ключей массива в порядке убывания с сохранением связи с элементами |

| Функция | Описание |
|---|---|
| `bool ksort (array &array [, int sort_flags])` | Сортировка ключей массива в порядке возрастания с сохранением связи с элементами |
| `bool natsort (array &array);` | Натуральная сортировка строковых значений в массиве |
| `mixed next(array a)` | Следующий элемент массива |
| `mixed prev(array a)` | Предыдущий элемент массива |
| `mixed reset(array a)` | Первый элемент массива |
| `bool rsort (array &array [, int sort_flags])` | Сортировка элементов массива в порядке убывания без сохранения связи с ключами |
| `bool sort (array &array [,int sort_flags])` | Сортировка элементов массива в порядке возрастания без сохранения связи с ключами |
| `bool uasort (array array, callback cmp_function)` | Пользовательская сортировка элементов с сохранением связи с ключами |
| `bool uksort (array array, callback cmp_function)` | Пользовательская сортировка ключей с сохранением связи с элементами |
| `bool usort (array array, callback cmp_function)` | Пользовательская сортировка элементов без сохранения связи с ключами |
| **Математические функции** | |
| `mixed abs(mixed number)` | Получение модуля числа |
| `float acos(float arg)` | Арккосинус |
| `float asin(float arg)` | Арксинус |
| `float atan (float arg)` | Арктангенс |
| `string base_convert(string number, int frombase, int to-base)` | Перевод систем исчисления |
| `int bindec(string binary_string)` | Перевод из двоичной системы в десятичную |
| `int ceil(double val)` | Округление в большую сторону |
| `float cos(float arg)` | Косинус |
| `string decbin(int number)` | Перевод из десятичной системы в двоичную |
| `float exp(float arg)` | Экспонента |
| `int floor(double val)` | Округление в меньшую сторону |
| `float log(float arg)` | Натуральный логарифм |
| `mixed max(mixed arg1 [int arg2, ..., int argn])` | Максимальное число |
| `mixed min(mixed arg1 [int arg2, ..., int argn])` | Минимальное число |
| `int mt_getrandmax()` | Получение максимального случайного числа |
| `int mt_rand(int min=0, int max=RAND_MAX)` | Генерация случайного числа |

| Функция | Описание |
|---|---|
| void mt_srand(int seed) | Настройка генератора случайных чисел |
| float pow(float base, float exp) | Возведение в степень |
| double round(double val) | Округление по законам арифметики |
| float sqrt(float arg) | Квадратный корень |
| float sin(float arg) | Синус |
| float tan(float arg) | Тангенс |
| **Функции работы со строками** | |
| string chr(int code) | Символ по известному ASCII-коду |
| string convert_cyr_string(string str, char from, char to) | Конвертация кириллических кодировок |
| int crc32(string str) | Получение контрольной суммы строки |
| array explode (string sep, string string [, int limit ]) | Разбиение строки на части по разделителю |
| string Htmlentities(string str [int quote_style [, string charset]]) | Преобразование спецсимволов в сущности HTML |
| string HtmlSpecialChars(string str) | Преобразование символов разметки HTML в сущности |
| string iconv (string in_chs, string out_chs, string str) | Конвертация любых кодировок |
| string implode(string glue , array pieces) | Склеивание элементов массива в строку |
| string ltrim(string st) | Удаление ведущих пробелов |
| string md5 (string st) | Получение хэш-кода строки |
| int ord (char ch) | ASCII-код символа |
| string nl2br(string string) | Преобразование символа перевода строки в <br> |
| void printf(string format [, mixed args, …]) | Печать форматированного вывода |
| string rtrim(string st) | Удаление концевых пробелов |
| string sprintf(string format [, mixed args, …]) | Форматированный вывод строки |
| int strcmp(string strl, string str2) | Сравнение строк |
| string strip_tags (string str [, string allowable_tags]) | Ликвидация тегов |
| int strlen(string st) | Длина строки |
| int strpos(string where, string what, int fromwhere=0) | Поиск позиции первого вхождения подстроки |
| string str_repeat (string st, string number) | Повтор строки |
| string str_replace(string from, string to, string str [, int count]) | Замена вхождений подстроки |
| int strrpos (string where, char what) | Поиск позиции последнего вхождения подстроки |
| string strtolower (string str) | Перевод символов в нижний регистр |
| string strtoupper (string str) | Перевод символов в верхний регистр |
| string substr(string str, int from [,int length]) | Вывод подстроки |
| string trim(string st) | Удаление пробелов |

| Функция | Описание |
|---|---|
| **Функции для работы с датой и временем** | |
| `int checkdate(int month, int day, int year)` | Проверка корректности даты |
| `string date(string format [,int timestamp])` | Строка форматированного вывода даты и времени |
| `array getdate{int timestamp)` | Массив, содержащщий информацию о дате и времени |
| `int mktime([int h[,int m[,int s[,int mn[,int d[,int y]]]]]])` | Генерация Unix timestamp указанного момента |
| `int time()` | Генерация Unix timestamp текущего момента |