Лекция 5

Функции, определяемые пользователем

Первоначально функции в программировании имели тот же, или почти тот же смысл, что и в математике. Существует даже целый класс языков функционального программирования (Lisp, Haskell,...), основная идея которых состоит в применении "чистых" функций, понимаемых точно так же как и в математике. Это объясняется тем, что ЭВМ на ранних стадиях развития (конец 40-х, начало 50-годов XX века) использовались в основном для решения научных задач с большим количеством математических расчётов, т.е. расчётов по формулам. Один из первых языков программирования так и назывался ForTran - транслятор формул.

Пример записи формулы в математике и в программе на языке Fortran

В математике

$$y = \frac{\sin(\ln(|x|) + \cos(e^2))}{tg(2x)};$$

на языке Fortran

```
y = (sin(log(abs(x))) + cos(exp(x)))/tan(2*x)
```

В обоих случаях все функции, используемые в формуле, вычисляются при заданных значениях аргументов, и полученные значения подставляются в формулу. Программисты говорят, что функция возвращает значение. В формуле указываются имя и аргументы (фактические параметры) функции. Сама функция - это обособленная часть программы. Чтобы функцию можно было вызывать с разными переменными или выражениями, в ней принимаемое значение аргумента присваивается переменной, видимой только в этой функции. Такие переменные называются локальными, или переменными с локальной областью видимости. Придумано огромное количество способов выделения областей видимости. Использование тех или иных видов областей видимости и различных структур данных в основном определяют различия языков программирования и усилия, затрачиваемые на их изучение.

Кроме функций в Fortran-е используются подпрограммы, отличающиеся от функций, только тем, что подпрограмма не возвращает значения. Со временем стало ясным, что различия между функцией и подпрограммой несущественны и их в новых языках программирования перестали различать. В Perl-е используется термин *подпрограмма*, в C, JavaScript и PHP - *функция*.

В программах часто используются *глобальные переменные*, которые видны во всей программе. Глобальная переменная, используемая в функции, может изменять своё значение вне функции, поэтому даже при одинаковых значениях формальных параметров функция может возвращать разные значения.

Создание пользовательской функции в PHP

В C нужно объявить (указать транслятору) прототип функции, отличающийся от функции отсутствием тела. Описание функции - суть сама функция.

В PHP указывать транслятору прототип функции не нужно. К сожалению, описание функции в PHP называется объявлением. Оно имеет следующий синтаксис:

```
function имя_функции(список формальных параметров)
{
   тело функции
}
```

Имя функции должно состоять из латинских букв, цифр и знака подчёркивания и начинаться с буквы или знака подчёркивания.

Список формальных параметров - это последовательность имён переменных, разделённых запятыми. Он может быть пустым.

Тело функции - последовательность операторов.

Пример. Вычислить n! двумя способами.

```
$N1 = _fact(5);
$k = 5;
$N2 = factRec($k);
  //Вычисление факториала с помощью цикла
function _fact($n)
{   $k=1;
    for($i = 2;$i <= $n; $i++) $k *= $i;
    return $k;
}
//Вычисление факториала с помощью рекурсивной функции
function factRec($n)
{   if($n < 2)return 1;
    else return $n*factRec($n-1);
}
```

Оператор "*return выражение*" служит для выхода из функции и возврата значения выражения. Оператор return возвращает значение, которое может иметь тип *скаляр, массив* или *объект*.

Пример возврата нескольких скаляров в виде массива

```
list ($Fam, $Imja, $vozrast) = spisok();
print "$Fam $Imja, возраст - $vozrast года<BR>";
//Напечатается: Петрова Маша, возраст - 4 года
function spisok()
{   $v = 4;
    return array ('Петрова', 'Маша',  $v );
}
```

Функция array(список) превращает список в массив, а языковая конструкция "*list(список) = массив*" разлагает массив на скалярные переменные.

Вызов функции с переменным числом параметров

Количество фактических параметров при вызове функции может быть меньше или равно количеству формальных параметров. Отбрасывать можно параметры только с конца списка фактических параметров. Если в списке фактических параметров параметр отброшен, то соответствующему формальному параметру должно быть присвоено начальное значение. Передаваемые значения фактических параметров имеют более высокий приоритет перед начальными значениями формальных параметров.

Пример вызова функции с переменных количеством параметров

```
<?php
print SumABC(1).'<BR>';      // 91  1+40+50
print SumABC(1,2).'<BR>';    // 53  1+2+50
print sumAbc(1,2,3).'<BR>'; // 6   1+2+3

function SumABC($x,$y=40,$z=50)
{return $x+$y+$z;
}
?>
```

Передача массивов в функции

Массивы и скалярные переменные могут передаваться в функцию в любом порядке, только список фактических и список формальных параметров должны строго соответствовать друг другу.

Пример

```
$x = 5;
$A1 = array(1,2,3,4);
$A2= array(
          array(8,9,10),
          array(4,5,6,'семь'),
          array('a','b','c'),
         );
f_Mas($x,$A1,'y',$A2,15);
function f_Mas($a,$Ar1,$b,$Ar2,$c)
{ print "a=$a, Ar1[1]=$Ar1[1], b=$b,
Ar2[1][1]=".$Ar2[1][1].", c=$c<BR>";
}
//Напечатается
//a=5, Ar1[1]=2, b=y, Ar2[1][1]=5, c=15
```

Примечание. Элементы двумерного массива не интерполируются, поэтому в примере использована операция конкатенации.

Статические переменные

Статические переменные видны только в теле функции и сохраняют своё значение при выходе из функции. Объявляются статические переменные с помощью оператора

```
static имя_переменной = значение;
```

Оператор static выполняется только при первом вызове функции.

Пример

```
for($i=1; $i<4; $i++) f_static();
function f_static()
{ static $a = 0;
  $a++;
  print "$a<BR>";
}
/* Напечатается
  1
  2
  3
*/
```

Локальные, глобальные и суперглобальные переменные

Данный раздел демонстрирует, как резко усложняется язык программирования при организации взаимодействия разных областей видимости.

Пример А

```
1   $a=2;       //$a - глобальная переменная
2   $b = f($a);
3   print "a=$a, b=$b<BR>";//a=3, b=8
    // print "***** Суперглобальный массив ****<BR>";
    //foreach($GLOBALS as $kluch => $znach) print
"$kluch - $znach<BR>";
4   function f($c) //$c=2 - формальный параметр с
локальной
    {           //областью видимости
5      $a = 5;  //локальная переменная
6      $c++;
7      print "a=$a<BR>";//напечатается 5
8      $d = $a + $c; //d = 5 + 3 =8
9      global $a; // с этой точки в функции видна
глобальная
                  // переменная $a, заслонившая
локальную $a
10     $a++; //$a=3
11     return $d;
    }
```

В примере главная программа состоит из строк 1-3. Это область видимости глобальных переменных $a и $b. Строки 4-11 - область видимости локальных переменных $c и $d. Локальная переменная $a видна в строках 5-8 до объявления *global $a*, после которого видна глобальная переменная с тем же именем "$a".

Распространено мнение, что применение глобальных переменных в локальных областях видимости опасно. Из моей практики следует обратное. Использование в функциях с помощью объявления *global* переменных, которые по своему смыслу являются глобальными для всей программы, или большей её части, сильно упрощает программу по сравнению с вариантом передачи в функции всех необходимых переменных через механизм фактических и формальных параметров.

Пример функции с 22-мя глобальными переменными

```
#******** Вызов функции      *******
stat_viv();
. . . . . .
. . . . . .
#******** Печать статистики  *******
function stat_viv()
{global
$Z,$n,$plan,$bezEx,$lgot,$vsFakt,$naObOsn,$naOb,$spec
Nab,$gosObZak,
 $nc,$ng,$prin_g,$net_g,$prin_c,$net_c,
 $budg,$kontr,$bezExBK,$lgotBK,$vneOl,$vneOlBK;
    if(!$prin_c)$prin_c=0;
  . . . . . .
  . . . . . .
}
```

Синтаксис объявления *global*

```
   global список_переменных
Например
   global $x, $y, $z;
```

Список всех глобальных имён программы хранится в суперглобальном ассоциативном массиве *$GLOBALS*. Если выполнить два закомментированных оператора между строками 3 и 4 в примере А в начале этого раздела, то выведется содержимое массива *$GLOBALS* для этого примера:

```
***** Суперглобальный массив ****
GLOBALS - Array
_POST - Array
_GET - Array
_COOKIE - Array
_FILES - Array
a - 3
b - 8
```

```
znach - 8
kluch - znach
```

Суперглобальные переменные

Суперглобальные переменные - это встроенные переменные (ассоциативные массивы), которые всегда доступны во всех областях видимости.

- GLOBALS - Ассоциативный массив (array), содержащий ссылки на все переменные, определенные в данный момент в глобальной области видимости скрипта. Имена переменных являются ключами массива.
- $_SERVER - массив переменных среды сервера.
- $_GET - массив переменных, передаваемых из браузера (от клиента).
- $_POST - массив переменных, передаваемых из браузера (от клиента)/
- $_COOKIE - массив cookkie-переменных.
- $_REQUEST - массив всех $_GET, $_POST и cookie-переменных.
- $_FILES - массив, содержащий информацию о загруженных файлах.
- $_SESSION - массив переменных сеанса.
- $_EVN - массив переменных окружения.