Комментарии

PHP поддерживает комментарии в стиле 'C', 'C++' и оболочки Unix (стиль Perl). Например:

<?php

    echo "Это тест"; // Это однострочный комментарий в стиле C++

    /\* Это многострочный комментарий

       ещё одна строка комментария \*/

    echo "Это ещё один тест";

    echo "Последний тест"; # Это комментарий в стиле оболочки Unix

?>

Переменные

Переменные хранят отдельные значения, которые можно использовать в выражениях PHP. Для определения переменных применяется знак доллара $. Например:

$num;

Здесь определена переменная $num. Поскольку определение переменной - это отдельная инструкция, она завершается точкой с запятой.

Как правило, названия переменный начинаются с маленькой буквы или символа подчеркивания. Стоит учитывать, что PHP является регистрозависимым языком, а значит, переменные $num и $Num будут представлять две разные переменные.

Также при наименовании переменных нам надо учитывать следующие правила:

Имена переменных должны начинаться с алфавитного символа или с подчеркивания

Имена переменных могут содержать только символы: a–z, A–Z, 0–9, и знак подчеркивания

Имена переменных не должны включать в себя пробелы

С помощью операции присвоения (=) переменной присваивается определенное значение:

Отличие кавычек двойные и одинарные:

<?php

$name = 'Vasya';

// выведет "My name is $name"

echo 'My name is $name';

// выведет "My name is Vasya"

echo "My name is $name";

?>

Результат:



$num = 10;

Здесь определена переменная $num, которая хранит число 10.

После определения переменной и присвоения ей значения мы можем использовать ее в выражениях PHP. Например, вывести ее значение на веб-страницу:

<!DOCTYPE html>

<html>

<head>

<title>Test VARS</title>

<meta charset="utf-8" />

</head>

<body>

<?php

// определение переменной $num

$num = 10;

// вывод значения переменной $num на веб-страницу

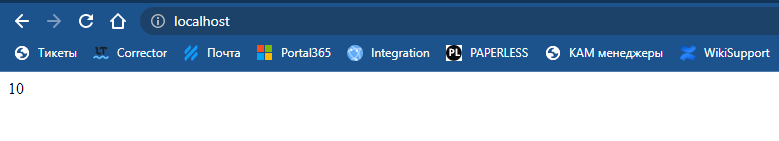
echo $num;

?>

</body>

</html>

С помощью функции echo значение переменной $num выводится на веб-страницу. И при обращении к скрипту мы увидим значение переменной $num:



Отличительной особенностью переменных является то, что мы можем изменять их значение:

<!DOCTYPE html>

<html>

<head>

<title>Test vars</title>

<meta charset="utf-8" />

</head>

<body>

<?php

// определение переменной $num

$num = 10;

// вывод значения переменной $num на веб-страницу

echo $num;

// меняем значение переменной

$num = 22;

echo $num;

?>

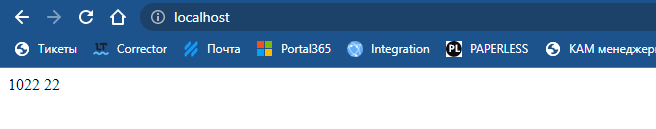
<?php

    echo $num; // Мы находимся в одном скриптовом файле, а это значит, что $num доступен и здесь

?>

</body>

</html>



Вывод информации в теге h1:

<!DOCTYPE html>

<html>

<head>

<title>test vars</title>

<meta charset="utf-8" />

</head>

<body>

<?php

// определение переменной $num

$num = 10;

// вывод значения переменной $num на веб-страницу

echo $num;

// меняем значение переменной

$num = 22;

echo $num;

?>

<?php

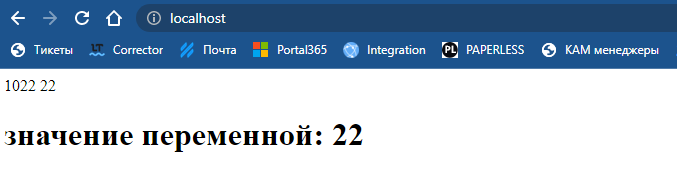
    echo $num; // Мы находимся в одном скриптовом файле, а это значит, что $num доступен и здесь

?>

<h1> значение переменной: <?php echo $num;?> </h1>

</body>

</html>



Также можно присваивать значение другой переменной:

<!DOCTYPE html>

<html>

<head>

<title>test vars</title>

<meta charset="utf-8" />

</head>

<body>

<?php

$a = 15;

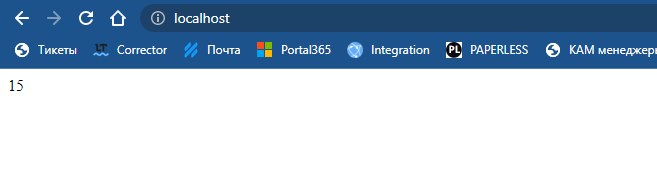
$b = $a;

echo $b;

?>

</body>

</html>



Если переменная объявлена, но ей изначально не присвоено никакого значения (иначе говоря она не инициализирована), то нам будет проблематично ее использовать. Например:

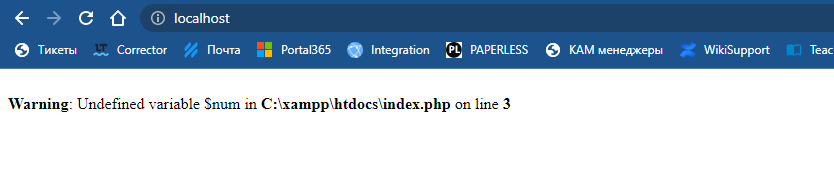
<?php

$num;

echo $num;

?>

При попытке вывести значение переменной мы получим диагностическое сообщение о том, что переменная не определена:



Вывод значения переменной

В предыдущих примерах для вывода значения переменной применялась команда echo, после которой указывалось выводимое значение. Однако есть и другой способ вывести значение переменной. Например, мы хотим одновременно вывести значения двух переменных:

<?php

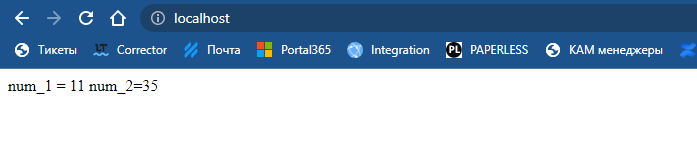
$num\_1 = 11;

$num\_2 = 35;

echo "num\_1 = $num\_1  num\_2=$num\_2";

?>

Здесь функции echo передается строка. Чтобы встроить в строку значение переменной, в этой строке указываем имя переменной вместе со знаком $. И кода в строке PHP встретит выражение $num\_1, он заменит это выражение значением переменной $num\_1. То же самое касается и переменной $num\_2. В итоге при выполнении этого скрипта браузер отобразит значения обеих переменных:



Типы данных

Языки программирования делятся на языки с динамической (dynamic typing) и статической типизацей (static typing). Статическая типизация — когда тип переменной точно известен в compile-time, например в Clang int a = 1; и в run-time тип переменной «a» может быть только integer.

С динамической типизацией попроще. Тип переменной выводится в run-time, например, в PHP $a = 1; и в run-time переменная «$a» может быть числом 1 или даже строкой «string». Тип переменной обуславливает операции, которые мы можем с ней производить, например, числа – складывать, а в строках искать вхождение подстроки.

Типизация, в свою очередь, делится на сильную (strong typing) и слабую (weak typing). Когда мы выполняем какую-либо операцию при сильной типизации, то два операнда должны иметь один и тот же тип, например в Python 1 + “string” вызовет ошибку, потому что 1 – число, а «string» – строка.

А вот в языках со слабой типизацией ошибки не будет, например в PHP 1 + “string” вернёт цифру 1. И причиной всему является приведение типов. Интерпретатор PHP начинает парсить строку «string», чтобы привести её к числу. Напарсить ничего стоящего у него не получается и он приводит строку «string» к цифре 0, которое в итоге и прибавляется к цифре 1. Строку «42string» интерпретатор приведёт к числу 42, а если надо привести строку «47.4string» к числу с плавающей точкой, то мы получим 47.4.

Таким образом, PHP – это язык со слабой динамической типизацией. Информация о выведенном типе переменной хранится в структуре «\_zval\_struct», в этой же структуре хранится ссылка на union-структуру zvalue\_value (zend\_value в PHP7), где и хранится реальное значение, на которое и указывает переменная.

Знание того, как устроен и работает zend\_value помогает ответить на такие вопросы, как, например, «А какой тип будет у переменной «$a», которая ссылается на самое большое число (константа PHP\_INT\_MAX), если её сложить саму с собой (тип intger поменяется на double)?».

Это значит, что тип данных переменной выводится во время выполнения, и в отличии от ряда других языков программирования в PHP не надо указывать перед переменной тип данных.

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. Поэтому вначале именно их и рассмотрим.

int (целочисленный тип)

**32-битные сборки PHP:**

* Целые числа могут быть от **-2 147 483** 648 до **2 147 483 647** (~ ± 2 миллиарда)

**64-битные сборки PHP:**

* Целые числа могут быть от **-9 223 372 036 854** 775 808 до **9 223 372 036 854** 775 807 (~ ± 9 квинтиллионов)

Цифры включительно.

*Примечание: некоторые 64-битные сборки когда-то использовали 32-битные целые числа, особенно старые сборки PHP для Windows.*

Значения за пределами этих диапазонов представлены значениями с плавающей запятой, как и нецелые значения в этих диапазонах. Интерпретатор автоматически определит, когда должно произойти это переключение на плавающую запятую, исходя из того, не может ли значение результата вычисления быть представлено в виде целого числа.

PHP не поддерживает целые числа без знака как таковые, ограничивая максимальное значение всех целых чисел диапазоном целого числа со знаком.

Представляет целое число со знаком.

<?php

$num = -100;

echo $num;

?>

Здесь переменная $num представляет целочисленный тип, так как ей присваивается целочисленное значение.

Кроме десятичных целых чисел PHP обладает возможностью использовать также двоичные, восьмеричные и шестнадцатеричные числа. Шаблоны чисел для других систем:

шестнадцатеричные : 0[xX][0-9a-fA-F]

восьмеричные : 0[0-7]

двоичные : 0b[01]

Например:

<?php

// Все числа в десятичной системе имеют значение 28

$num\_10 = 28; // десятичное число

$num\_2 = 0b11100; // двоичное число (28 в десятичной системе)

$num\_8 = 034; // восьмеричное число (28 в десятичной)

$num\_16 = 0x1C; // шестнадцатиричное число (28 в десятичной)

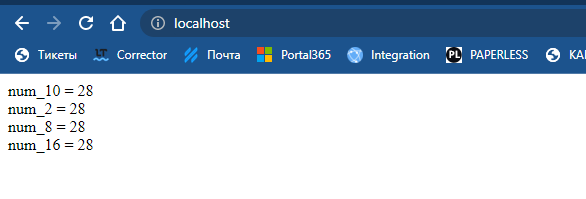
echo "num\_10 = $num\_10 <br>";

echo "num\_2 = $num\_2 <br>";

echo "num\_8 = $num\_8 <br>";

echo "num\_16 = $num\_16";

?>



<?php

$a = 1234; // десятичное число

$a = 0123; // восьмеричное число (эквивалентно 83 в десятичной системе)

$a = 0o123; // восьмеричное число (начиная с PHP 8.1.0)

$a = 0x1A; // шестнадцатеричное число (эквивалентно 26 в десятичной системе)

$a = 0b11111111; // двоичное число (эквивалентно 255 в десятичной системе)

$a = 1\_234\_567; // десятичное число (с PHP 7.4.0)

?>

Переменная типа int занимает в памяти 32 бита, то есть может принимаь значения от -2 147 483 648 до 2 147 483 647. Если переменная получает числовое значение вне этого диапазона, то она трактуется как переменная типа float

Тип float (числа с плавающей точкой)

Размер числа с плавающей точкой зависит от платформы. Максимально возможное значение, как правило, составляет 1.8\*10308 с точностью около 14 десятичных цифр. Например:

<?php

$a1 = 1.5;

$a2 = 1.3e4; // 1.3 \* 10^4

$a3 = 6E-8; // 0.00000006

?>

**Экспоненциальная запись** очень полезна для написания длинных чисел в краткой форме. Числа в экспоненциальной записи имеют следующий вид: **мантисса × 10экспонент**. Например, рассмотрим **выражение** 1.2 × 104. Значение 1.2 — это **мантисса**(или ***«значащая часть числа»***), а 4 — это **экспонент**(или ***«порядок числа»***). Результатом этого выражения является значение 12000.

Обычно, в экспоненциальной записи, в целой части находится только одна цифра, все остальные пишутся после разделительной точки (в дробной части).

Рассмотрим массу Земли. В десятичной системе счисления она представлена как 5973600000000000000000000 кг. Согласитесь, очень большое число (даже слишком большое, чтобы поместиться в целочисленную переменную размером 8 байт). Это число даже трудно читать (там 19 или 20 нулей?). Но используя экспоненциальную запись, массу Земли можно представить, как 5.9736 × 1024кг (что гораздо легче воспринимается, согласитесь). Еще одним преимуществом экспоненциальной записи является сравнение двух очень больших или очень маленьких чисел — для этого достаточно просто сравнить их экспоненты.

В языке C++ буква е /Е означает, что число 10 нужно возвести в степень, которая следует за этой буквой. Например, 1.2 × 104 эквивалентно 1.2e4, значение 5.9736 × 1024 еще можно записать как 5.9736e24.

Для чисел меньше единицы экспонент может быть отрицательным. Например, 5e-2 эквивалентно 5 \* 10-2, что, в свою очередь, означает 5 / 102 или 0.05. Масса электрона равна 9.1093822e-31 кг.

На практике экспоненциальная запись может использоваться в операциях присваивания следующим образом:

**Конвертация чисел в экспоненциальную запись**

Для конвертации чисел в экспоненциальную запись необходимо следовать процедуре, указанной ниже:

   Ваш экспонент начинается с нуля.

   Переместите разделительную точку (которая разделяет целую и дробную части) влево, чтобы слева от нее осталась только одна ненулевая цифра:

   каждое перемещение точки влево увеличивает экспонент на 1;

   каждое перемещение точки вправо уменьшает экспонент на 1.

   Откиньте все нули перед первой ненулевой цифрой в целой части.

   Откиньте все конечные нули в правой (дробной) части, *только если исходное число является целым* *(без разделительной точки)*.

Рассмотрим примеры:

Исходное число: 42030  
Перемещаем разделительную точку на 4 цифры влево: 4.2030e4  
Слева (в целой части) нет нулей: 4.2030e4  
Отбрасываем конечный нуль в дробной части: 4.203e4 (4 значащие цифры)

Исходное число: 0.0078900  
Перемещаем разделительную точку на 3 цифры вправо: 0007.8900e-3  
Отбрасываем нули слева: 7.8900e-3  
Не отбрасываем нули справа (исходное число является дробным): 7.8900e-3 (5 значащих цифр)

Исходное число: 600.410  
Перемещаем разделительную точку на 2 цифры влево: 6.00410e2  
Слева нет нулей: 6.00410e2  
Нули справа оставляем: 6.00410e2 (6 значащих цифр)

Самое главное, что нужно запомнить — это то, что цифры в мантиссе (часть перед e) называются **значащими цифрами**. Количество значащих цифр определяет **точность** самого значения. Чем больше цифр в мантиссе, тем точнее значение.

Тип bool (логический тип)

Переменные логического типа могут принимать два значения: true и false или иначе говоря истина и ложь. Чаще всего логические значения используются в условных конструкциях:

<?php

$foo = true;

$boo = false;

?>

Тип string (строки)

Для работы с текстом можно применять строки. Строки бывают двух типов: в двойных кавычках и одинарных. От типа кавычек зависит обработка строк интерпретатором. Так, переменные в двойных кавычках заменяются значениями, а переменные в одинарных кавычках остаются неизменными.

<?php

$a=10;

$b=5;

$result = "$a+$b <br>";

echo $result;

$result = '$a+$b';

echo $result;

?>

В этом случае мы получим следующий вывод:

10+5

$a+$b

Кроме обычных символов, строка может содержать специальные символы, которые могут быть неправильно интерпретированы. Например, нам надо добавить в строку кавычку:

<?php

$text = "Модель "Apple II"";

?>

Данная запись будет ошибочна. Чтобы исправить ошибку, мы можем сочетать различные типы кавычек ('Модель "Apple II"' или "Модель 'Apple III'") или использовать слеш, чтобы ввести кавычку в строку:

<?php

$text = "Модель \"Apple II\"";

?>

Специальное значение null

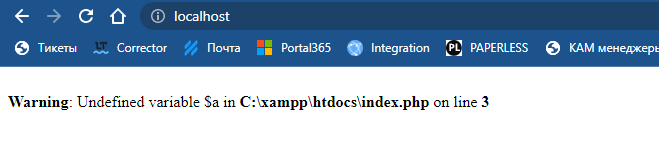
Значение null указывает, переменная не имеет значения. Использование данного значения полезно в тех случаях, когда мы хотим указать, что переменная не имеет значения. Например, если мы просто определим переменную без ее инициализации, и затем попробуем ее использовать, то нам интерпретатор выдаст сообщение, что переменная не установлена:

<?php

$a;

echo $a;

?>



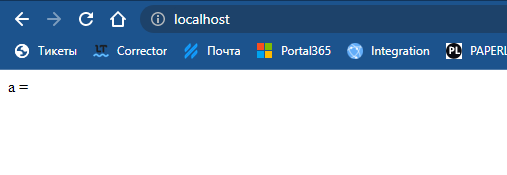
Использование значения null поможет избежать данной ситуации. Кроме того, мы сможем проверять наличие значения и в зависимости от результатов проверки производить те или иные действия:

<?php

$a = null;

echo "a = $a";

?>



Константа null не чувствительна к регистру, поэтому мы можем написать и так:

<?php

$a = NULL;

?>

Динамическая типизация

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

<?php

$id = 123;

echo "<p>id = $id</p>";

$id = "jhveruuyeru";

echo "<p>id = $id</p>";

?>

Константы

Константы, как и переменные хранят определенное значение, только в отличие от переменных значение констант может быть установлено только один раз, и далее мы уже не можем его изменить. Константы обычно определяются для хранения значений, которые должны оставаться неизменными на протяжении всей работы скрипта.

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;

?>

Обычно названия констант используют заглавные символы, но это условность.

После определения константы мы можем ее использовать также, как и обычную переменную.

PHP позволяет устанавливать значения констант на основе вычисляемых выражений:

<?php

const PI = 2.1415 + 1;

echo PI;        // 3.1415

?>

Единственное исключение - мы не сможем изменить значение константы. То есть выражение PI = 3.1415;, которое должно изменить значение константы, не срабатает.

Функция define

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

<?php

    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.

Например, определим числовую константу:

<?php

define("NUMBER", 22);

echo NUMBER;    // 22

?>

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

Например, выведем текущую выполняемую строку и название файла:

<?php

echo "Cтрока " . \_\_LINE\_\_ . " в файле " . \_\_FILE\_\_;

?>

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

<?php

const PI = 3.14;

if (!defined("PI"))

    define("PI", 3.14);

else

    echo "Константа PI уже определена";

?>

Константа - это идентификатор (имя) для простого значения. Как следует из названия, это значение не может измениться в ходе выполнения скрипта (кроме магических констант, которые на самом деле не являются константами). Константы чувствительны к регистру. По принятому соглашению, имена констант всегда пишутся в верхнем регистре.

Имя константы должно соответствовать тем же правилам именования, что и другие имена в PHP. Правильное имя начинается с буквы или символа подчёркивания, за которым следует любое количество букв, цифр и символов подчёркивания. Регулярное выражение для проверки правильности имени константы выглядит так: ^[a-zA-Z\_\x80-\xff][a-zA-Z0-9\_\x80-\xff]\*$

<?php

// Правильные имена констант

define("FOO",     "что-то");

define("FOO2",    "что-то ещё");

define("FOO\_BAR", "что-то большее");

// Неправильные имена констант

define("2FOO",    "что-то");

// Это верное объявление, но лучше его не использовать:

// PHP однажды может зарегистрировать волшебную константу,

// которая нарушит работу скрипта

define("\_\_FOO\_\_", "что-то");

?>

<?php

define('MIN\_VALUE', '0.0');   // RIGHT - Works OUTSIDE of a class definition.

define('MAX\_VALUE', '1.0');   // RIGHT - Works OUTSIDE of a class definition.

//const MIN\_VALUE = 0.0;         RIGHT - Works both INSIDE and OUTSIDE of a class definition.

//const MAX\_VALUE = 1.0;         RIGHT - Works both INSIDE and OUTSIDE of a class definition.

?>

Операции в PHP

В PHP мы можем использовать различные операторы: арифметические, логические и т.д. Рассмотрим каждый тип операций.

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:

<?php

$a = 12;

echo $a % 5; // равно 2

?>

\*\* (возведение в степень)

Например, $a \*\* 2

Примеры операций:

<?php

$a = 8 + 2; // 10, сложение

$a = 8 - 2; // 6, вычитание

$a = 8 \* 2; // 16, умножение

$a = 8 / 2; // 4, деление

$a = 8 % 2; // 0, деление по модулю

$a = 8 \*\* 2; // 64, возведение в степень

?>

Инкремент и декремент

Отдельно следует сказать операции инкремента и декремента, которые также являются арифметическими операциями, но производятся над одним операндом.

Инкремент - операция ++ увеличивает число на единицу. Например, ++$a

Есть два типа инкремента: префиксный инкремент (++$a) и постфиксный ($a++). Важно понимать разницу между этими операциями. Рассмотрим сначала префиксный инкремент:

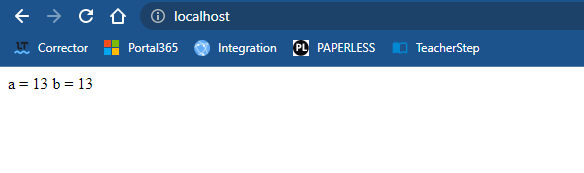
<?php

$a = 12;

$b = ++$a; // $b равно 13

echo "a = $a   b = $b";

?>



Здесь сначала к значению переменной $a прибавляется единица, а затем ее значение приравнивается переменной $b.

Теперь посмотрим, что будет в случае с постфиксным инкрементом:

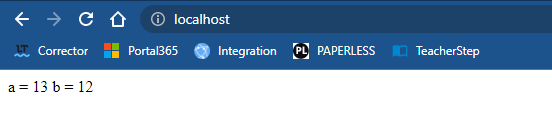
<?php

$a = 12;

$b = $a++; // $b равно 12

echo "a = $a   b = $b";

?>



Здесь сначала значение переменной $a передается переменной $b, а затем происходило увеличение значения переменной $a.

Декремент - операция -- представляет уменьшение значения на единицу. Она аналогично инкременту бывает префиксной и постфиксной и работает аналогично. Например, префиксный декремент:

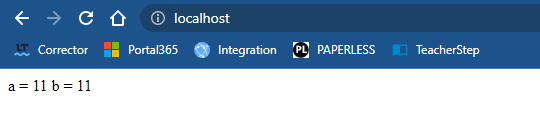
<?php

$a = 12;

$b = --$a; // $b равно 11

echo "a = $a   b = $b";

?>



Сначала значение переменной $a уменьшается на единицу, а затем ее значение приравнивается переменной $b.

Постфиксный декрементом:

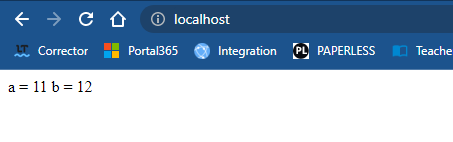
<?php

$a = 12;

$b = $a--; // $b равно 12

echo "a = $a   b = $b";

?>



Здесь же сначала значение переменной $a передается переменной $b и только после этого уменьшается на единицу.

Объединение строк

Для объединения строк используется оператор "точка". Например, соединим несколько строк:

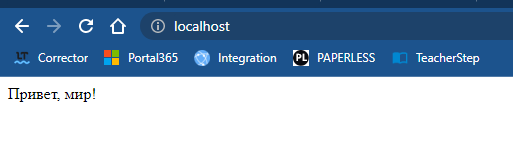
<?php

$a="Привет, ";

$b=" мир";

echo $a . " " . $b . "!";   // Привет мир!

?>



Если переменные представляют не строки, а другие типы, например, числа, то их значения преобразуются в строки и затем также происходит операция объединения строк.

Операции сравнения

Операции сравнения, как правило, применяются в условных конструкциях, когда надо сравнивать два значения, и в зависимости от результата сравнения выполнить некоторые действия. Имеются следующие операции сравнения.

==

Оператор равенства сравнивает два значения, и если они равны, возвращает true, иначе возвращает false: $a == 5

===

Оператор тождественности также сравнивает два значения, и если они равны, возвращает true, иначе возвращает false: $a === 5

!=

Сравнивает два значения, и если они не равны, возвращает true, иначе возвращает false: $a != 5

!==

Сравнивает два значения, и если они не равны, возвращает true, иначе возвращает false: $a !== 5

>

Сравнивает два значения, и если первое больше второго, то возвращает true, иначе возвращает false: $a > 5

<

Сравнивает два значения, и если первое меньше второго, то возвращает true, иначе возвращает false: $a < 5

>=

Сравнивает два значения, и если первое больше или равно второму, то возвращает true, иначе возвращает false: $a >= 5

<=

Сравнивает два значения, и если первое меньше или равно второму, то возвращает true, иначе возвращает false: $a <= 5

Оператор равенства и тождественности

Оба оператора сравнивают два выражения и возвращают true, если выражения равны. Но между ними есть различия. Если в операции равенства принимают два значения разных типов, то они приводятся к одному - тому, который интерпретатор найдет оптимальным. Например:

<?php

$a = (2 == "2");    // true (значения равны)

$b = (2 === "2");   // false (значения представляют разные типы)

?>

Строка "2" по сути представляет то же значение, что и число 2, оператор сравнения возвратит true. Однако они представляют разные типы, поэтому оператор тождественности возвратит false

Аналогично работают операторы неравенства != и !==.

<?php

$a = (2 != "2");    // false, так как значения равны

$b = (2 !== "2");   // true, так как значения представляют разные типы

?>

Оператор <=>

Отдельно стоит сказать про оператор <=>. Он также сравнивает два значения и возвращает

0, если оба значения равны

1, если значение слева больше, чем значение справа

–1, если значение слева меньше, чем значение справа

Применение:

<?php

$a = 2 <=> 2;     // 0    (эквивалентно 2 == 2)

$b = 3 <=> 2;     // 1    (эквивалентно 3 > 2)

$c = 1 <=> 2;     // -1   (эквивалентно 1 < 2)

echo "a=$a   b=$b   c=$c";  // a=0  b=1  c=-1

?>

Логические операции

Логические операции обычно применяются для объединения результатов двух операций сравнения. Например, нам надо выполнить определенное действие, если истинны будут несколько условий. Имеются следующие логические операции:

&&

Возвращает true, если обе операции сравнения возвращают true, иначе возвращает false: $a && $b

and

Аналогично операции &&: $a and $b

||

Возвращает true, если хотя бы одна операция сравнения возвращают true, иначе возвращает false: $a || $b

or

Аналогично операции ||: $a or $b

!

Возвращает true, если операция сравнения возвращает false: !$a

xor

Возвращает true, если только одно из значений равно true. Если оба равны true или ни одно из них не равно true, возвращает false: $a xor $b

Примеры логических операций:

<?php

$a = (true && false);   // false

// аналогично

$a = (true and false);  // false

$b = (true || false); // true

// аналогично следующей операции

$b = (true or false); // true

$c = !true;           // false

?>

Отдельно примеры с операцией xor:

<?php

$a = (true xor true);   //  false

$b = (false xor true);  //  true

$c = (false xor false); //  false

?>

Поразрядные операции

Поразрядные операции производятся над отдельными разрядными или битами числа. Числа рассматриваются в двоичном представлении, например, 2 в двоичном представлении 010, число 7 - 111.

& (логическое умножение)

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

<?php

$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:

<?php

$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:

<?php

$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.

<?php

$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

Операции присваивания

=

Приравнивает переменной определенное значение: $a = 5

+=

Сложение с последующим присвоением результата. Например:

<?php

$a=12;

$a += 5;

echo $a; // равно 17

?>

-=

Вычитание с последующим присвоением результата. Например:

<?php

$a=12;

$a -= 5;

echo $a; // равно 7

?>

\*=

Умножение с последующим присвоением результата:

<?php

$a=12;

$a \*= 5;

echo $a; // равно 60

?>

/=

Деление с последующим присвоением результата:

<?php

$a=12;

$a /= 5;

echo $a; // равно 2.4

?>

.=

Объединение строк с присвоением результата. Применяется к двум строкам. Если же переменные хранят не строки, а, к примеру, числа, то их значения преобразуются в строки и затем проводится операция:

<?php

$a=12;

$a .= 5;

echo $a; // равно 125

// идентично

$b="12";

$b .="5"; // равно 125

?>

%=

Получение остатка от деления с последующим присвоением результата:

<?php

$a=12;

$a %= 5;

echo $a; // равно 2

?>

\*\*=

Получение результата от возведения в степень:

<?php

$a=8;

$a \*\*= 2;

echo $a; // равно 64 (8 в степени 2)

?>

&=

Получение результата от операции логического умножения:

<?php

$a=5; $a &= 4; // 101&100=100 - 4

?>

|=

Получение результата от операции логического сложения:

<?php

$a=5; $a |= 4; // 101|100=101 - 5

?>

^=

Получение результата от операции исключающего ИЛИ:

<?php

$a=5; $a ^= 4; // 101^100=001 - 1

?>

<<=

Получение результата от операции сдвига влево:

<?php

$a=8; $a <<= 1; // 1000 << 1 = 10000 - 16

?>

>>=

Получение результата от операции сдвига вправо:

<?php

$a=8; $a >>= 1; // 1000 >> 1 = 100 - 4

?>

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:

| **Порядок выполнения операторов** | | |
| --- | --- | --- |
| **Ассоциативность** | **Оператор** | **Дополнительная информация** |
| (н/а) | clone new | [clone](https://www.php.net/manual/ru/language.oop5.cloning.php) и [new](https://www.php.net/manual/ru/language.oop5.basic.php#language.oop5.basic.new) |
| правая | \*\* | [арифметические операторы](https://www.php.net/manual/ru/language.operators.arithmetic.php) |
| (н/а) | + - ++ -- ~ (int) (float) (string) (array) (object) (bool) @ | [арифметические операторы](https://www.php.net/manual/ru/language.operators.arithmetic.php) (унарные + и -), [инкремент/декремент](https://www.php.net/manual/ru/language.operators.increment.php), [побитовые операторы](https://www.php.net/manual/ru/language.operators.bitwise.php), [приведение типов](https://www.php.net/manual/ru/language.types.type-juggling.php#language.types.typecasting) и [оператор управления ошибками](https://www.php.net/manual/ru/language.operators.errorcontrol.php) |
| левая | instanceof | [типы](https://www.php.net/manual/ru/language.operators.type.php) |
| (н/а) | ! | [логические операторы](https://www.php.net/manual/ru/language.operators.logical.php) |
| левая | \* / % | [арифметические операторы](https://www.php.net/manual/ru/language.operators.arithmetic.php) |
| левая | + - . | [арифметические операторы](https://www.php.net/manual/ru/language.operators.arithmetic.php) (бинарные + и -), [операторы, работающие с массивами](https://www.php.net/manual/ru/language.operators.array.php) и [строковые операторы](https://www.php.net/manual/ru/language.operators.string.php) (. до PHP 8.0.0) |
| левая | << >> | [побитовые операторы](https://www.php.net/manual/ru/language.operators.bitwise.php) |
| левая | . | [строковые операторы](https://www.php.net/manual/ru/language.operators.string.php) (начиная с PHP 8.0.0) |
| неассоциативна | < <= > >= | [операторы сравнения](https://www.php.net/manual/ru/language.operators.comparison.php) |
| неассоциативна | == != === !== <> <=> | [операторы сравнения](https://www.php.net/manual/ru/language.operators.comparison.php) |
| левая | & | [побитовые операторы](https://www.php.net/manual/ru/language.operators.bitwise.php) и [ссылки](https://www.php.net/manual/ru/language.references.php) |
| левая | ^ | [побитовые операторы](https://www.php.net/manual/ru/language.operators.bitwise.php) |
| левая | | | [побитовые операторы](https://www.php.net/manual/ru/language.operators.bitwise.php) |
| левая | && | [логические операторы](https://www.php.net/manual/ru/language.operators.logical.php) |
| левая | || | [логические операторы](https://www.php.net/manual/ru/language.operators.logical.php) |
| правая | ?? | [операторы сравнения с null](https://www.php.net/manual/ru/language.operators.comparison.php#language.operators.comparison.coalesce) |
| неассоциативна | ? : | [тернарный оператор](https://www.php.net/manual/ru/language.operators.comparison.php#language.operators.comparison.ternary) (лево-ассоциативный до PHP 8.0.0) |
| правая | = += -= \*= \*\*= /= .= %= &= |= ^= <<= >>= ??= | [операторы присваивания](https://www.php.net/manual/ru/language.operators.assignment.php) |
| (н/а) | yield from | [yield from](https://www.php.net/manual/ru/language.generators.syntax.php#control-structures.yield.from) |
| (н/а) | yield | [yield](https://www.php.net/manual/ru/language.generators.syntax.php#control-structures.yield) |
| (н/а) | print | [print](https://www.php.net/manual/ru/function.print.php) |
| левая | and | [логические операторы](https://www.php.net/manual/ru/language.operators.logical.php) |
| левая | xor | [логические операторы](https://www.php.net/manual/ru/language.operators.logical.php) |
| левая | or | [логические операторы](https://www.php.net/manual/ru/language.operators.logical.php) |

Чем выше оператор в этой таблице, тем больше его приоритет. Например:

<?php

$a = 10 + 5 \* 2;    // 20

?>

Операция умножения имеет больший приоритет, чем сложение. Поэтому сначала выполняется подвыражение 5 \* 2

При этом скобки повышают приоритет операции, используемой в выражении.

<?php

$a = (10 + 5) \* 2;  // 30

?>

В данном случае сначала выполняется подвыражение 10 + 5 и только потом умножение.

Конструкция if..else и тернарная операция

Условные конструкции позволяют направлять работу программы в зависимости от условия по одному из возможных путей. И одной из таких конструкций в языке PHP является конструкция if..else

Конструкция if..else

Конструкция if (условие) проверяет истинность некоторого условия, и если оно окажется истинным, то выполняется блок выражений, стоящих после if. Если же условие ложно, то есть равно false, тогда блок if не выполняется. Например:

<?php

$a = 4;

if($a>0){

    echo "Переменная a больше нуля";

}

echo "<br>конец выполнения программы";

?>

            $number = (int)$\_GET['number'];

            echo "Your entered: $number";

            if($number === 90)

                echo "your number: $number";

            else if($number === 80)

                echo "your number: $number";

            else if($number === 95)

                echo "your number: $number";

            else

                echo "I don't know your number!";

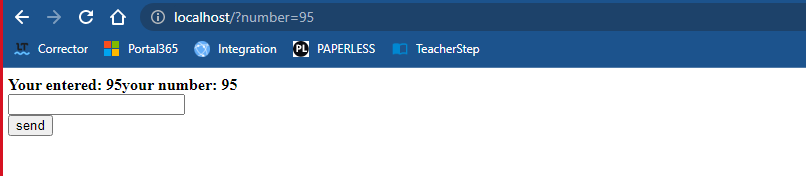
        ?>

    <form action="">

        <input type="text" name="number"><br>

        <input type="submit" value="send"><br>

    </form>



Блок выражений ограничивается фигурными скобками. И так как в данном случае условие истинно (то есть равно true): значение переменной $a больше 0, то блок инструкций в фигурных скобках также будет выполняться. Если бы значение $a было бы меньше 0, то блок if не выполнялся.

Если блок if содержит всего одну инструкцию, то можно опустить фигурные скобки:

<?php

$a = 4;

if($a>0)

    echo "Переменная a больше нуля";

echo "<br>конец выполнения программы";

?>

Можно в одной строке поместить всю конструкцию:

<?php

if($a>0) echo "Переменная a больше нуля";

?>

В данном случае к блоку if относится только инструкция echo "Переменная a больше нуля";

else

Блок else содержит инструкции, которые выполняются, если условие после if ложно, то есть равно false:

<?php

$a = 4;

if($a > 0){

    echo "Переменная a больше нуля";

}

else{

    echo "Переменная a меньше нуля";

}

echo "<br>конец выполнения программы";

?>

Если $a больше 0, то выполняется блок if, если нет, то выполняется блок else.

Поскольку здесь в обоих блоках по одной инструкции, также можно было не использовать фигурные скобки для определения блоков:

<?php

if($a > 0)

echo "Переменная a больше нуля";

else

echo "Переменная a меньше нуля";

?>

Специфика php как и в JS

            //Don't work!

            $result = 0.1 + 0.2;

            if($result === 0.3) // false

                echo "Worked!";

            //Work!

            $result = 0.1 + 0.2;

            if(round($result, 1) === 0.3) // true

                echo "Worked!";

            $result = 0.2 + 0.2;

            if($result === 0.4) // true

                echo "Worked!";

<https://stackoverflow.com/questions/33726153/why-can-php-calculate-0-1-0-2-when-other-languages-fail>

elseif

Конструкция elseif вводит дополнительные условия в программу:

<?php

$a = 5;

if($a>0){

    echo "Переменная a больше нуля";

}

elseif($a < 0){

    echo "Переменная a меньше нуля";

}

else{

    echo "Переменная a равна нулю";

}

?>

Можно добавить множество блоков elseif. И если ни одно из условий в if или elseif не выполняется, тогда срабатывает блок else.

Определение условия

Выше в качестве условия применялись операции сравнения. Однако в реальности в качестве условия может применяться любое выражение, а не только такое, которое возвращает true или false. Если передаваемое выражение равно 0, то оно интерпретируется как значение false. Другие значения рассматриваются как true:

<?php

if (0) {}       // false

if (-0.0) {}    // false

if (-1) {}      // true

if ("") {}      // false (пустая строка)

if ("a") {}     // true (непустая строка)

if (null) {}    // false (значие отсутствует)

?>

Альтернативный синтаксис if

PHP также поддерживает альтернативный синтаксис для конструкции if..else, при которой вместо открывающей фигурной скобки ставится двоеточие, а в конце всей конструкции ставится ключевое слово endif.

<?php

$a = 5;

if($a > 0):

    echo "Переменная a больше нуля";

elseif($a < 0):

    echo "Переменная a меньше нуля";

else:

    echo "Переменная a равна нулю";

endif;

?>

Комбинированный режим HTML и PHP

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

<!DOCTYPE html>

<html>

<head>

<title>Combo</title>

<meta charset="utf-8" />

</head>

<body>

<?php

$a = 5;

?>

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

<h2>Переменная a больше нуля</h2>

<?php } ?>

</body>

</html>

В данном случае само условие указывется в отдельном блоке php: <?php if ($a > 0) { ?>. Важно, что при этом этот блок содержит только открывающую фигурную скобку "{".

Завершается конструкция if другим блоком php, который содержит закрывающую фигурную скобку: <?php } ?>

Между этими двумя блоками php располагается код, который отображается на html-странице, если условие в if истинно. Причем этот код представляет именно код html, поэтому здесь можно разместить различные элементы html, как в данном случае элемент <h2>

Еще пример:

    <?php

        $result = 52;

    ?>

    <?php if($result === 52) {?>

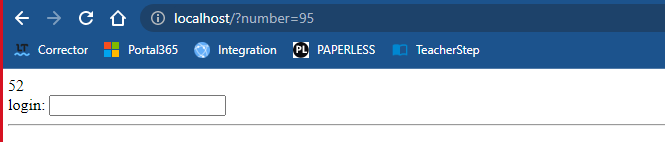
        <form action="">

            <label for="login">login: </label>

            <input type="text" name="login"><hr>

        </form>

    <?php }?>



При необходимости можно добавить выражения else и elseif:

<!DOCTYPE html>

<html>

<head>

<title> Combo </title>

<meta charset="utf-8" />

</head>

<body>

<?php

$a = -5;

?>

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

<h2>Переменная a больше нуля</h2>

<?php } elseif($a < 0) { ?>

<h2>Переменная a меньше нуля</h2>

<?php } else { ?>

<h2>Переменная a равна нулю</h2>

<?php } ?>

</body>

</html>

Также можно применять альтернативный синтаксис:

<!DOCTYPE html>

<html>

<head>

<title> Combo </title>

<meta charset="utf-8" />

</head>

<body>

<?php

$a = 0;

?>

<?php if ($a > 0): ?>

<h2>Переменная a больше нуля</h2>

<?php elseif($a < 0): ?>

<h2>Переменная a меньше нуля</h2>

<?php else: ?>

<h2>Переменная a равна нулю</h2>

<?php endif; ?>

</body>

</html>

Тернарная операция

Тернарная операция состоит из трех операндов и имеет следующее определение: [первый операнд - условие] ? [второй операнд] : [третий операнд]. В зависимости от условия тернарная операция возвращает второй или третий операнд: если условие равно true, то возвращается второй операнд; если условие равно false, то третий. Например:

<?php

$a = 1;

$b = 2;

$z = $a < $b ? $a + $b : $a - $b;

echo $z;

?>

Если значение переменной $a меньше $b и условие истинно, то переменная $z будет равняться $a + $b. Иначе значение $z будет равняться $a - $b

Конструкции switch и match

Конструкция switch..case

Конструкция switch..case является альтернативой использованию конструкции if..elseif..else. Оператор switch получает некоторое выражение и сравнивает его с набором значений:

<?php

switch (выражение){

    case значение1: действия; break;

    case значение2: действия; break;

    //...........................

    case значениеN: действия; break;

}

?>

Можно указать без break:

    <?php

    $result = 52;

        switch ($result) {

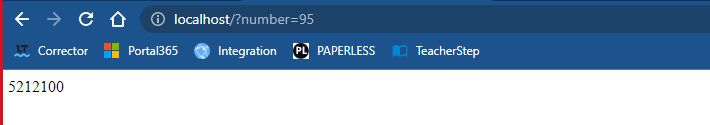
            case 52: echo "52";

            case 12: echo "12";

            case 100: echo "100";

        }

    ?>



После ключевого слова switch в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями, помещенными после операторов сase. И если совпадение будет найдено, то будет выполняться определенный блок сase.

Конструкция switch может содержать произвольное количество операторов case. В конце блока сase ставится оператор break, чтобы избежать выполнения других блоков.

Весь блок switch помещается в фигурные скобки, однако блок каждого отдельного оператора case НЕ помещается в фигурные скобки.

Например, возьмем следующую конструкцию if..elseif..else:

<?php

$a = 3;

if($a==1)     echo "сложение";

elseif($a==2) echo "вычитание";

elseif($a==3) echo "умножение";

elseif($a==4) echo "деление";

?>

Теперь перепишем ее с помощь конструкции switch..case:

<?php

$a = 3;

switch($a)

{

    case 1:

        echo "сложение";

        break;

    case 2:

        echo "вычитание";

        break;

    case 3:

        echo "умножение";

        break;

    case 4:

        echo "деление";

        break;

}

?>

То есть на вход в конструкцию switch передается переменная $a<, и ее значение последовательно сравнивается со значениями, указанными после операторов case. Так, в данном случае переменная $a равна 3, поэтому выполнение дойдет до блока

<?php

case 3:

    echo "умножение";

    break;

?>

и этот блок и будет выполняться.

Если мы хотим также обработать ситуацию, когда совпадения не будет найдено, то можно добавить блок default:

<?php

$a = 3;

switch($a)

{

    case 1:

        echo "сложение";

        break;

    case 2:

        echo "вычитание";

        break;

    default:

        echo "действие по умолчанию";

        break;

}

?>

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

<?php

$a = 3;

switch($a):

    case 1:

        echo "сложение";

        break;

    case 2:

        echo "вычитание";

        break;

    default:

        echo "действие по умолчанию";

        break;

endswitch

?>

match

Начиная с версии 8.0 в PHP была добавлена поддержка другой, похожей конструкции - match. Она позволяет оптимизировать конструкцию switch. Конструкция match также принимает некоторое выражение и сравнивает его с набором значений.

Например, пусть у нас есть следующий код:

<?php

$a = 2;

switch($a)

{

    case 1:

        $operation = "сложение";

        break;

    case 2:

        $operation = "вычитание";

        break;

    default:

        $operation = "действие по умолчанию";

        break;

}

echo $operation;

?>

Перепишем этот пример с использованием match:

<?php

$a = 2;

$operation = match($a)

{

    1 => "сложение",

    2 => "вычитание",

    default => "действие по умолчанию",

};

echo $operation;

?>

Итак, match в скобках также принимает некоторое сравниваемое выражение (в данном случае это переменная $a). Сам блок кода match также обертывается в фигурные скобки, однако в конце после закрывающей фигурной скобки необходимо поставить точку с запятой. А вместо операторов case просто указываются значения, с которыми сравнивается выражение.

Но в отличие от switch, конструкция match возвращает некоторый результат. Поэтому после каждого сравнимого значения ставится оператор =>, после которого идет возвращаемый результат.

То есть в данном случае переменная $a равна 2, поэтому в конструкции match будет выполнять блок

<?php

2 => "вычитание"

?>

Этот блок установит в качестве возвращаемого результата строку "вычитание".

Поскольку конструкция match возвращает результат, то этот результат мы можем присвоить другой переменной:

<?php

$operation = match($a){

    //.................

}

?>

В итоге в переменной $operation будет храниться строка "вычитание".

Также мы можем переписать предыдущий пример следующим образом:

<?php

$a = 2;

match($a)

{

    1 => $operation = "сложение",

    2 => $operation = "вычитание",

    default => $operation = "действие по умолчанию",

};

echo $operation;

?>

Сравнение значений и типов

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. Рассмотрим разницу на примере. Пусть есть следующая конструкция switch:

<?php

switch (8.0) {

    case "8.0":

      $result = "строка";

      break;

    case 8.0:

      $result = "число";

      break;

  }

  echo $result; // строка

?>

В конструкцию switch передается в качестве выажения число 8.0, но с точки зрения внутренней логики конструкции switch это выражение также соответствует строке "8.0". Поэтому в данном случае будет выполняться блок

<?php

case "8.0":

    $result = "строка";

    break;

?>

Теперь посмотрим, что будет в аналогичном примере с match:

<?php

match (8.0) {

    "8.0" => $result = "строка",

    8.0 => $result = "число"

  };

  echo $result; // число

?>

Конструкция match также будет учитывать тип выражения, а тип в данном случае float, поэтому будет выполняься блок:

<?php

8.0 => $result = "число"

?>

Циклы

Для совершения повторяемых действий в PHP, как и в других языках программирования, используются циклы. В PHP имеются следующие виды циклов:

for

while

do..while

foreach (рассматривается в теме про массивы)

Цикл for

Цикл for имеет следующее формальное определение:

for ([инициализация счетчика]; [условие]; [изменение счетчика])

{

    // действия

}

Рассмотрим стандартный цикл for:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Циклы</title>

</head>

<body>

    <?php

        for ($i = 1; $i < 10; $i++)

        {

            echo "Квадрат числа $i равен " . $i \* $i . "<br/>";

        }

    ?>

</body>

</html>

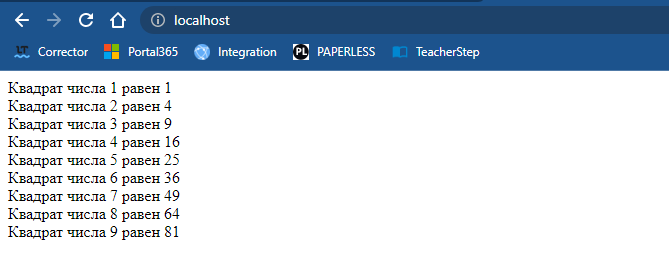
Первая часть объявления цикла - $i = 1 - создает и инициализирует счетчик - переменную $i. И перед выполнением цикла его значение будет равно 1. По сути это то же самое, что и объявление переменной.

Вторая часть - $i < 10; - условие, при котором будет выполняться цикл. В данном случае цикл будет выполняться, пока $i не достигнет 10.

Третья часть - $i++ - изменение счетчика на единицу. Опять же нам необязательно увеличивать на единицу. Можно уменьшать: $i--. Или, например, увеличить не на 1, на 3: $i +=3.

В итоге блок цикла сработает 9 раз, пока значение $i не станет равным 10. И каждый раз это значение будет увеличиваться на 1. Каждое отдельное повторение цикла называется итерацией. Таким образом, в данном случае произойдет 9 итераций.

В итоге браузер отобразит нам следующий результат:



Объявление цикла for может опускать отдельный части. Например, опустить определение счетчика (он может быть определен вне цикла):

<?

$i = 5;

for (; $i < 10; $i++)

{

    echo $i;

}

?>

Можно опустить изменение значения счетчика и изменять его внутри цикла:

<?

$i = 0;

for (; $i < 10;)

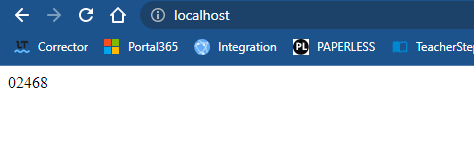
{

    echo $i;

    $i += 2;

}

В данном случае в цикле на каждой итерации переменная $i увеличивает значение на 2. Соответственно мы получим следующий результат:



Также можно в объявлении цикла определять и использовать сразу несколько переменных:

<?

for ($i =1, $j=1; $i + $j < 10; $i++, $j+=2)

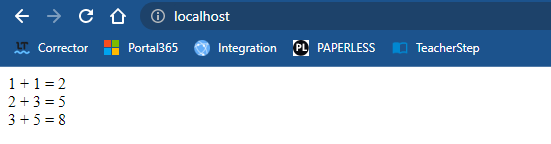
{

    echo "$i + $j = " . $i + $j . "<br>";

}

?>

В данном случае в объявлении цикла определяются две переменных: $i и $j. При каждой итерации переменная $i увеличивается на 1, а $j - на 2. При этом цикл продолжается, пока сумма двух переменных не достигнет 10:



Также можно применять альтернативный синтаксис, при котором вместо открывающей фигурной скобки ставится двоеточие, а вместо закрывающей фигурной скобки - ключевое слово endfor:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Циклы</title>

</head>

<body>

    <?php

        for ($i = 1; $i < 10; $i++):

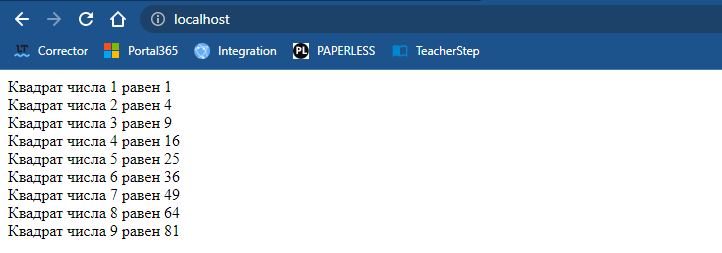
            echo "Квадрат числа $i равен " . $i \* $i . "<br/>";

        endfor;

    ?>

</body>

</html>



Цикл while

Цикл while проверяет истинность некоторого условия, и если условие истинно, то выполняются блок выражений цикла:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Циклы</title>

</head>

<body>

<?php

$counter = 1;

while($counter<10)

{

    echo $counter \* $counter . "<br />";

    $counter++;

}

?>

</body>

</html>

Если в блоке while всего одна инструкция, то фигурные скобки блока можно опустить:

<?php

$counter = 0;

while(++$counter<10)

    echo $counter \* $counter . "<br />";

?>

Также можно применять альтернативный синтаксис, при котором вместо открывающей фигурной скобки ставится двоеточие, а вместо закрывающей фигурной скобки - ключевое слово endwhile:

<?php

$counter = 1;

while($counter<10):

    echo $counter \* $counter . "<br />";

    $counter++;

endwhile;

?>

Цикл do..while

Цикл do..while похож на цикл while, только теперь выполняется блок цикла, и только потом выполняется проверка условия. То есть даже если условие ложно, то блок цикла выполнится как минимум один раз:

<?php

$counter = 1;

do

{

    echo $counter \* $counter . "<br />";

    $counter++;

}

while($counter<10)

?>

Операторы continue и break

Иногда возникает ситуация, когда требуется выйти из цикла, не дожидаясь его завершения. В этом случае мы можем воспользоваться оператором break:

<?php

for ($i = 1; $i < 10; $i++)

{

    $result = $i \* $i;

    if($result>80)

    {

        break;

    }

    echo "Квадрат числа $i равен $result <br/>";

}

?>

И если вдруг результат операции окажется больше 80, то происходит выход из цикла.

Для управления циклами также применяется оператор continue. Он осуществляет переход к следующей итерации цикла:

<?php

for ($i = 1; $i < 10; $i++)

{

    if($i==5)

    {

        continue;

    }

    echo "Квадрат числа $i равен " . $i \* $i . "<br/>";

}

?>

При выполнении программы, когда значение $i станет равным 5, произойдет переход к следующей итерации, а все остальные выражения, которые следуют после оператора continue выполняться не будут.

Вложенные циклы

Циклы могут располагаться внутри циклов. Например, используем вложенный цикл, чтобы вывести таблицу умножения:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Циклы</title>

</head>

<style>

table, th, td {

  border:1px solid black;

}

</style>

<body>

    <table>

<?php

for ($i = 1; $i < 10; $i++)

{

    echo "<tr>";

    for ($j = 1; $j < 10; $j++)

    {

        echo "<td>" . $i \* $j . "</td>";

    }

    echo "</tr>";

}

?>

</table>

</body>

</html>

В данном случае код php помещен в элемент <table>, то есть в нем будет создаваться таблица.

Во внешнем цикле определяется переменная-счетчик $i. В этом - внешнем цикле на страницу выводится тег <tr>, то есть тег строки:

<?php

for ($i = 1; $i < 10; $i++)

{

    echo "<tr>";

Далее запускается вложенный цикл, определяется переменная-счетчик $j. Цикл последовательно увеличивает значение переменной $j на единицу и умножает ее значение на значение переменной $i. Результат выводится в элемент <td>:

<?php

for ($j = 1; $j < 10; $j++)

{

    echo "<td>" . $i \* $j . "</td>";

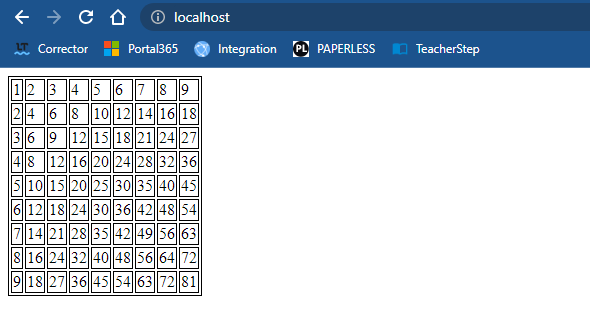
После 9 итераций, когда переменная $j достигнет 10, вложенный цикл завершается, и внешний цикл выводит закрывающий тег строки:

echo "</tr>";

}

И запускается новая итерация внешнего цикла.

В итоге скрипт сформирует таблицу умножения:



<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>test</title>

    <style>

        table, tr, td {

            border: 1px solid green;

        }

        td {

            padding: 5px;

            font-family: sans-serif;

            font-size: 18px;

            color: brown;

        }

        .whiteCell {

            display: inline-block;

            width: 100px;

            height: 100px;

            background-color: gainsboro;

        }

        .blackCell {

            display: inline-block;

            width: 100px;

            height: 100px;

            background-color: black;

        }

    </style>

</head>

<body>

<?php

    $row = 8;

    $col = 8;

    for ($i=0; $i < $row; $i++) {

        for($j=0; $j < $col; $j++) {

            if($j % 2 != 1 && $i % 2 != 1 || $j % 2 == 1 && $i % 2 == 1) // white

                echo "<div class=\"whiteCell\"></div>";

            else

                echo "<div class=\"blackCell\"></div>";

        }

        echo "<br>";

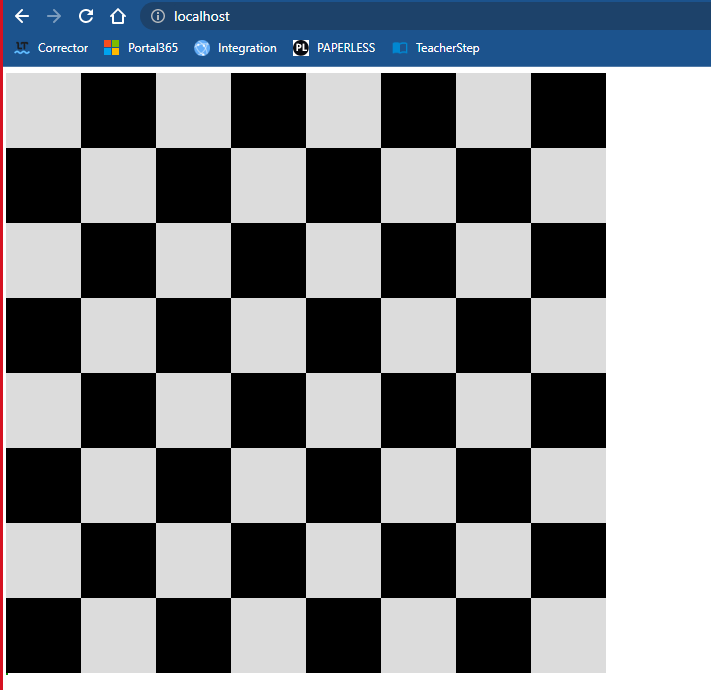
    }

?>

</body>

</html>

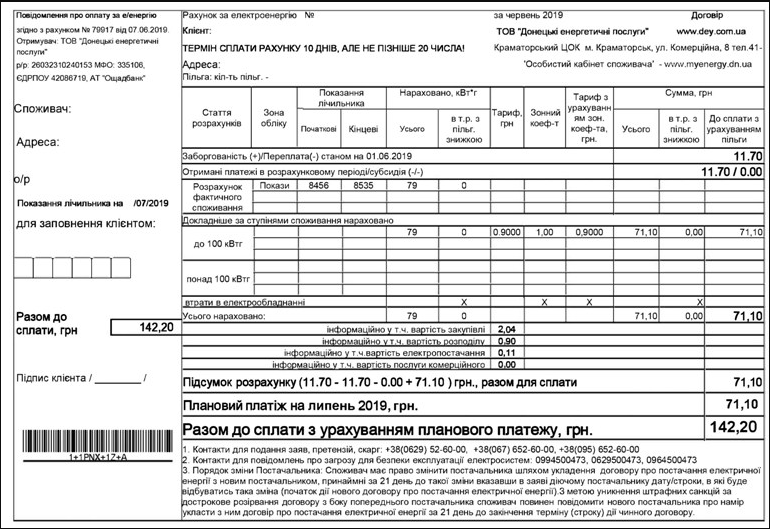
Result:



Материалы к уроку:

1. <https://www.php.net/manual/ru/language.basic-syntax.comments.php>
2. Типы <https://www.php.net/manual/ru/language.types.type-system.php>
3. <https://www.php.net/manual/en/control-structures.if.php>
4. Книга <https://drive.google.com/file/d/1QrZIPGf5UFVajoLiYnwaX4Z83SiPrkkl/view?usp=sharing>
5. <https://addphp.ru/materials/base/1_6.php>
6. <http://www.php.su/learnphp/cs/?co>
7. <https://ru.hexlet.io/courses/php-basics/lessons/if-else/theory_unit>

Домашнее задание:

1. Бухгалтерия вводит нужные данные в поля input, создайте квитанцию на основании введенных данных. Пример квитанции ниже: Как реализовать ввод, есть пример в Уроке 1  
   

Простейшая арифметика

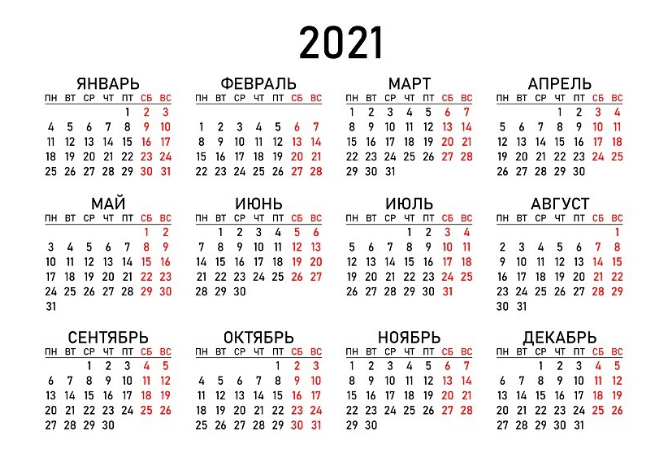
1. Даны два числа. Найти их сумму и произведение.
2. Даны два числа. Найдите сумму их квадратов.
3. Даны три числа. Найдите их среднее арифметическое.
4. Даны три числа x,y и z. Найдите (x+1)4−2(z−2x2+y2)+|siny|−−−−−√
5. Даны три ненулевых числа. Найдите всевозможные результаты деления суммы двух из них на оставшееся третье число.
6. Дано натуральное числа. Найдите остатки от деления этих чисел на 3 и на 5.
7. Дано число. Увеличьте его на 30%, на 120%.
8. Дано два числа. Найдите сумму 40% от первого числа и 84% от второго числа.
9. Дано трехзначное числа. Найдите сумму его цифр.
10. Дано трехзначное числа. Поменяйте среднюю цифру на ноль.
11. Найдите число, полученное выписыванием в обратном порядке цифр данного трехзначного натурального числа.

Условный оператор

1. Дано число. Если оно больше 10, то увеличьте его на 100, иначе уменьшите на 30.
2. Дано натуральное число. Если оно четное, то уменьшите его в 2 раза, иначе увеличьте в 3 раза.
3. Дано число. Если оно не меньше 50, то выведите квадрат этого числа, если же это число больше 10 и меньше 30, то выведите ноль, в остальных случаях выведите слово "Ошибка"
4. Дано два числа. Вывести наибольшее из них.
5. Дано два числа. Вывести 'Да', если они отличаются на 100, иначе вывести 'Нет'
6. Дано два числа. Вывести 'Да', если они отличаются не более чем на 20, иначе вывести 'Нет'.
7. Дан номер месяца. Вывести название поры года (весна, лето и так далее) или слово 'Ошибка', если месяца с таким номером не существует.
8. Вычислите значение выражения x2−4y−1−−−−√sin2x+|x|. Перед вычислением проверить корректность значений переменных.
9. В данном трехзначном числе переставьте цифры так, чтобы новое число оказалось наибольшим из возможных.
10. Решите  уравнение ax2+bx+c=0.
11. Вычислите значение выражения max2(x,y,z)−3x⋅min(x,y,z)sin1+max(x,y−z)/min(y,z). Если вычисление невозможно, то сообщите об этом пользователю.
12. Пользователь в форму вводит 4 значное число от 1 до 9999, сделать вывод на экран этого числа словами.

Например 123 – Сто Двадцать Три

Циклы

1. Пользователь вводит в input год, программа показывает календарь на этот год с подсветкой выходных и праздничных дней. (Примерный пример ниже)  
   Примерный вид календаря:  
   
2. На шахматной доске нарисовать каждую фигуру и все возможные ходы этой фигуры. На одной шахматной доске стоит 1 фигура и программа показует все возможные ходы этой фигуры как на примере ниже, такое же действие нужно сделать для каждой фигуры! Фигуру на доску можно поставить в любое место, функционала перестановки фигуры нет, куда поставить фигуру и показать ее возможности остается за Вами для облегчения задачи. Правила хода шахматных фигур можно посмотреть здесь <https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B0%D0%B2%D0%B8%D0%BB%D0%B0_%D1%88%D0%B0%D1%85%D0%BC%D0%B0%D1%82>

Дополнительно, но необязательно можете создать меню в котором пользователь делает выбор фигуры и указывает координаты куда он хочет ее поставить.  
Например, пользователь выбрал королева и поставил координаты 1d:

