Урок 2

Условные блоки, ветвление функции в PHP

Изучение принципов ветвления логики, их представления в виде схем и реализации в языке PHP. Знакомство с понятием функции. Реализация рекурсивных функций

Принципы ветвления, визуализация, блок-схемы

Операторы if, if-else

Оператор switch

Тернарный оператор

Функции

Области видимости переменных

Стандартные функции PHP

Практическое задание

Используемая литература

Принципы ветвления, визуализация, блок-схемы

В программном коде, как и в жизни, множество решений зависит от внешних факторов. И эта зависимость выражается в вербальном виде «Если случится событие А, то я выполню действие Б». Именно по такому принципу начинает строиться ветвление во всех языках программирования.

В русском языке для ветвления используется слово «если», в программировании применяются специальные операторы, обеспечивающие выполнение определённой команды или набора команд при условии истинности логического выражения или группы выражений. Ветвление – одна из трёх базовых конструкций структурного программирования (наряду с последовательным выполнением команд и циклом).

Для справки: в дискретной математике, которая является одной из фундаментальных наук, лежащих в основах программирования, условие ветвления есть предикат. Почитать об этом можно в дополнительной литературе.

Прежде чем приступать к написанию ветвлений на языке PHP, стоит поговорить о случаях, когда на ветвление влияет множество факторов. В этой связи следует визуализировать для себя логику программы или её части, чтобы не запутаться при реализации. Для решения задачи визуализации применяются так называемые блок-схемы.

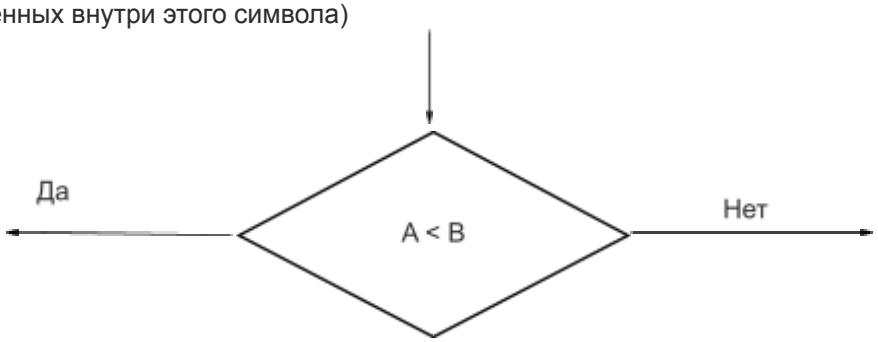
Блок-схема – распространенный тип схем, описывающих алгоритмы или процессы, в которых отдельные шаги изображаются в виде блоков различной формы, соединенных между собой линиями, указывающими направление последовательности. Сама блок-схема состоит из стандартных элементов:

Процесс (функция обработки данных любого вида)

Данные

Предопределённый процесс (символ отображает предопределённый процесс, состоящий из одной или нескольких операций или шагов программы, которые определены в другом месте)

Решение (то, о чём мы говорили в самом начале, – ситуация, имеющая одну точку входа и ряд альтернативных выходов, один и только один из которых может быть использован после вычисления условий, определенных внутри этого символа)



Терминатор (начало или конец программы)

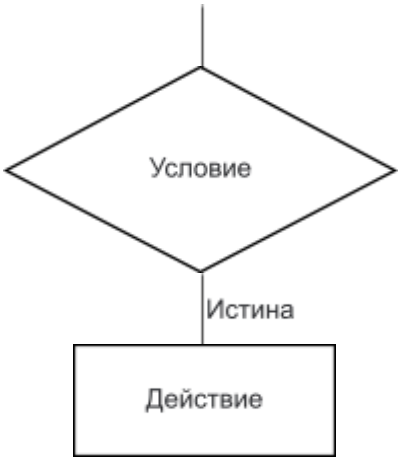
Для наших целей на текущий момент перечисленных блоков вполне достаточно. Более подробный материал по блок-схемам можно найти, перейдя по прилагающейся к занятию ссылке.

Итак, для изучения ветвлений нам потребуется элемент «Решение».

Операторы if, if-else

Для реализации ветвления в PHP используется оператор if.

```
<?php
if( Условие ) {
  Действие;
}
?>
```

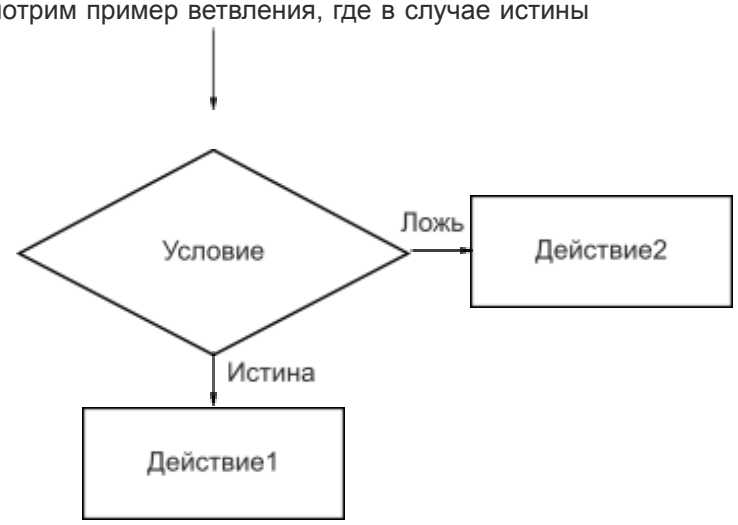


Условие – это любое выражение, возвращающее булевское значение (true, false), т.е. такой вопрос, на который ответить можно только двумя способами: либо да, либо нет. Действие выполняется тогда, когда условие истинно (true). Обычно условием является операция сравнения, либо несколько таких операций, объединённых логическими связками (И, ИЛИ). В результате проверки какого-либо условия может выполняться сразу несколько операторов:

```php
<?php
if( Условие ) {
  Действие1;
  Действие2;
}
?>
```

Что делать, если одного условия недостаточно? Рассмотрим пример ветвления, где в случае истины мы выполним одно действие, а в случае лжи – другое:

```php
<?php
if( Условие ) {
  Действие1;
}
else{
  Действие2;
}
?>
```



Попробуем реализовать простой пример:

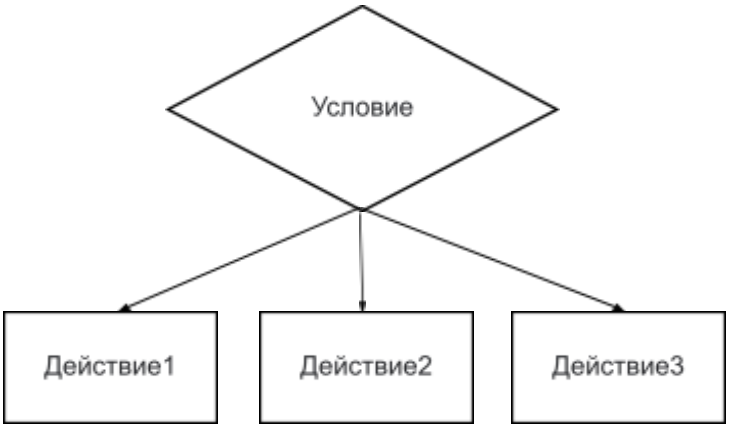
```php
<?php
$x = 5;
$y = 42;
if( $x > $y )
  echo $x + $y;
else
  echo $x * $y;
?>
```

Обратите внимание, если по условию нужно выполнять всего один оператор, фигурные скобки можно не ставить.

Логику ветвления не всегда получается уложить в две ветки, однако PHP позволяет разделять нашу программу на сколько угодно вариантов с помощью конструкции else if, позволяющей анализировать дополнительное условие. При этом выполняться будет первое условие, вернувшее true.

Представим следующую задачу: даны два произвольных числа и их соотношение друг с другом необходимо вывести на экран. По сути, у нас будет три варианта: либо первое число больше, либо второе, либо они равны.

```php
<?php
$x = 5;
$y = 42;
if( $x > $y )
  echo "$x больше $y";
else if ( $x < $y )
  echo "$x меньше $y";
else
  echo "$x равен $y";
?>
```



Оператор switch

Теперь представим ситуацию, когда нужно разделить программу не на 2 или 3 варианта, а на большее количество. Конечно, можно много раз использовать конструкцию else if, но это способно привести к серьёзному ухудшению читаемости кода. Поэтому существует специальный оператор выбора из нескольких вариантов – switch. Он имеет следующий синтаксис:

```php
<?php
switch(переменная){
  case Значение1:
      Действие1;
      break;
  case Значение2:
      Действие2;
      break;
  default:
      Действие3;
}
?>
```

Оператор switch смотрит на значение переменной (вместо неё также может стоять выражение, возвращающее значение) и сравнивает его с предложенными вариантами. В случае совпадения выполняется соответствующий блок кода. Если же после прохода по всем вариантам совпадения так и не обнаружилось, выполняются операторы из блока default. Это необязательный блок, который может отсутствовать.

Обратите внимание на ключевое слово break в конце каждого блока case. Оно ставится в 99% случаев и означает, что нужно прекратить выполнение операций внутри switch. В случае, когда в конце блока case нет оператора break, интерпретатор продолжит выполнять действия из следующих блоков.

```php
<?php
$now = 'evening';
switch ($now){
  case 'night':
  echo 'Доброй ночи!';
  break;
case 'morning':
  echo 'Доброе утро!';
  break;
case 'evening':
  echo 'Добрый вечер!';
  break;
default:
  echo 'Добрый день!';
  break;
}
?>
```

```php
<?php
$now = 'evening';
if ($now == 'night'){
  echo 'Доброй ночи!';
}
else if ($now == 'morning'){
  echo 'Доброе утро!';
}
else if ($now == 'evening'){
  echo 'Добрый вечер!';
}
else{
  echo 'Добрый день!';
}
?>
```

Тернарный оператор

Тернарный оператор – это операция, возвращающая либо второй, либо третий операнд в зависимости от условия (первого операнда). Звучит страшно, но выглядит достаточно просто:

```php
<?php
(Условие) ? (Операнд по истине) : (Операнд по лжи);
?>
```

Например, мы хотим сохранить максимальное из двух произвольных чисел в какую-то переменную. В этом случае вместо громоздких строк ветвления можно написать:

```php
<?php
$x = 10;
$y = 15;
$max = ($x > $y) ? $x : $y;
?>
```

Тернарный оператор – красивая возможность, делающая код лаконичнее. Однако злоупотреблять этой возможностью, усложняя код, не стоит, как и любым другим инструментом.

Функции

Представим себе, что, используя код одного из примеров, мы хотим построить с пользователем диалог. Код в любой программе работает последовательно, строка за строкой. Таким образом, условие уже отработано, вернуться к нему невозможно. Как решить эту задачу? Мы можем

скопировать весь блок операций ещё несколько раз. Но что, если количество раз неизвестно заранее? Да и copy-paste – это уж совсем плохое решение. Тогда на помощь приходят 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, а их значения используются после её выполнения. Программист может создавать необходимые ему функции и логику их выполнения.

Если проводить аналогию с реальной жизнью, то функция – это некий навык скрипта, который он знает и умеет делать. Вы же не учитесь ходить каждый раз, когда перемещаетесь между точками? Вы просто выполняете функцию «Ходить». Также и скрипт может иметь описанную функцию go, способную вызываться в любой момент времени.

Функция в PHP объявляется с помощью ключевого слова function. За ним следует название функции, которое мы придумываем сами. Затем в круглых скобках через запятую указываются параметры, которые данная функция принимает. По сути, параметры – это входные данные для функции, над которыми она будет выполнять какую-то работу. После указания параметров в фигурных скобках следует тело функции. Обратите внимание, что пробела между скобками и названием функции, в отличие от if и switch, быть не должно. После объявления функции, мы можем её вызвать и посмотреть, как она работает. Описание функции может находиться и до, и после её вызова.

```
<?php
function имя_функции(параметр1, параметр2, …){
  Действия
}
?>
```

Давайте создадим функцию, которая будет сравнивать числа:

```
<?php
function compare_numbers($x, $y){
  if ($x > $y)
    echo "$x > $y";
  else if ($x < $y)
    echo "$x < $y";
  else
    echo "$x = $y";
}
?>
```

Имея такую функцию в арсенале, можно сравнивать сколько угодно пар чисел, не задумываясь о процессе сравнения.

```
<?php
function compare_numbers($x, $y){
  if ($x > $y)
    echo "$x > $y";
  else if ($x < $y)
```

```
    echo "$x < $y";
  else
    echo "$x = $y";
}
compare_numbers(10, 20);
compare_numbers(20, 10);
compare_numbers(20, 20);
?>
```

При вызове функции необходимо обязательно передавать в неё количество параметров, заявленное при её создании. Их может быть 0 и более. Если параметры не переданы, при вызове функции нужно просто указать пустые скобки:

```
<?php
function myFunction(){
// делаем что-то
}
myFunction();
?>
```

Функция всегда возвращает значение. Если это не указать явно, то будет возвращено значение типа NULL. Оператор return позволяет завершить выполнение функции, вернув конкретное значение. Если в функции не указано, что она возвращает, то, по сути, результатом её работы может являться только вывод какого-то текста на экран (см. предыдущую функцию). Однако в большинстве случаев мы хотим использовать результат работы функции в остальной программе. Тогда необходимо использовать оператор return. Например, напишем функцию, возвращающую среднее арифметическое двух чисел:

```
<?php
function average($x, $y)
{
  return ($x + $y)/2;
}
$avg = average(42, 100500);
echo $avg;
?>
```

Таким образом, мы не только учим наш скрипт определённым навыкам, но и можем хранить результат выполнения каждой функции для дальнейшего использования.

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

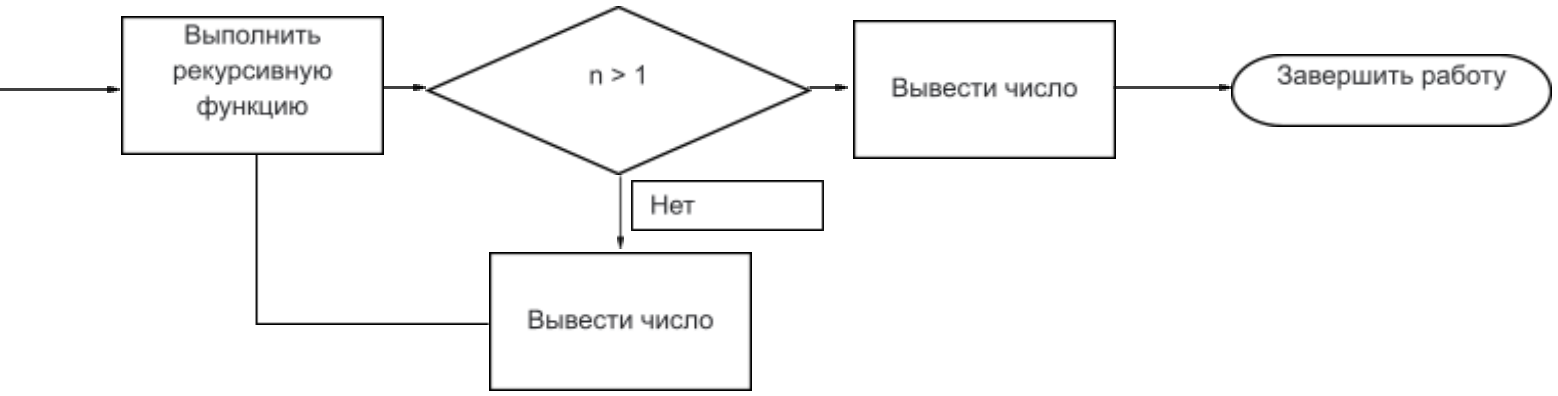
```
<?php
function mult($a, $b = 1){
  return $a * $b;
}
```

```
echo mult(8);
echo mult(8, 2);
?>
```

Одним из наиболее интересных моментов в вопросе использования функций является рекурсия. Рекурсия – это вызов функцией самой себя:

```
<?php
function recusiveFunction(){
  …
  return recusiveFunction ();
}
?>
```

Когда это может быть полезно? Не стоит сейчас углубляться в решение задач обхода деревьев, но гораздо проще привести пример с вычислением последовательности n чисел Фибоначчи (каждое последующее число равно сумме двух предыдущих чисел). Каждый раз мы не знаем, сколько чисел Фибоначчи запросит пользователь, но, используя рекурсию, мы можем не думать об этом. Схема будет следующей:




```
<?php
function fibonacci($n, $prev1 = 1, $prev2 = 0){
        $current = $prev1 + $prev2;
        echo "$current ";
        if($n > 1)
                fibonacci($n - 1, $current, $prev1);
}
fibonacci(15);
?>
```

Области видимости переменных

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

Все переменные, которые объявлены в теле скрипта вне функций, – глобальные. Переменные, объявленные в функциях, имеют локальную область видимости. Переменные, которые функция принимает в качестве параметров, также являются локальными.

```
<?php
function changeX($x){
   $x += 5;
   echo $x;
}
$x = 1;
echo $x;         // выводит 1
changeX($x); // выводит 6
echo $x;        // выводит 1
?>
```

Переменная $x, объявленная внутри функции changeX, не имеет никакого отношения к глобальной $x, объявленной сразу после функции. Локальные переменные сохраняют своё значение только во время выполнения функции, а после её завершения стираются из памяти компьютера. Команда $x += 5 меняла значение локальной переменной, так как отрабатывала внутри функции. Глобальная переменная при этом оставалась неизменной. Поэтому скрипт выводит последовательность «161».

Стандартные функции PHP

В стандартной установке интерпретатора PHP уже включено несколько сотен различных функций для выполнения самого разного спектра задач: работа со строками, числами, сетью, файлами и так далее.

```
<?php
$a = 'test';
var_dump($a);
?>
```

В данном примере: var_dump – это название функции. Через запятую в скобках передаются аргументы.

Ещё один пример:

```
<?php
$name = 'Alex';
$string = 'Hello, ' . $name;
$otherString = str_replace('Hello', 'Goodbye', $string);
```

```
echo $otherString;
?>
```

Стандартная функция str_replace заменяет подстроку, переданную в первом аргументе, на значение, переданное во втором аргументе, в строке, переданной в третьем аргументе, и возвращает данное значение. Это означает, что результат работы этой функции будет присвоен переменной $otherString.

Практическое задание

1. Объявить две целочисленные переменные $a и $b и задать им произвольные начальные значения. Затем написать скрипт, который работает по следующему принципу:
    a. Если $a и $b положительные, вывести их разность.
    b. Если $a и $b отрицательные, вывести их произведение.
    c. Если $a и $b разных знаков, вывести их сумму.

    Ноль можно считать положительным числом.
2. Присвоить переменной $a значение в промежутке [0..15]. С помощью оператора switch организовать вывод чисел от $a до 15.
3. Реализовать основные 4 арифметические операции в виде функций с двумя параметрами. Обязательно использовать оператор return.
4. Реализовать функцию с тремя параметрами: function mathOperation($arg1, $arg2, $operation), где $arg1, $arg2 – значения аргументов, $operation – строка с названием операции. В зависимости от переданного значения операции выполнить одну из арифметических операций (использовать функции из пункта 3) и вернуть полученное значение (использовать switch).
5. Посмотреть на встроенные функции PHP. Используя имеющийся HTML шаблон, вывести текущий год в подвале при помощи встроенных функций PHP.
6. *С помощью рекурсии организовать функцию возведения числа в степень. Формат: function power($val, $pow), где $val – заданное число, $pow – степень.
7. *Написать функцию, которая вычисляет текущее время и возвращает его в формате с правильными склонениями, например: 22 часа 15 минут, 21 час 43 минуты.

Дополнительные материалы

1. https://ru.wikipedia.org/wiki/Рекурсия
2. http://php.net/manual/ru/language.variables.scope.php
3. http://php.net/manual/ru/language.variables.variable.php
4. http://php.net/manual/ru/ref.strings.php
5. http://php.net/manual/en/function.date.php

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Котеров Д.: PHP 5 в подлиннике.
2. Head First PHP and MySQL.