выражение в языке PHP - это вообще любая строка, число, объект, массив, или что-то ещё, что представлено в явном виде или в виде каких-либо вычислений. Например:

'строка с пробелами'

7

2 + 2

5 / 2

код расположен между тегами **<?php** ... **?>**

Для оформления кода PHP также можно использовать **краткую версию тегов: <? и ?>.** Для этого в файле **php.ini** надо изменить строку:

|  |  |
| --- | --- |
| 1 | short\_open\_tag = Off |

на

|  |  |
| --- | --- |
| 1 | short\_open\_tag = On |

### Сокращенная версия тегов php

Если нам надо вывести на веб-страницу одно какое-нибудь значение, то мы можем использовать специальную форму тегов php - **<?= ... ?>** - после знака = ("равно") ставится выводимое выражение. Например:

2 + 2 =  <?= 2+2 ?>

Переменная – это какое-то значение, которому соответствует определённое имя.  
Как в математике, x = 2 – переменная x равна двум. Здесь **x** – имя переменной, **2** – её значение.

В PHP переменные начинаются со знака "**$**".  
Например: **$x**.

Стоит обратить внимание, что имена переменных в PHP являются регистрозависимыми. То есть **$x** и **$X** – две разные переменные.

**Комментарии**

<?php

echo "<p>Привет мир!</p>"; // вывод сообщения

/\*

многострочный комментарий

вывод результата арифметического выражения

echo "2 + 2 = " . (2+2);

\*/

**Правила:**

* Имя переменной должно состоять только из английских слов, и уж тем более только из букв английского алфавита.  
  Никаких **$imyaKota** или тем более **$имяКота**, строго **$catName**
* Имя переменной должно начинаться с маленькой буквы, а следующие слова, содержащиеся в её имени, должны начинаться с большой буквы. Этот стиль называется lowerCamelCase, или "нижнийВерблюжийРегистр"  
  Никаких **$catname** или **$CatName**, только **$catName**.

В сравнении с другими языками программирования, (например, C/C++/Java), в PHP одной и той же переменной можно присваивать значения разных типов. Поэтому в PHP типы следует относить не к переменной, а к значению. И говорить, например, что в переменной **$var** находится значение типа integer. Это называется "**динамическая типизация"** – одна и та же переменная может хранить значения разных типов.

Строки могут быть определены с помощью **двойных или одинарных кавычек**. Но при этом есть разница – в двойных кавычках интерпретатор будет пытаться найти переменные, а внутри одинарных кавычек – нет.

Стоит отметить, что при использовании одинарных кавычек работа со строками будет происходить быстрее, так как не будет происходить поиск (или правильнее сказать [парсинг](https://php.zone/post/793" \t "_blank)) переменных внутри строки.

В PHP есть десять базовых типов данных:

* bool (логический тип)
* int (целые числа)
* float (дробные числа)
* string (строки)
* array (массивы)
* object (объекты)
* callable (функции)
* mixed (любой тип)
* resource (ресурсы)
* null (отсутствие значения)

Из этих типов данных первые четыре являются скалярными: bool, int, float, string.

Шаблоны чисел для других систем:

* шестнадцатеричные : 0[xX][0-9a-fA-F]
* восьмеричные : 0[0-7]
* двоичные : 0b[01]

Константа null не чувствительна к регистру

Поскольку PHP - язык с динамической типизацией, то мы можем присваивать одной и той же переменной значения разных типов:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <?php  $id = 123;  echo "<p>id = $id</p>";  $id = "jhveruuyeru";  echo "<p>id = $id</p>";  ?> |

**Арифметические операции**

* + (операция сложения)

Например, $a + 5

* - (операция вычитания)

Например, $a - 5

* \* (умножение)

Например, $a \* 5

* / (деление)

Например, $a / 5

* % (получение остатка от деления)

Например:

|  |  |
| --- | --- |
| 1  2 | $a = 12;  echo $a % 5; // равно 2 |

* \*\* (возведение в степень)

Например, $a \*\* 2

**var\_dump().** Она позволяет вывести тип и значение чего-либо

<?php

**var\_dump**(5/2);

var\_dump($x == $y); *//проверка на равенство*

var\_dump($x === $y); *//проверка на тождественное равенство*

var\_dump($x != $y); *//проверка на неравенство*

var\_dump($x !== $y); *//проверка на тождественное неравенство*

 Оператор **==** приводит операнды к одному типу и после сравнивает их значения. Так строка '2' была преобразована к числу и значения оказались равными.  
Оператор тождественного равенства **===** не выполняет приведения типов и сравнивает сначала то, что типы значений идентичны, например, целые числа, а затем сравнивает их значения. И если они одинаковы, то только в таком случае возвращает true.

#### **Spaceship**

А теперь рассмотрим оператор сравнения, который появился в PHP7. Это спейсшип (или космический корабль) **<=>**.

Логика у этого оператора следующая:  
$a <=> $b  
Если $a > $b, вернёт 1  
Если $a == $b, вернёт 0  
Если $a < $b, вернёт -1

**Логические операторы**

А теперь поговорим об одном из самых важных типов операторов, а именно – логических. Эти операторы используются для вычисления условий. Результатом такого оператора всегда будет истина или ложь. При этом операндами так же выступают истина или ложь.  
Давайте перечислим их:

* && - логическое И, вернёт true, только если оба операнда истинны (true)
* || - логическое ИЛИ, вернёт true, если хотя бы один из операндов является истинным
* ! – ОТРИЦАНИЕ, возвращает true, если операнд – false, и наоборот. Применяе тся если нужно инвертировать условие, зачастую это очень кстати.
* xor – исключающее ИЛИ, вернёт true, только если один из операндов true, а другой – false

### Поразрядные операции

Поразрядные операции производятся над отдельными разрядными или битами числа.

* & (логическое умножение)

Умножение производится поразрядно, и если у обоих операндов значения разрядов равно 1, то операция возвращает 1, иначе возвращается число 0. Например:

|  |  |
| --- | --- |
| 1  2  3 | $a = 4; //100  $b = 5; //101  echo $a & $b; // равно 4 - 100 |

Здесь число 4 в двоичной системе равно 100, а число 5 равно 101. Поразрядно умножим числа и получим (1\*1, 0\*0, 0 \*1) = 100, то есть число 4 в десятичном формате.

* | (логическое сложение)

Похоже на логическое умножение, операция также производится по двоичным разрядам, но теперь возвращается единица, если хотя бы у одного числа в данном разряде имеется единица. Например:

|  |  |
| --- | --- |
| 1  2  3 | $a = 4; //100  $b = 5; //101  echo $a | $b; // равно 5 - 101 |

Поразрядно сложим числа и получим (1+1, 0+0, 0+1) = 101, то есть число 5 в десятичном формате.

* ^ (операция исключающего ИЛИ)

Возвращает единицу, если у обоих операндов разные значения соответствующих разрядов. Но если у обоих операндов значения соответствующих разрядов совпадают, то возвращается 0. Например:

|  |  |
| --- | --- |
| 1  2 | $a = 5 ^ 4; // 101^100=001  - в десятичой системе 1  $b = 7 ^ 4; // 111^100=011  - в десятичой системе 3 |

Поразрядно произведем операцию 5 ^ 4 (в двоичной системе она аналогична операции 101^100): (1^1, 0^0, 0^1) = 001. В случае с 1^1 значения разрядов совпадают, поэтому возвращается 0. Во втором случае - 0^0 значения также совпадают, поэтому также возвращается 0. В третьем случае - 0^1 значения разные, поэтому возвращается 1. В итоге получится 001 или число 1 в десятичной системе.

* ~ (логическое отрицание)

инвертирует все разряды: если значение разряда равно 1, то оно становится равным нулю, и наоборот.

|  |  |
| --- | --- |
| 1  2  3 | $a = 4;     //00000100  $b = ~$a;   //11111011  -5  echo $b;    // равно -5 |

* <<

x<<y - сдвигает число x влево на y разрядов. Например, 4<<1 сдвигает число 4 (которое в двоичном представлении 100) на один разряд влево, то есть в итоге получается 1000 или число 8 в десятичном представлении

* >>

x>>y - сдвигает число x вправо на y разрядов. Например, 16>>1 сдвигает число 16 (которое в двоичном представлении 10000) на один разряд вправо, то есть в итоге получается 1000 или число 8 в десятичном представлении

\*\*=

Получение результата от возведения в степень

### Приоритет операций

Если одно выражение содержит несколько разных операций, то при выполнении выражения учитывается приоритет операций. Сначала выполняются операции с большим приоритетом и в конце с меньшим приоритетом.

Приоритет операций можно описать следующей таблицей:

|  |
| --- |
| \*\* |
| ++ -- ~ |
| ! |
| \* / % |
| + - . |
| << >> |
| < > <= >= |
| == != === !== << <=< |
| & |
| ^ |
| | |
| && |
| || |
| ? : (тернарный оператор) |
| = += -= \*= \*\*= /= .= %= &= |= ^= <<= >>= (операторы присваивания) |
| and |
| xor |
| or |

## **Конструкция if-elseif-else**

### Комбинированный режим HTML и PHP

Также мы можем написать конструкцию if..else иным образом, переключаясь внутри конструкции на код HTML:

<?php

$a = 5;

?>

<?php if ($a > 0) **{ ?>**

<h2>Переменная a больше нуля</h2>

**<?php }** ?>

## **Приведение к boolean**

**<?php**

$x = (boolean)3;

var\_dump($x);

Результатом будет *true*.

К *false* будут приводиться следующие значения:

* '' (пустая строка)
* 0 (число 0)

## **Оператор switch**

$x = 1;

**switch** ($x) {

**case** 1:

**echo** 'Число равно 1';

**break**;

**case** 2:

**echo** 'Число равно 2';

**break**;

**default**:

**echo** 'Число не равно ни 1, ни 2';

**Функции:**

$string = 'abracadabra';

**echo** str\_replace('a', 'o', $string);

Конструкция switch..case также поддерживает альтернативный синтаксис, при котором вместо открывающей блок фигурной скобки ставится двоеточие, а вместо закрывающей фигурной скобки - ключевое слово **endswitch**

Конструкция **match** также принимает некоторое выражение и сравнивает его с набором значений.

$a = 2;

$operation = match($a)

{

    1 => "сложение",

    2 => "вычитание",

    default => "действие по умолчанию",

};

echo $operation;

Также мы можем переписать предыдущий пример следующим образом:

|  |
| --- |
| $a = 2;  match($a)  {      1 => $operation = "сложение",      2 => $operation = "вычитание",      default => $operation = "действие по умолчанию",  };  echo $operation; |

Стоит отметить важное отличие конструкции **switch** от **match**: **switch** сравнивает только значение, но не учитывает тип выражения. Тогда как **match** также учитывает тип сравниваемого выражения.

**Массивы:**

Есть несколько способов определения массивов. Первый способ представляет использование функции **array()**:

|  |
| --- |
| $numbers = array(); |

Второй способ представляет использование квадратных скобок **[]**:

|  |
| --- |
| $numbers = []; |

При определении массива мы сразу можем передать ему начальные данные. Если применяются квадратные скобки, то элементы массива передаются внутри скобок:

|  |
| --- |
| $numbers = [1, 2, 3, 4]; |

Аналогичное определение массива с помощью функции array():

|  |
| --- |
| $numbers = array(1, 2, 3, 4); |

Для обращения к элементам массива применяются **ключи**. Ключ может представлять число или строку или одновременно и числа, и строки.

Для добавления нового элемента в массив мы можем, как в примере выше, просто установить новый элемент по еще не установленному индексу. Но есть и другой способ:

<?php

$numbers = [1, 4, 9, 16];

$numbers[] = 25;

echo $numbers[4];   // 25

?>

**Чтобы получить полное наглядное представление о том, как в конкретном массиве сопоставляются ключи и значения элементов, можно использовать функцию**print\_r**, в которую в качестве параметра передается массив:**

|  |
| --- |
| <?php  $numbers = [1, 4, 9, 16];  $numbers[] = 25;  print\_r($numbers);// Array ( [0] => 1 [1] => 4 [2] => 9 [3] => 16 [4] => 25 )  ?> |

### Оператор =>

Оператор **=>** позволяет сопоставить ключ с определенным значением.

$numbers = [0=>1, 1=>4, 2=>9, 3=>16];

// $numbers = array(0=>1, 1=>4, 2=>9, 3=>16);

### Перебор массива

Для перебора массива мы можем применять стандартный метод **for**

### Цикл foreach

Тем не менее выше использованный способ перебора не поможет, если индексы определяются вручную и отличаются они от соседних индексов не на единицу, а на произвольную величину. В этом случае мы можем использовать специальный цикл - **foreach**

<?php

$users = [1 => "Tom", 4 => "Sam", 5 => "Bob", 21 => "Alice"];

$num = count($users);

**foreach($users as $element)**

{

    echo "$element<br />";

}

?>

Цикл **foreach** позволяет извлекать не только значения, но и ключи элементов:

<?php

$users = [1 => "Tom", 4 => "Sam", 5 => "Bob", 21 => "Alice"];

$num = count($users);

foreach($users as $key => $value)

{

    echo "$key - $value<br />";}?>

**Ассоциативные массивы** представляют подвид массивов, в которых, в отличие от обычных массивов, в качестве ключа применяются строки.

ассоциативного массива мы явным образом указываем ключ элемента, после которого идет оператор **=>**

$words = array("red" => "красный", "blue" => "синий", "green" => "зеленый");

Для перебора ассоциативного массива применяется цикл **foreach**

### Смешанные массивы

PHP позволяет использовать в одном массиве числовые и строковые индексы:

$data = [1=> "Tom", "id132" => "Sam", 56 => "Bob"];

echo $data[1];  // Tom

echo "<br />";

echo $data["id132"];    // Sam

**массивы** могут также быть **многомерными**, то есть такими, где элемент массива сам является массивом.

$families = array(array("Tom", "Alice"), array("Bob", "Kate"));

$families = [["Tom", "Alice"], ["Bob", "Kate"]];

чтобы обратиться к элементам внутри этих вложенных массивов, нам необходимо использовать второй индекс - $families[0][0]

Перебор многомерного массива:

$families = [["Tom", "Alice"], ["Bob", "Kate"], ["Sam", "Mary"]];

foreach ($families as $family)

{

    echo "<tr>";

    foreach ($family as $user)

    {

        echo "<td>$user</td>";

    }

    echo "</tr>";

}

?>

Также можно определять многомерные ассоциативные массивы

$phones = array(

        "apple"=> array("iPhone 12", "iPhone X", "iPhone 12 Pro") ,

        "samsumg"=>array("Samsung Galaxy S20", "Samsung Galaxy S20 Ultra"),

        "nokia" => array("Nokia 8.3", "Nokia 3.4"));

foreach ($phones as $brand => $items)

{

    echo "<h3>$brand</h3>";

    echo "<ul>";

    foreach ($items as $key => $value)

    {

        echo "<li>$value</li>";

    }

    echo "</ul>";

}

?>

### Именнованные параметры

Начиная с версии 8.0 в PHP была добавлена поддержка **именнованных параметров**. Так, до PHP 8.0 при вызове функции значения можно было передать параметрам только **по позиции**. Именнованные параметры позволяют передавать значения параметрам по имени:

<?php

function displayInfo($name, $age = 18)

{

    echo "<div>Имя: $name <br />Возраст: $age</div><hr>";

}

displayInfo(age: 23, name: "Bob");

displayInfo(name: "Tom", age: 36);

displayInfo(name: "Alice");

?>

Можно сочетать передачу значений параметрам по имени и по позиции. При этом любые именованные необязательные параметры должны располагаться после НЕименованных параметров

### Переменное количество параметров

В PHP функция может принимать переменное количество параметров. Для этого у функции определяется один параметр, перед которым указывается оператор **...** (три точки). Такой параметр рассматривается как массив:

<?php

function sum(...$numbers)

{

    $result = 0;

    foreach($numbers as $number) {

        $result += $number;

    }

    echo "<p>Сумма: $result</p>";

}

sum(1, 2, 3);

sum(2, 3);

sum(4, 5, 8, 10);

?>

Если функция должна принимать и другие параметры, то параметр, который представляет переменное количество значений, указывается в конце после остальных параметров.

function getAverageScore($name, ...$scores)

{

    $result = 0.0;

    foreach($scores as $score) {

        $result += $score;

    }

    $result = $result / count($scores);

    echo "<p>$name: $result</p>";

}

getAverageScore("Tom", 5, 5, 4, 5);

getAverageScore("Bob", 4, 3, 4, 4, 4);

**Анонимные функции** позволяют передавать в качестве параметров функции другие функции или присваивать их переменным.

Анонимная функция определяется как обычная функция за тем исключением, что она не имеет имени.

$hello = function($name)

{

    echo "<h2>Hello $name</h2>";

};

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

function sum($numbers)

{

    $result = 0;

    foreach($numbers as $number){

        $result += $number;

    }

    return $result;

}

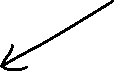
$myNumbers = [-2, -1, 0, 1, 2, 3, 4, 5];

$numbersSum = sum($myNumbers);



echo $numbersSum;           // 12

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



<?php

function sum($numbers, $condition)

{

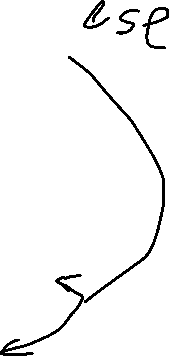
    $result = 0;

    foreach($numbers as $number){

        if($condition($number))

        {

            $result += $number;



        }

    }

    return $result;

}

// для четных чисел

$isEvenNumber = function($n){ return $n % 2 === 0;};

// для положительных чисел

$isPositiveNumber = function($n){ return $n > 0;};

$myNumbers = [-2, -1, 0, 1, 2, 3, 4, 5];

$positiveSum = sum($myNumbers, $isPositiveNumber);

$evenSum = sum($myNumbers, $isEvenNumber);

echo "Сумма положительных чисел: $positiveSum <br /> Сумма четных чисел: $evenSum";

?>

## Замыкания / Closure

Замыкания в PHP представляют анонимную функцию, которая может использовать переменные из своего локального окружения. В отличие от обычных анонимных функций замыкания в PHP применяют выражение **use**.

$a = 8;

$b = 10;

$closure = function($c) **use($a, $b)**

{

    return $a + $b + $c;

};

$result = $closure(22); // 40

echo $result;

**Стрелочные функции** (arrow function) позволяют упростить запись 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**:

$a = 8;

$b = 10;

$closure = fn($c) => $a + $b + $c;

$result = $closure(22); // 40

Фактически будет аналогично:

|  |
| --- |
| $closure = function($c) use($a, $b)  {      return $a + $b + $c;  }; |

**Генератор** предоставляет функцию, которая генерирует набор значений.

Для возвращения значения из функции применяется оператор **yield**. Но в отличие от **return** оператор **yield** сохраняет состояние функции, позволяя ей продолжать работу с того места, когда остановилось ее выполнение

function generateNumbers()

{

    for ($i = 0; $i <= 5; $i++) {

        yield $i;

    }

}

foreach(generateNumbers() as $number)

{

    echo $number; // 012345

}

С помощью оператора **from** можно определять массив - источник данных для генератора:

function generateNumbers()

{

    yield 1;

    yield from [2, 3, 4];

    yield 5;

}

foreach(generateNumbers() as $number)

{

    echo $number; // 12345

}

**Ссылки** в PHP позволяют ссылаться на область памяти, где расположено значение переменной или параметра. Для создания ссылки перед переменной указывается символ амперсанда - **&**.

<?php

$tom = "Tom";

$sam = &$tom;   // передача ссылки

$sam = "Sam";

echo "tom = $tom <br>";   // tom = Sam

echo "sam = $sam";              // sam = Sam

?>

### Возвращение ссылки из функции

Функция также может возвращать ссылку. В этом случае при определении и вызове функции перед ее именем ставится знак амперсанда:

<?php

function &checkName(&$name)

{

    if($name === "admin") $name = "Tom";

    return $name;

}

$userName = "admin";

$checkedName = &checkName($userName);

echo "<br />userName: $userName";

echo "<br />checkedName: $checkedName";

?>

В данном случае функция checkName() получает параметр по ссылке и возвращает ссылку - фактически ссылку, которая передается в функции. Для этого перед определением функции указан символ амперсанда:

function **&**checkName(&$name)

Для имитации работы функция проверяет имя пользователя и изменяет его на некоторое стандартное, если оно равно "admin".

При вызове функции перед ее именем указывается символ амерсанда:

$checkedName = **&**checkName($userName)

После выполнения функции переменная $checkedName фактически будет содержать ссылку на переменную $userName.

### Переменные в блоках цикла 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

### Локальные переменные

Локальные переменные создаются внутри функции. К таким переменным можно обратиться только изнутри данной функции.

### Статические переменные

На локальные переменные похожи статические. Они отличаются тем, что после завершения работы функции их значение сохраняется. При каждом новом вызове функция использует ранее сохраненное значение.

### Глобальные переменные

Глобальные переменные по умолчанию не доступны внутри функции.

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

<?php

$name = "Tom";

function hello()

{

    global $name;

    echo "Hello " . $name;

}

hello();    // Hello Tom

?>

В качестве альтернативы оператору **global** для обащения к глобальным переменным мы можем использовать встроенный массив **$GLOBALS**

<?php

**$name** = "Tom";

function changeName()

{

    $username **= $GLOBALS["name"];**

    echo "Старое имя: $username <br>";

    // изменяем значение переменной $name

**$GLOBALS["name"]** = "Tomas";

}

changeName();

echo "Новое имя: " . $name;

?>

### Оператор const

Для определения константы применяется оператор **const**, при этом в названии константы знак доллара $ (в отличие от переменных) не используется.

<?php

const PI = 3.14;

echo PI;

?>

### Функция define

Также для определения константы может применяться функция **define()**, которая имеет следующую форму:

|  |  |
| --- | --- |
| 1 | define(string $name, string $value) |

Параметр $name передает название константы, а параметр $value - ее значение. Значение константы может представлять тип int, float, string, bool, null или массивы.

### Магические константы

Кроме создаваемых программистом констант в PHP имеется еще несколько так называемых "магических" констант, которые есть в языке по умолчанию:

* **\_\_FILE\_\_**: хранит полный путь и имя текущего файла
* **\_\_LINE\_\_**: хранит текущий номер строки, которую обрабатывает интерпретатор
* **\_\_DIR\_\_**: хранит каталог текущего файла
* **\_\_FUNCTION\_\_**: название обрабатываемой функции
* **\_\_CLASS\_\_**: название текущего класса
* **\_\_TRAIT\_\_**: название текущего трейта
* **\_\_METHOD\_\_**: название обрабатываемого метода
* **\_\_NAMESPACE\_\_**: название текущего пространства имен
* **::class/span>: полное название текущего класса**

### Проверка существования константы

Чтобы проверить, определена ли константы, мы можем использовать функцию **bool defined(string $name)**. Если константа $name определена, то функция будет возвращать значение true

const PI = 3.14;

if (!defined("PI"))

    define("PI", 3.14);

else

    echo "Константа PI уже определена";

### Оператор isset

Функция **isset()** позволяет определить, инициализирована ли переменная или нет. Если переменная определена, то isset() возвращает значение true. Если переменная не определена, то isset() возвращает false. Также если переменная имеет значение null функция isset() также возвращает false.

$message = "Hello PHP";

if(isset($message))

    echo $message;

else

    echo "переменная message не определена";

### empty

Функция empty() проверяет переменную на "пустоту". "Пустая" переменная - это переменная, значение которой равно null, 0, false или пустой строке - в этом случае функция empty() возвращает true:

### unset

С помощью функции **unset()** мы можем уничтожить переменную:

<?php

$a=20;

echo $a; // 20

unset($a);

echo $a; // ошибка, переменная не определена

?>

### Получение типа переменной

Для получения типа переменной применяется функция **gettype()**, которая возвращает название типа переменной, например, integer (целое число), double (число с плавающей точкой), string (строка), boolean (логическое значение), null, array (массив), object (объект) или unknown type.

Также есть ряд специальных функций, которые возвращают true или false в зависимости от того, представляет ли переменная определенный тип:

* **is\_integer($a)**: возвращает значение true, если переменная $a хранит целое число
* **is\_string($a)**: возвращает значение true, если переменная $a хранит строку
* **is\_double($a)**: возвращает значение true, если переменная $a хранит действительное число
* **is\_numeric($a)**: возвращает значение true, если переменная $a представляет целое или действительное число или является строковым представлением числа.
* **is\_bool($a)**: возвращает значение true, если переменная $a хранит значение true или FALSE
* **is\_scalar($a)**: возвращает значение true, если переменная $a представляет один из простых типов: строку, целое число, действительное число, логическое значение.
* **is\_null($a)**: возвращает значение true, если переменная $a хранит значение null
* **is\_array($a)**: возвращает значение true, если переменная $a является массивом
* **is\_object($a)**: возвращает значение true, если переменная $a содержит ссылку на объект

### Установка типа. Функция settype()

С помощью функции **settype()** можно установить для переменной определенный тип. Она принимает два параметра: settype("Переменная", "Тип"). В качестве первого параметра используется переменная, тип которой надо установить, а в качестве второго - строковое описание типа, которое возвращается функцией gettype().

Если удалось установить тип, то функция возвращает true, если нет - то значение false.

Например, установим для переменной целочисленный тип:

|  |
| --- |
| <?php  $a = 10.7;  settype($a, "integer");  echo $a; // 10  ?> |

Поскольку переменная $a представляет действительное число 10.7, то его вполне можно преобразовать в целое число через отсечение дробной части. Поэтому в данном случае функция settype() возвратит true.

**Преобразование типов**

По умолчанию PHP при необходимости автоматически преобразует значение переменной из одного типа в другой. По этой причине явные преобразования в PHP не так часто требуются. Тем не менее мы можем их применять.

Для явного преобразования перед переменной в скобках указывается тип, в который надо выполнить преобразование:

|  |  |
| --- | --- |
| 1  2  3 | $boolVar = false;  $intVar = (int)$boolVar; // 0  echo "boolVar = $boolVar<br>intVar = $intVar"; |

В данном случае значение "false" преобразуется в значение типа int, которое будет храниться в переменной $intVar. А именно значение false преобразуется в число 0. После этого мы сможем использовать данное значение как число.

При использовании выражения echo для вывода на страницу передаваемые значения автоматически преобразуются в строку. И поскольку переменная boolVar равна false, ее значение будет преобазовано в пустую строку. Тогда как значение 0 преобразуется в строку "0".

В PHP могут применяться следующие преобразования:

* **(int), (integer)**: преобразование в int (в целое число)
* **(bool), (boolean)**: преобразование в bool
* **(float), (double), (real)**: преобразование в float
* **(string)**: преобразование в строку
* **(array)**: преобразование в массив
* **(object)**: преобразование в object

## **Операции с массивами**

### Функция is\_array

Функция **is\_array()** проверяет, является ли переменная массивом, и если является, то возвращает true, иначе возвращает false. Например:

|  |  |
| --- | --- |
| 1  2  3 | $users = ["Tom", "Bob", "Sam"];  $isArray = is\_array($users);  echo ($isArray==true)?"это массив":"это не массив"; |

### Функции count/sizeof

Функция **count()** и **sizeof()** получают количество элементов массива:

|  |  |
| --- | --- |
| 1  2  3  4  5 | $users = ["Tom", "Bob", "Sam"];  $number = count($users);  // то же самое, что  // $number = sizeof($users);  echo "В массиве users $number элемента/ов"; |

### Функции shuffle

Функция **shuffle** перемешивает элементы массивы случайным образом:

|  |  |
| --- | --- |
| 1  2  3  4  5 | $users = ["Tom", "Bob", "Sam", "Alice"];  shuffle($users);  print\_r($users);  // один из возможных вариантов  //Array ( [0] => Bob [1] => Tom [2] => Alice [3] => Sam ) |

### Функции compact

Функция **compact** позволяет создать из набора переменных ассоциативный массив, где ключами будут имена переменных:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?php    $model = "Apple II";  $producer = "Apple";  $year = 1978;    $data = compact("model", "producer", "year");  print\_r($data);  // получится следующий вывод  // Array ( [model] => Apple II [producer] => Apple [year] => 1978 )  ?> |

Функция compact получает в скобках набор переменных. Каждая переменная указывается в кавычка без знака $. Результатом функции является новый массив.

### Сортировка массивов

В PHP имеются два типа сортировки: сортировка строк по алфавиту и сортировка чисел по возрастанию/убыванию. Если сортируемые значения представляют строки, то они сортируются по алфавиту, если числа - то они сортируются в порядке возрастания чисел. PHP по умолчанию самостоятельно выбирает тип сортировки.

Для сортировки по возрастанию используется функция **asort**:

|  |  |
| --- | --- |
| 1  2  3  4  5 | $users = ["Tom", "Bob", "Sam", "Alice"];  asort($users);  print\_r($users);  // вывод отсортированного массива  // Array ( [3] => Alice [1] => Bob [2] => Sam [0] => Tom ) |

В данном случае значения массива представляют строки, поэтому PHP выберет сортировку по алфавиту. Однако с помощью дополнительного параметра мы можем явно указать интерпретатору PHP тип сортировки. Данный параметр может принимать три значения:

* SORT\_REGULAR: автоматический выбор сортировки
* SORT\_NUMERIC: числовая сортировка
* SORT\_STRING: сортировка по алфавиту

Укажем явно тип сортировки:

|  |  |
| --- | --- |
| 1 | asort($users, SORT\_STRING); |

Чтобы отсортировать массив в обратном порядке, применяется функция **arsort**:

|  |  |
| --- | --- |
| 1 | arsort($users); |

#### Сортировка по ключам

Функция asort производит сортировку по значениям элементов, но также существует и еще и сортировка по ключам. Она представлена функцией **ksort**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | $states = ["Spain" => "Madrid", "France" => "Paris", "Germany" => "Berlin", ];  asort($states);  print\_r($states);  // массив после asort   - сортировка по значениям элементов  // Array ( [Germany] => Berlin [Spain] => Madrid [France] => Paris )    ksort($states);  print\_r($states);  // массив после ksort - сортировка по ключам элементов  //  Array ( [France] => Paris [Germany] => Berlin [Spain] => Madrid ) |

Сортировка по ключам в обратном порядке выполняется функцией **krsort()**:

|  |  |
| --- | --- |
| 1 | krsort($states); |

#### Естественная сортировка

Хотя выше описанные функции сортировки прекрасно выполняют свою работу, но их возможностей все-таки недостаточно. Например, отсортируем по возрастанию следующий массив:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php  $os = array("Windows 7", "Windows 8", "Windows 10");  asort($os);  print\_r($os);  // результат  // Array ( [2] => Windows 10 [0] => Windows 7 [1] => Windows 8 )  ?> |

Так как значения представляют строки, то PHP сортирует по алфавиту. Однако подобная сортировка не учитывает числа и регистр. Поэтому значение "Windows 10" будет идти в самом начале, а не в конце, как должно было быть. И для решения этой проблемы в PHP есть функция **natsort()**, которая выполняет естественную сортировку:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <?php  $os = array("Windows 7", "Windows 8", "Windows 10");  natsort($os);  print\_r($os);  // результат  // Array ( [0] => Windows 7 [1] => Windows 8 [2] => Windows 10)  ?> |

Если нам надо еще при этом, чтобы сортировка не учитывала регистр, то мы можем применить функцию **natcasesort()**:

|  |  |
| --- | --- |
| 1 | natcasesort($os); |

## Отправка форм

Создание форм состоит из следующих аспектов:

* Создание элемента <form><form> в разметке HTML
* Добавление в этот элемент одно или несколько поле ввода
* Установка метода передачи данных. Чаще всего используются методы GET или POST
* Установка адреса, на который будут отправляться введенные данные

## Безопасность данных

Например, введем в поле для имени "<script>alert(“hi”);</script>":

После отправки данных в html разметку будет внедрен код javascript, который выводит окно с сообщением.

Рекомендуется применять функцию **htmlentities()**. В качестве параметра она принимает значение, которое надо экранировать

Еще одна специальная функция - **htmlspecialchars()** похожа по действию на **htmlentities**

Еще одна функция - функция **strip\_tags()** позволяет полностью исключить теги html

## Отправка массивов

первое поле добавляет в массив элемент с ключом "first"

|  |  |
| --- | --- |
| 1 | <input type="text" name="users[first]" /> |

Поэтому на сервере мы можем с помощью данного ключа получить соответствующий элемент:

|  |  |
| --- | --- |
| 1 | $firstUser = $\_POST["users"]["first"]; |

## Работа с полями ввода форм

Флажки или чекбоксы (html-элемент <input type="checkbox"/>) могут находиться в двух состояниях: отмеченном (checked) и неотмеченном. Например:

|  |  |
| --- | --- |
| 1 | Запомнить: <input type="checkbox" name="remember" checked="checked" /> |

### Переключатели

Переключатели или радиокнопки позволяют сделать выбор между несколькими взаимоисключающими вариантами

<?php

if(isset($\_POST["course"]))

{

    $course = $\_POST["course"];

    echo $course;

}

?>

<h3>Форма ввода данных</h3>

<form method="POST">

    <**input type="radio**" name="course" value="ASP.NET" />ASP.NET <br>

    <input type="radio" name="course" value="PHP" />PHP <br>

    <input type="radio" name="course" value="Node.js" />Node.js <br>

    <input type="submit" value="Отправить">

</form>

### Список

Список представляет элемент select, который предоставляет выбор одного или нескольких элементов:

<?php

if(isset($\_POST["course"]))

{

    $course = $\_POST["course"];

    echo $course;

}

?>

<h3>Форма ввода данных</h3>

<form method="POST">

    <select name="course" size="1">

**<option value**="ASP.NET">ASP.NET</option>

        <option value="PHP">PHP</option>

        <option value="Ruby">RUBY</option>

        <option value="Python">Python</option>

    </select>

    <input type="submit" value="Отправить">

</form>

## Отправка файлов на сервер

Чтобы загрузить файл на сервер, нам надо использовать форму с параметром enctype="multipart/form-data" и массив $\_FILES.

<?php

if ($\_FILES && $\_FILES["filename"]["error"]== UPLOAD\_ERR\_OK)

{

    $name = $\_FILES["filename"]["name"];

    move\_uploaded\_file($\_FILES["filename"]["tmp\_name"], $name);

    echo "Файл загружен";

}

?>

<h2>Загрузка файла</h2>

<form method="post" enctype="multipart/form-data">

Выберите файл: <input type="file" name="filename" size="10" /><br /><br />

<input type="submit" value="Загрузить" />

</form>

### Мультизагрузка

Изменим скрипт **upload.php** так, чтобы он поддерживал множественную загрузку:

if($\_FILES)

{

    foreach ($\_FILES["uploads"]["error"] as $key => $error) {

        if ($error == UPLOAD\_ERR\_OK) {

            $tmp\_name = $\_FILES["uploads"]["tmp\_name"][$key];

            $name = $\_FILES["uploads"]["name"][$key];

            move\_uploaded\_file($tmp\_name, "$name");

        }   }

    echo "Файлы загружены";

}

?>

<h2>Загрузка файла</h2>

<form method="post" enctype="multipart/form-data">

    <input type="file" name="uploads[]" /><br />

    <input type="file" name="uploads[]" /><br />

    <input type="file" name="uploads[]" /><br />

    <input type="submit" value="Загрузить" />

</form>

## Объекты и классы

Для создания класса в PHP используется ключевое слово **class**, после которого идет название класса и фигурные скобки {} - блок кода класса. Например, новый класс, представляющий пользователя:

|  |  |
| --- | --- |
| 1  2 | class Person  {} |

Чтобы создать объект класса Person, применяется ключевое слово **new**:

|  |
| --- |
| <?php  class Person  { }   $person = new Person();  print\_r($person);  ?> |

В данном случае переменная $person является объектом класса Person. С помощью функции print\_r() можно вывести содержимое объекта, как и в случае с массивами.

### Ключевое слово this

Для обращения к свойствам и методам объекта внутри его класса применяется ключевое слово **this**.

Для обращения к полям и методам внутри класса также применяется оператор доступа **->**, перед которым идет **$this**. Причем это **$this** указывает именно на текущий объект.

class Person

{

    public $name = "Undefined", $age = 18;

    function displayInfo()

    {

        echo "Name: $this->name; Age: $this->age<br>";

    }

}

### Сравнение объектов

При сравнении объектов классов следует принимать во внимание ряд особенностей. В частности, при использовании оператора равенства **==** два объекта считаются равными, если они представляют один и тот же класс и их свойства имеют одинаковые значения.

А при использовании оператора эквивалентности **===** оба объекта считаются равными, если обе переменных классах указывают на один и тот же экземпляр класса



|  |
| --- |
| $person = new Person();    $tom = $person;  $tom -> name = "Tom";  $tom -> age = 36;    $tomas = $person;    if($tom == $tomas) echo "переменные tom и tomas равны<br>";  else echo "переменные tom и tomas НЕ равны<br>";    if($tom === $tomas) echo "переменные tom и tomas эквивалентны";  else echo "переменные tom и tomas НЕ эквивалентны"; |

Здесь объект класса Person создается только один раз: $person = new Person();. И затем обе переменных $tom и $tomas будут указывать на этот объект. При этом не имеет значения, для какой именно переменной мы устанавливаем свойства. Так как в реальности это будет один и тот же объект. В итоге и оператор ==, и оператор === при сравнении возвратят true

## Конструкторы и деструкторы

Конструкторы представляют специальные методы, которые выполняются при создании объекта и служат для начальной инициализации его свойств. Для создания конструктора надо объявить функцию с именем **\_\_construct** (с двумя подчеркиваниями впереди):

<?php

class Person

{

    public $name, $age;

    function \_\_construct($name, $age)

    {

        $this->name = $name;

        $this->age = $age;

    }

    function displayInfo()

    {

        echo "Name: $this->name; Age: $this->age<br>";

    }

}

$tom = new Person("Tom", 36);

$tom -> displayInfo();

### Объявление свойств через конструктор

Начиная с версии PHP 8 в языке появилась возможность определить свойства через список параметров конструктора. Любой параметр конструктора, который имеет модификатор доступа, например, **public**, будет автоматически представлять новое свойство.

<?php

class Person

{

    function \_\_construct(public $name, public $age)

    {

        $this->name = $name;

        $this->age = $age;

    }

    function displayInfo()

    {

        echo "Name: $this->name; Age: $this->age<br>";

    }

}

$tom = new Person("Tom", 36);

$tom -> displayInfo();

?>

### Деструкторы

Деструкторы служат для освобождения ресурсов, используемых программой - для освобождения открытых файлов, открытых подключений к базам данных и т.д. Деструктор объекта вызывается самим интерпретатором PHP после потери последней ссылки на данный объект в программе.

Деструктор определяется с помощью функции **\_\_destruct** (два подчеркивания впереди)

class Person

{

    public $name, $age;

    function \_\_construct($name, $age)

    {

        $this->name = $name;

        $this->age = $age;

    }

    function getInfo()

    {

        echo "Имя: $this->name ; Возраст: $this->age <br>";

    }

    function \_\_destruct()

    {

        echo "Вызов деструктора";

    }

}

**Анонимные классы** - это классы, которые не имеют имени. Обычно такие классы полезны, если нам необходимо один раз создать объект подобного класса. И больше этот класс не будет использоваться. Например:

|  |  |
| --- | --- |
| 1 | $person = new class {}; |

Для определения переменной аонимного класса, как в случае с обычным классом применяется оператор **new**, после которого идет ключевое слово class и затем - тело класса в фигурных скобках.

Анонимные классы, как и обычные классы могут определять свойства и методы.

$person = new class {

    public $name;

    function sayHello(){

        echo "Hello!<br>";

    }

};

$person->sayHello();

$person -> name = "Sam";

echo "Name: " . $person -> name . "<br>";

Также анонимные классы могут определять конструкторы:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | $person = new class("Bob") {        public $name;      function \_\_construct($name)      {          $this->name = $name;      }      function sayHello(){          echo "Hello!<br>";      }  };  echo "Hello, " . $person -> name . "<br>"; // Hello, Bob |