Часть 2. Практическая работа 2

Ветвления и функции

В этом Практическая работа вы:

* научитесь разветвлять вашу программу
* научитесь использовать свой код повторно
* узнаете про области видимости переменных и время их жизни

1. Ветвление программы

Условные операторы позволяют разветвлять код программы. В зависимости от проверяемых условий могут быть выполнены либо одни, либо другие действия. Это важнейший прием программирования. Пожалуй, без проверки условий невозможно представить не одной нормальной программы.

* 1. Оператор if

В самом простом виде использование оператора if выглядит так:

if (Условие) Действие;

Условие - это любое выражение с булевым значением (true или false). Действие выполняется тогда, когда Условие истинно (то есть равно true). В результате проверки какого-либо условия может выполняться сразу несколько операторов, тогда они образуют блок, который должен быть заключен в фигурные скобки:

if ( Условие)

{

Действие1;

Действие2;

}

Часто возникает необходимость указать какие-либо действия не только когда условие верно, но и когда оно ложно. В этом случае используется ключевое слово else:

|  |  |  |
| --- | --- | --- |
| if (Условие) | |  |
| { |  |  |
| }  else  { | Блок | операторов1 |
| } | Блок | операторов2 |

Ключевое слово elseif позволяет осуществлять проверку дополнительных условий, если предыдущие условия оказались ложными. Рассмотрим следующий пример:

if (Условие1) {

Блок операторов1 (Условие2)

}

elseif

{

}

elseif

{

}

else

Блок операторов2 (Условие3)

Блок операторов3 {

Блок операторов4

}

Допустим Условие 1 и Условие2 ложные, а Условие3 истинно. В данном случае интерпретатор PHP проверит сначала Условие 1, убедится что оно ложное, затем Условие2. После этого дойдет до Условия3, увидит, что оно истинное и войдет внутрь третьего блока операторов. После чего выполнение скрипта продолжится уже после всей этой конструкции.

<?php $a = 1;

|  |  |  |  |
| --- | --- | --- | --- |
| if ($a <  { | 2) |  |  |
| echo | "a | меньше | 2"; |
| } |  |  |  |
| elseif ($a | | < З) |  |
| { |  |  |  |
| echo | "a | меньше | 3"; |
| } |  |  |  |
| elseif ( | $a | < 4) |  |
| { |  |  |  |
| echo | "a | меньше | 4"; |

**else**

{

Чтобы лучше понять принцип работы, напишите следующий небольшой скрипт:

}

echo "a больше либо равно 4";

}

?>

Посмотрите результат выполнения, меняя значение $a.

Блоки кода операторов elseif и else выполняются только тогда, когда все проверенные ранее условия оказались равными false.

* 1. Тернарный оператор условия

Тернарный - это значит имеющий три операнда. Выглядит он следующим образом:

|  |  |
| --- | --- |
| Переменная = Условие ? Значение1 | : Значение2; |
| Если значение Условия истинно, Значение2. Пример: | то переменной присваивается Значение1, иначе |
| $а = (1 < 2) ? "1 меньше 2" : "1 | не меньше 2"; |

Убедитесь, что переменной $а в данном случае присваивается строка “1 меньше 2”, так как условие (1 < 2) истинно. Теперь попробуйте поменять знак с «<» на «>» и проверьте результат.

Тернарный оператор появился впервые в языке Си. Он позволяет включать в выражение проверку условия. Это красивая возможность, делающая код лаконичнее. Пользуйтесь тернарным оператором!

* 1. Оператор switch

Оператор switch вычисляет значение заданного выражения и сравнивает его с предложенными вариантами. В случае совпадения выполняется соответствующий блок кода.

Switch имеет следующий синтаксис:

switch (Выражение)

{

case Значение 1:

Блок операторов 1; break;

case Значение2:

Блок операторов2; break; default:

Блок операторов3;

}

Сначала switch вычисляет значение Выражения, после чего сравнивает со значение каждой метки case (Значение 1, Значение2). Как только было обнаружено совпадение, выполняется соответствующий блок операторов.

Обратите внимание на ключевое слово break в конце каждого блока case. break говорит о том, что выполнение операторов в блоке switch окончено, и нужно переходить к выполнению операторов после всего блока switch (после фигурной скобки). В случае, когда в конце блока case нет оператора break, интерпретатор продолжит выполнять операторы следующих блоков.

Если же, пройдясь по всем меткам case, так и не встретилось совпадения, выполняются операторы из блока default. Впрочем, это не обязательный блок и он может отсутствовать.

Пример:

<?php $a = 1;

**switch** ($a)

{

**case** 1:

|  |  |  |
| --- | --- | --- |
| echo " | | a равно 1"; |
| break; | |  |
| case 2: |  |  |
| echo " | | a равно 2"; |

**case** 3:

echo "a равно либо 2, либо 3. Не могу определить точнее, " .

"потому что забыли поставить break в предыдущем case";

**break**; **default**:

echo "a не равно ни 1, ни 2, ни 3";

}

?>

Нужно отметить, что иногда пропуск оператора break может использоваться намеренно для обеспечения правильной логики программы.

Особенностью языка PHP является то, что метками case могут быть переменные простых типов (не массивы и не объекты). Например, следующий скрипт верен с точки зрения синтаксиса языка:

<?php $a = 1;

$b = 2;

**switch**($a)

{

**case** 1:

|  |  |  |  |
| --- | --- | --- | --- |
| echo | " | a равно 1"; |  |
| break; | |  | |
| case $b: |  |  | |
| echo | " | a равно 2"; |  |
| break; | |  | |
| default: |  |  | |
| echo | " | a не равно | ни 1, ни 2" ; |

?>

1. Функции

}

Функция - это блок кода, к которому можно обращаться из разных частей скрипта. Функции могут иметь входные и выходные параметры. Входные параметры могут использоваться в операциях, которые содержит функция. Выходные параметры устанавливаются функцией, их значения используются после выполнения функции. Программист сам может определять необходимые ему функции и их логику выполнению.

* 1. Определение функции

Функция объявляется с помощью ключевого слова function. Если функция имеет параметры, они объявляются как переменные в объявлении функции:

function my\_function($my\_parameter)

{

echo "you send '$my\_parameter' to my function<br/>";

Вот и определение нашей первой функции. После ключевого слова function идет название функции. Мы назвали её my\_function. После этого в круглых скобках перечисляются параметры, которые мы хотим передать в функцию. В нашем случае мы описали параметр $my\_parameter. Теперь внутри тела функции, которое заключается в фигурные скобки «{» и «}», мы можем использовать переменную $my\_parameter. В частности, мы выводим её значение внутри уже известного нам оператора echo.

Обратиться к нашей функции теперь можно следующим образом:

my\_function("это мой параметр");

Только что мы вызвали нашу функцию, передав туда строку «это мой параметр». Давайте теперь взглянем на скрипт, в котором используется только что написанная функция:

<?php

function my\_function($my\_parameter)

{

echo "you send '$my\_parameter' to my function<br>";

}

my\_function('^TO мой параметр"); my\_function("a это мой новый параметр");

?>

Перепишите и выполните этот скрипт. Как вы можете увидеть, наша функция отработала два раза, при этом на экране появились две строчки:

you send 'это мой параметр' to my function

echo "you send '$my\_parameter1' and " .

'$my\_parameter2' to my function<br>";

you send 'а это мой новый параметр' to my function

Если требуется передать несколько параметров, то они разделяются запятой:

function my\_function($my\_parameter1, $my\_parameter2) {

}

Функция всегда возвращает значение. Если это не указать явно, то будет возвращено NULL. Оператор return позволяет завершить выполнение функции, вернув конкретное значение.

<?php

function my\_function\_sum($a, $b)

{

return $a+$b;

}

echo "если сложить 2 и 3, то получится "; echo my\_function\_sum(2, 3);

?>

Можно сохранять полученное из функции значение в переменной: $sum = my\_function\_sum(2, 3);

Return может ничего и не возвращать, а просто использоваться для выхода из функции: <?php

function not\_1($a)

{

if($a == 1)

return;

echo "очевидно, ваш параметр не равен 1";

}

not\_1(1); not\_1(2); not\_1(3);

?>

Пустой return - это тоже самое, что return NULL;

В данном случае строчка сообщения выведется только два раза, потому что в первом случае (not\_1 (1) ;) выполнится условие оператора if , отработат оператор return и произойдет выход из функции.

* 1. Значения по умолчанию

PHP позволяет определять значения по умолчанию для параметров функций. Единственное условие - эти параметры должны идти последними в объявлении функции. Присвоение значения по умолчанию делает параметр необязательным, то есть при вызове функции его можно опустить:

<?php

function sum($a, $b = 1)

{

return $a + $b;

}

echo sum(1); echo sum(1, 2);

?>

Сначала функция echo выведет значение 2, а потом 3. В первом случае мы вызвали функцию только с одним параметром, поэтому переменная $b (второй параметр) получила значение по умолчанию. Во втором случае, как и положено, переменной $b присваивается значение 2.

* 1. Рекурсия

Внутри блока кода функций можно пользоваться абсолютно любыми средствами языка PHP также как и вне функций. В том числе существует возможность вызова функции самой себя. Эта операция и называется рекурсией. Рекурсию часто позволяет найти элегантное решение задачи. Разберем самый распространенный пример использования рекурсии - вычисление факториала: function fact($x)

{

if ($x == 1)

return 1;

else

return $x \* fact($x - 1);

}

Факториал n - это произведение всех натуральных числе до n включительно. В математике факториал обозначается символом «!». То есть, к примеру, 5! = 1\*2\*3\*4\*5.

Как работает наша рекурсивная функция факториала? Сначала осуществляется проверка переданного параметра. В случае, если аргумент равен единице, мы возвращаем единицу (1! = 1). Если же наш параметр не равен единице, мы воспользуемся правилом:

n! = n\*(n-1) !

В терминах PHP это выглядит так:

$х \* fact($x - 1) ;

Когда программа доберется до этой строчки, произойдет новый вызов функции fact но уже с параметром на единицу меньше.

Рекурсией нужно пользоваться осторожно. Попробуйте, к примеру, выполнить в скрипте fact(-1). Как видите, функция никогда не попадет на строчку

return 1;

в случае, если передается отрицательное значение. Поэтому следует внимательно проектировать функции с рекурсией и предусматривать обработку некорректных значений аргументов.

1. Область видимости и время жизни переменных

Переменные по области видимости можно разделить на глобальные и локальные. К первым можно обращаться из любого места программы, вторые доступны только в конкретном ее месте.

Все переменные, которые объявлены в теле скрипта вне функций - глобальные. Переменные, объявленные в функциях, имеют локальную область видимости.

Рассмотрим следующий пример: <?php

function printA()

{

$a = 2; echo $a;

}

$a = 1;

echo $a; // выводит 1

printA(); // выводит 2

echo $a; // выводит 1

?>

Переменная $a, объявленная в нутрии функции printA не имеет никакого отношения к глобальной $a, объявленной сразу после функции. Поэтому скрипт выводит последовательность «121».

Локальные переменные сохраняют свое значение только во время выполнения функции. При новом вызове функции значения локальных переменных инициализируются заново: <?php

function printA() {

$a = 0; $a++; echo $a;

}

printA(); printA(); printA();

?>

Этот скрипт выведет три единицы, потому что при каждом обращении к функции printA значение переменной $а сначала устанавливается равным нулю.

Однако есть исключение. Чтобы функция все же «помнила» значения уже инициализированных переменных используется ключевое слово static. Таким образом, можно объявить статические переменные, которые инициализируются функцией только один раз:

<?php

function printA()

{

static $a = 0;

$a++; echo $a;

}

printA(); printA(); printA();

?>

Этот скрипт выведет последовательность «123», потому что обнуление $а происходит только при первом вызове функции. При последующих вызовах функции PHP будет знать, что переменная $a уже существует и пропустит строчку со словом static.

Все глобальные переменные доступны через системную переменную $\_GLOBALS. Она представляет собой ассоциативный массив. О том, что это такое мы как раз расскажем в следующем Практическая работа.

Резюме

Использование функций значительно упрощает жизнь программистам. С их помощью код можно хорошо структурировать, избавиться от повторения блоков кода в скрипте, избежать появления ненужных глобальных переменных. Со временем у программиста появляются библиотеки собственных функций, которые он может переносить из одного проекта в другой.

Старайтесь минимизировать область видимости переменных. Это сделает вашу программу гибче для внесения изменений и легче в разборе. По возможности избегайте глобальных переменных, передавая каждой функции только ту информацию, которая ей нужна в качестве параметров.

Стоит помнить, что большинство велосипедов уже изобретено. Поэтому перед тем как писать функцию вычисления синуса, нужно поинтересоваться, не решена ли уже ваша задача стандартными средствами PHP, или же какой-нибудь специальной библиотекой, которых в Интернете достаточно.

Задания:

1. Объявите в начале скрипта две целочисленных переменных $а и $b, начальные значения определите с помощью констант. Написать скрипт:
2. если $а и $b положительные - выведите их сумму.
3. если $а и $b отрицательные - выведите из разность.
4. если $а и $b разных знаков - выведите их произведение.

Ноль можно считать положительным числом.

1. Выведите большее из чисел используя тернарный оператор.
2. Присвойте $а значение в промежутке [0..9]. С помощью оператора switch организуйте вывод чисел от $a до 9;
3. Реализуйте все арифметические операции в виде функций с двумя параметрами.
4. Реализуйте функцию с тремя параметрами: function mathOperation($arg1, $arg2, $operation), где $arg1, $arg2 - значения аргументов. $operation - строка с названием операции. В зависимости от переданного значения операции выполните одну из арифметических операций (используйте функции из пункта 4) и верните полученное значение (используйте switch).
5. С помощью рекурсии организуйте функцию возведения числа в степень. Формат: function power($val, $pow), где $val - заданное число, $pow - степень.
6. Написать функцию, которая принимают в качестве аргументов два числа и вычисляет из них большее. Написать такую же функцию, чтобы она вычисляла меньшее число.

Проверить, если произведение $a и $b больше 100, но меньше 1000, то от большего числа отнять меньшее и вывести результат на экран. А если произведение этих чисел больше 1000, то произведение этих чисел разделить на большее из чисел.