

```
$my_variable; $My_variable; $myvariable;
```

integer (целое число)

```
echo PHP_INT_MAX; // 9223372036854775807
```

double (вещественное число)

```
$x = 0.00012;
```

string (строка текста)

строка в апострофах (\' трактуется PHP как апостроф ;
\\ один обратный слеш ;
символ \$ не имеет специального значения)

строка в кавычках (\n символ новой строки;
\r символ возврата каретки;
\t символ табуляции;
\\$ предваряет любую переменную ; .
\" обозначает кавычку;
\\ обозначает обратный слеш)

boolean (логический тип)

```
echo false; // выводит пустую строку, т. е. ничего не выводит  
echo true; // выводит 1
```

array (ассоциативный массив)

```
$a = array(  
    0=> "Нулевой элемент",  
    "surname" => "Иванов",  
    "name" => "Иван");
```

```
echo $a["surname"]; // выведет Иван  
$a["1"] = "Первый элемент"; // создаст элемент и присвоит ему значение  
$a[" surname "] = "Петров"; // присвоит существующему элементу новое  
значение  
$a = array('1' => 'Элемент массива') + $a;
```

```
foreach ($a as &$value) {  
    echo $value ;  
}
```

Присвоение значения

```
$имя_переменной = значение;  
$a = 0; $b = 1;  
if($a = $b) echo "a и b одинаковы";  
else echo "a и b различны";  
$a = ($b = 4) + 5;
```

Проверка существования

```
if (isset($my_variable))
```

Передача множества параметров

```
if (isset($_POST['name'], $_POST['pass'])) { }
```

Уничтожение

```
// Переменная $a не существует  
$a = "Hello world!"; //переменная $a инициализирована  
echo $a;  
// удаление переменной $a  
unset($a); // переменная $a не существует  
echo $a; // предупреждение: нет такой переменной $a  
unset($a["1"]);
```

Определение типа переменной

```
is_double($a)           $b = 3.1;
is_string($a)           $c = true;
is_bool($a)             var_dump($b, $c);
is_null($a)             //float(3.1) bool(true)
is_array($a)
```

```
$a = false;
if (is_bool($a) === true) {
    echo "Это булева переменная";
}
echo gettype($a) // boolean
```

Установка типа переменной

```
settype($var, $type)

settype($a, "integer");
settype($b, "string");
echo $a; // 5 (целое)
echo $b; // "1" (строка)
```

floatval (\$var) — возвращает значение заданной переменной в виде числа с плавающей точкой.

var — значение для конвертации

```
$a='12.3HelloWorld';  
$float_value_of_var = floatval($a);  
echo $float_value_of_var; // 12.3
```

intval(\$var[, \$base])

base — основание системы исчисления для преобразования

```
echo intval('42,6'); // 42  
echo intval('42', 8); // 34
```

```
$value = 3.14;  
echo (int)$value ." (" . gettype((int)$value) . ")"; //3 (integer)  
echo (string)$value ." (" . gettype((string)$value) . ")"; //3.14 (string)  
echo (boolean)$value ." (" . gettype((boolean)$value) . ")"; //1 (boolean)
```

Операция преобразования

6

(int) (integer)	Приведение к целому типу
(bool) (boolean)	Приведение к логическому типу <code>\$a = 10; // \$a – это целое число</code> <code>\$b = (boolean) \$a; // \$b – это булев тип</code> <code>echo \$b;</code>
(float) (double) (real)	Приведение к вещественному типу
(string)	Приведение к строке <code>\$a = 10; // \$a – это целое число</code> <code>\$b = "\$a"; // \$b – это строка</code> <code>\$c = (string) \$a; // \$c – это строка</code> <code>if (\$c === \$b) {</code> <code> echo "они одинаковы"; }</code>
(array)	Приведение к массиву
(object)	Приведение к объекту

Жесткая (&) ссылка — переменная, которая является синонимом другой переменной (элемент массива).

```
$c = 10;  
$d = &$c; // теперь $d - то же самое, что и $c  
$d = 0; //на самом деле $c = 0  
echo "d = $d, c = $c"; // выводит "d = 0, c = 0"
```

Ссылаться можно на элемент массива:

```
$Spec = array('специальность' => 'ПИЭ', 'группа' => '310');  
$spec = &$Spec['группа'];  
echo $Spec['группа'];  
echo $spec;  
unset ($spec);
```

Жесткая ссылка не может ссылаться на несуществующий объект:

```
$A=array('a' => 'aaa', 'b' => 'bbb');  
$b=&$A['c']; // создается переменная $b  
echo "Элемент с индексом 'c': (".$A['c'].")";  
$A['c']="ccc";  
echo "Элемент с индексом 'c': (".$A['c'].")";
```

Символическая ссылка — строковая переменная, хранящая *имя* другой переменной. Чтобы получить значение переменной, на которую указывает символическая ссылка, необходимо применить оператор разыменовывания — дополнительный знак \$ перед именем ссылки.

```
$red= "красный";  
$blue= "синий";  
$color= "red";  
echo $$color;
```

/*Обычная переменная используется в качестве ссылки. Интерпретатор получает не значение самой переменной, а значение переменной, имя которой хранится в переменной color.

```
*/
```


Собственные, новые константы — величины, значения которых не изменяются в течение работы программы (математические константы, пути к файлам, разнообразные пароли).

```
define ($name, $value, $case_sen);
```

\$name - имя константы;

\$value - значение константы;

\$case_sen - необязательный параметр логического типа, указывающий, следует ли учитывать регистр букв (true) или нет (false).

```
define("STUDENTS", 30);
```

```
$group = STUDENTS/2;
```

```
echo $group; //15
```

```
echo STUDENTS; //30
```

```
echo students;
```

Предопределенные константы:

_ FILE _ — возвращает имя файла, включая полный путь: ;

```
echo "Имя файла: " . _ FILE _; // Имя файла: C:\data\index.php
```

_ NAMESPACE _ — имя текущего пространства имен.

```
namespace MySampleNS;
```

```
echo "Пр. имен: " . _ NAMESPACE _; // Пр. имен: MySampleNS
```

_ LINE _ — возвращает текущий номер строки, которую обрабатывает в текущий момент интерпретатор;

_ FUNCTION _ — имя текущей функции;

_ CLASS _ — имя текущего класса;

_ PHP_VERSION _ — версия интерпретатора PHP;

Проверка существования константы

```
if (defined('STUDENTS')) {  
    echo STUDENTS;  
}
```

- значение не изменяется в процессе выполнения скрипта;
- имя константы должно соответствовать правилам имен в PHP (начинается с буквы или символа подчеркивания и состоит из букв, цифр и подчеркиваний);
- константы доступны из любой области видимости;
- нельзя использовать имя константы непосредственно в текстовой строке;
- имя константы не должно начинаться со знака \$;
- имя константы зависит от регистра;

Арифметические операции

$a + b$ — сложение; $a - b$ — вычитание;

$a * b$ — умножение; a / b — деление;

$a \% b$ — остаток от деления a на b ; $a ** b$ — возведение a в степень b

Строковые операции

$a.b$ — слияние строк a и b ;

$a[n]$ — символ строки в позиции n .

```
 $\${'a'.'b'} = 'hello world'; echo $ab; // hello world$ 
```

Операции инкремента и декремента

$\$a++$ — увеличение переменной $\$a$ на 1;

$\$a--$ — уменьшение переменной $\$a$ на 1.

```
 $\$a = 10;$ 
```

```
 $\$b = \$a++;$ 
```

```
echo "a = $a, b = $b";
```

```
 $\$a = 10;$ 
```

```
 $\$b = --\$a;$ 
```

```
echo "a = $a, b = $b";
```

Операции сравнения

>, < и == || (логическое или), &&(логическое и), !(логическое не)

$a == b$ — истина, если a равно b ;

$a != b$ — истина, если a не равно b ;

$a === b$ — истина, если a эквивалентно b ;

$a !== b$ — истина, если a не эквивалентно b ;

$a < b$ — истина, если a меньше b^* ;

$a > b$ — истина, если a больше b ;

$a <= b$ — истина, если a меньше либо равно b ;

$a >= b$ — истина, если a больше либо равно b ;

$a <=> b$ — возвращает -1, если a меньше b , 0, если a равно b и 1, если a больше b .

Операции эквивалентности

```
$first = 0; // ноль
```

```
$second=""; // пустая строка
```

```
if ($first == $second) echo "переменные равны";
```

```
$first = 0; // ноль
```

```
$second=""; // пустая строка
```

```
if ($first === $second) echo "переменные эквиваленты ";
```

Условные операции — возвращает *y*, в случае если *x* принимает значение `true`, и *z* в случае, если *x* принимает значение `false`

`x ? y : z`

```
$x = -17;
```

```
$x = $x < 0 ? -$x : $x;
```

```
echo $x; // 17
```

Допускается не указывать средний параметр условного оператора:

```
$x = $x ? 1;
```

```
echo $x; // 1
```

Проверка установлено ли значение массива:

```
if ($_POST["reg"] ? true : false) echo "Кнопка нажата!";
```

```
if (isset($_POST["reg"]) ? true : false) echo "Кнопка нажата!";
```

```
$val= (isset($_POST["reg"]) ? $_POST["reg"] : false);
```

require имя_файла;

include имя_файла;

Инструкции однократного включения

include_once(\$a);

require_once('header.php');

Другие инструкции

function — объявление функции;

return — возврат из функции;

class — объявление класса;

new — создание объекта;

var, private, static, public— определение свойства класса;

yield— передача управления из генератора;

throw — генерация исключения;

try-catch— перехват исключения

```
function имяФункции {аргумент1 [=значение1] ,..., аргN[=знN]} {  
операторы_тела_функции;  
return "значение, возвращаемое функцией";  
}
```

Инструкция return

```
function_mySqr($n) {  
return $n * $n;  
}  
echo mySqr(10); // выводит 100
```

Параметры по умолчанию (задаются совершенно единообразно или опускаются):

```
function selectItems($items, $selected =0) { ... }  
echo selectItems($names, "Значение"); // выбран элемент  
echo selectItems ($names); // ни один элемент не выбран по умолчанию
```

// Ошибка! Опускать параметры можно только справа налево!

```
function selectItems($selected= 0, $items) {...}  
echo selectItems($names);
```



```
function myecho() {  
    foreach (func_get_args() as $v) {  
        echo "$v<br/>"; // выводим элемент }  
    }  
    // Отображаем строки одну под другой  
    myecho("Иванов", "Петров", "Сидоров");
```

func_get_args() — возвращает список всех аргументов, указанных при вызове функции.

Оператор ... (рассматривается как массив) можно использовать перед аргументами. Переменный аргумент ... \$surname будет заполняться в зависимости от количества переданных аргументов

```
function myecho(...$surname){  
    foreach ($surname as $v) {  
        echo "$v<br />"; // выводим элемент }  
    }  
    // Отображаем строки одну под другой  
    myecho("Иванов", "Петров", "Сидоров");
```

Оператор ... можно использовать при вызове из массива:

```
function myecho($fst, $snd, $thd, $fth){  
echo "Первый: $fst<br/>";  
echo "Второй: $snd<br/>";  
echo "Третий: $thd<br/>";  
echo "Четвертый: $fth<br/>";  
}
```

```
// Отображаем строки одну под другой  
$surname= ["Иванов", "Петров", "Сидоров", "Иванов"];  
myecho(... $surname); ?>
```

```
function myecho() {  
for ($i = 0; $i < func_num_args(); $i++) {  
echo func_get_arg($i)."<br />\n"; // выводим элемент  
} }  
// Отображаем строки одну под другой  
myecho("Иванов", "Петров", "Сидоров");
```

func_num_args() — возвращает общее число аргументов, переданных функции при вызове.

func_get_arg (int \$num) — возвращает значение аргумента с номером **\$num**, заданным при вызове функции. Нумерация отсчитывается с нуля.

func_get_args() — возвращает список всех аргументов, указанных при вызове функции. Чаще всего применение этой функции оказывается практически удобнее, чем первых двух.

```
function increment($a) {  
  
    echo "Текущее значение: $a<br/>";  
    $a++;  
    echo "Послеувеличения: $a<br/>";  
}  
  
$num= 10;  
  
echo "Начальное значение: $num<br/> ";  
  
increment($num);  
  
echo "После вызова функции: $num<br/>";
```

```
function increment(&$a) {  
//функция increment(&$a) получила доступ к переменной $num
```

```
echo "Текущее значение: $a<br/>";  
$a++;  
echo "Послеувеличения: $a<br/>";  
}
```

```
$num= 10;
```

```
echo "Начальное значение: $num<br/> ";
```

```
increment($num);
```

```
echo "После вызова функции: $num<br/>";
```

Допускается указывать типы аргументов и тип возвращаемого функцией значения:

```
function sum(int $fst, int $snd) : int {  
    return $fst + $snd;  
}  
echo sum(2, 2); // 4  
echo sum(2.5, 2.5); //4
```

Режим жесткой типизации аргументов функции (требование от аргументов функции указанных при объявлении типов):

```
declare (strict_types = 1);  
function sum(int $fst, int $snd) : int {  
    return $fst + $snd;  
}  
echo sum(2, 2); // 4  
echo sum(2.5, 2.5); // Fatal Error в PHP < 7,  
// ExceptionTypeError в PHP >=7
```

Глобальные переменные — это переменные, которые доступны всей программе, включая подпрограммы (пользовательские функции).

Локальные переменные — переменные, определенные внутри пользовательской функции. Доступны только внутри функции, в которой они определены.

```
$a = 100;  
function test($a) {  
  echo $a;  
  $a++;  
}  
test(1);  
echo $a;
```

```
$globalName = "Глобальная переменная";
```

```
function test() {  
    $localName = "Локальная переменная";  
    echo $localName; }  
  
test();  
echo $globalName;  
echo $localName
```

```
$globalName = "Глобальная переменная";
```

```
function test() {  
    $localName = "Локальная переменная";  
    echo $localName;  
    global $globalName;  
    echo $globalName;}  
  
test();
```


`$GLOBALS ['var']` — ассоциативный массив, содержащий ссылки на все переменные, объявленные в программе.

```
$globalName = "Глобальная переменная";
```

```
function test() {  
    echo $GLOBALS['globalName'] . "<br>";  
}  
test();
```

```
function GlobalVar() {  
    $GLOBALS['b'] = $GLOBALS['a'] * $GLOBALS['b'];  
}  
GlobalVar();  
echo($b);
```

- Статическая переменная** — переменная внутри функции, которая сохраняет свое значение после выхода из функции;
- существует только локальной области видимости функции;
 - сохраняет значение после завершения работы функции (когда выполнение программы выходит из области видимости);
 - при последующих вызовах функции возвращается значение, которое было сформировано при последнем вызове функции;
 - объявление производится с помощью ключевого слова `static`: `$var1`;

```
function Static_a(){
static $a = 1;
$a = $a*2;
echo $a;
}
Static_a(); // $a = 2
echo $a; // $a доступна только внутри функции
Static_a(); // $a =4
```

```
function count() {  
  static $count;  
  $count++;  
  echo $count;  
}  
for ($i = 0; $i < 5; $i++) count();
```

```
function funct(){  
  static $int = 0;      // верно  
  static $int = 1+2;    // неверно (поскольку это выражение)  
  static $int = sqrt(121); // неверно (поскольку это выражение)  
  
  $int++;  
  echo $int;  
}
```

- функции позволяют избежать повторения программного кода;
- одну и ту же функцию можно использовать в разных программах;
- функции повышают уровень модульности программы и облегчают ее проектирование;
- каждая функция не должна занимать больше 20-30 строк;
- небольшие, по возможности независимые, части, повысят читабельность, устойчивость и переносимость программ;
- желательно применять стандартные, встроенные (реализованные в ядре РНР) функции.

Сессии — механизм, позволяющий хранить индивидуальные для каждого пользователя данные (имя клиента, номер счета, содержимое "виртуальной корзины") между запусками сценария.

Сессия — это некоторое место долговременной памяти (обычно часть на жестком диске и часть — в cookies браузера), которое сохраняет свое состояние между вызовами сценариев одним и тем же пользователем.

Идентификатор сессии (Session ID) — уникальное имя для однозначного определения временного хранилища, которое будет использовано для запоминания данных сессии между запусками сценария.

`$_SESSION[]` — суперглобальный массив, который позволяет сохранять данные между запросами.

`session_id ()` — функция возвращает текущий идентификатор сессии SID;

`session_save_path` — функция определения директории для хранения.

`session_start ()` — функция инициализирует механизм сессий для текущего клиента, запустившего сценарий.

Действия:

- если клиент запускает программу впервые, устанавливается cookies с уникальным идентификатором и создается временное хранилище, ассоциированное с этим идентификатором;
- определяется, какое хранилище связано с текущим идентификатором пользователя;
- если в хранилище имеются какие-то данные, они помещаются в массив `$_SESSION`.

```
<?php
session_start();
if (!isset($_SESSION['counter'])) $_SESSION['counter']=0;
echo "Вы обновили эту страницу " .$_SESSION['counter']++." раз. ";
echo "<br><a href='".$_SERVER['PHP_SELF']."'>обновить</a>";
?>
```

session_destroy() — функция уничтожает хранилище сессии. При этом массив `$_SESSION[]` не очищается.

```
session_start();  
// очистка всех данных сессии  
unset($_SESSION['name']);  
session_unset();  
$_SESSION = array();  
  
// Уничтожить хранилище сессии  
session_destroy();  
header('Location: index.php');
```