Лабораторная работа «Функции»

Цель: изучить способы определения функций, научиться составлять программы с использованием пользовательских функций.

Порядок выполнения работы:

1) записать в тетрадь формат определения пользовательской функции;
2) перечислить виды пользовательских функций в php;
3) понять и отладить примеры;
4) оформить отчёт с выводом по изучению темы;
5) выполнить домашние задания.

Функции, определяемые пользователем

Функция пользователя определяется следующим образом:

```
function Имя_функции ($arg1, $arg2, ... $argN)
{
  Операторы;
  return "возвращаемое значение";
}
```

Возвращаемое значение определяется любым корректным php-выражением (значение его может быть любого типа, в том числе массивом или объектом).

Требования к функциям

Имя функции и имена параметров функции должны соответствовать правилам наименования в PHP

Имена функций нечувствительны к регистру, но во избежание ошибок лучше вызывать функцию по тому же имени, каким она была задана в определении.

У функции может не быть параметров, а также и возвращаемого значения.

Для вызова функции указывается имя функции и в круглых скобках список значений ее параметров:

Имя_функции ("значение_для_параметра1", "значение_для_параметра2", ...);

Значениями параметров могут быть имена переменных или константы

Если функция однажды определена в программе, то переопределить или удалить ее позже нельзя.

Переменные внутри функции являются локальными, создаются при ее вызове и уничтожаются при завершении.

Передача аргументов

Передача аргументов

по значению, используется по умолчанию, изменение значения аргумента внутри функции не влияет на его значение вне функции

по ссылке, позволяет изменять аргументы с сохранением их изменений вне функции, для этого перед именем аргумента следует написать знак &

задание значений аргументов по умолчанию, значение по умолчанию должно быть константным выражением, а не переменной или вызовом другой функции, аргументы, для которых задаются значения по умолчанию, должны быть записаны после всех остальных аргументов в определении функции

Пример функции

Пример функции, которая вычисляет сумму аргументов с проверкой их типа

```
<?php
function sum($a, $b)
{
  if (is_numeric($a) && is_numeric($b))
    return $a+$b;
  else
    return 'Нечисловые данные';
}
$x=sum(3,5);
echo $x,'<br />';
$x=sum('38 попугаев', 5);
echo $x;
```

Функции с переменным числом аргументов

В PHP можно создавать функции с переменным числом аргументов, т.е. не зная заранее, со сколькими аргументами ее вызовут. Для такой функции можно просто при определении функции использовать многоточие … перед именем аргумента. В этом случае они передаются при вызове функции в виде массива.

function fn(…$a) { }

$b = fn(2, 5, 7) //$a получит массив [2, 5, 7]

Можно и объединять с обычными аргументами

function fn($b, …$a) { }

Многоточие (...) можно использовать также при вызове функции, чтобы распаковать массив (array) или Traversable переменную в список аргументов

function fn($a, $b, $c) { }

$b = fn(…[2, 5, 7]) //$a, $b, $c получат соответственно значения 2, 5 и 7

Функции с переменным числом аргументов

Более старый вариант (нерекомендуемый для использования) - не указывать аргументы в функции, а для доступа к аргументам тогда можно использовать функции

func_num_args(void) : **int** – возвращает число аргументов, переданных в текущую функцию.

func_get_arg(int $arg_num) : mixed – возвращает аргумент из списка переданных в функцию аргументов, порядковый номер которого задан параметром $arg_num. Аргументы функции считаются начиная с нуля.

func_get_args (void) : array – возвращает массив, состоящий из списка аргументов, переданных функции. Каждый элемент массива соответствует аргументу, переданному функции.

Функции с переменным числом аргументов

```php
<?php
function sum()//без аргументов
{
  $args = func_get_args();//массив переданных данных
  $s = 0;
  foreach ($args as $a)  {
    if (is_numeric($a))
      $s += $a;
    else
      return 'Нечисловые данные';
  }
  return $s;
}
$x=sum(3, 5, 7);
echo $x,'<br>';
$x=sum('38 попугаев', 5);
echo $x;
```

```php
<?php
function sum(...$args)//$args передается в виде массива
{
    $s = 0;
    foreach ($args as $a)  {
        if (is_numeric($a))
            $s += $a;
        else
            return 'Нечисловые данные';
    }
    return $s;
}
$x=sum(3, 5, 7);
echo $x,'<br>';
$x=sum('38 попугаев', 5);
echo $x;
```

Глобальные переменные в функции

Чтобы использовать внутри функции переменные, заданные вне нее, эти переменные нужно объявить как глобальные. Для этого в теле функции следует перечислить их имена после ключевого слова global, например:

```
<?php
  $a = 1;
  function My_func()
  {
    global $a;
    ...
  }
```

Статические переменные в функции

Чтобы использовать переменные только внутри функции, при этом сохраняя их значения и после выхода из функции, нужно объявить эти переменные как статические. Статические переменные видны только внутри функции и не теряют своего значения, если выполнение программы выходит за пределы функции. Объявление таких переменных производится с помощью ключевого слова static:

```
<?php
function My_func()
  {
    static $a = 0;
    $a++;
    return $a;
  }
  for ($i = 0; $i < 5; $i++) echo My_func();
```

Статической переменной может быть присвоено любое значение, но не ссылка.

Возвращение ссылки из функции

В результате своей работы функция также может возвращать ссылку на какую-либо переменную. Это может пригодиться, если требуется использовать функцию для того, чтобы определить, какой переменной должна быть присвоена ссылка. Чтобы получить из функции ссылку, нужно при объявлении перед ее именем написать знак & и каждый раз при вызове функции перед ее именем тоже писать амперсанд &, например:

```
<?php
function &ref()
 {
   $a = 5;
   $b = 3;
   if ($a>$b) return $a;
   else return $b;
 }
 $c = &ref();
 echo $c;  // Выводит 5
?>
```

При использовании ссылок в переменную $c примера не копируется значение переменной $a, возвращенной функцией $ref, а создается ссылка на эту переменную.

Переменные функции

PHP поддерживает переменные функции. Если имя переменной заканчивается круглыми скобками, то PHP ищет функцию с таким же именем и пытается ее выполнить. Пример:

```
<?php
 function A($i) { echo "a $i"; }
 function B($i) { echo "b $i"; }
 function C($i) { echo "c $i"; }
 $F="A";  // или $F="B" или $F="C"
 $F(10);  // вызов функции, имя которой хранится в $F
```

Генераторы

Генератор может возвращать последовательность значений (yield), которая перебирается в цикле foreach

```
function f($max)// 1 1 2 3 5 8 13 …
{
    $i = 0;
    $n = 1;
    yield $n;
    while (true) {
        $temp = $n;
        $n += $i;
        $i = $temp;
        if ($n >$max) return;
        yield $n;
    }
}

foreach (f(1000) as $num) {
    echo $num, '<br>';
}
```

Рекурсия

PHP допускает рекурсию, т.е. многократный вызов функцией самой себя, пока не будет выполнено некоторое условие.
Например, функция для вывода многомерного массива

```
<?php
function recursion($x)
{
  if (is_array($x)) {
    foreach ($x as $value)
    recursion($value);
  } else {
    echo $x,'<br />';
  }
  return;
}
$x=array(array('Сервер'=>'Apache',
        'Язык программирования'=>'PHP','СУБД'=>'MySQL'),
        array(1, 2),
        array(7, 3, 2));
recursion($x);
```

Анонимные функции

Анонимные функции, или замыкания (closures), позволяют создавать функции, не имеющие определенных имен. Они наиболее часто используются в качестве значений callback-параметров.

Замыкания также могут быть использованы в качестве значений переменных; PHP автоматически преобразует такие выражения в экземпляры внутреннего класса Closure.

Анонимные функции объявляются без имени. Они могут также наследовать переменные из родительской области видимости с объявлением в конструкции use.

$a = 5;

**$b = function($x) use ($a) { }** //$a доступна, можно по ссылке &$a

Анонимные функции

```
<?php
$array1 = [1, 2, 3, 4, 5]; $array2 = $array1; $array3 = $array1;
function sqr(&$x) {
    $x = $x * $x;
}
array_walk($array1, 'sqr');
var_dump($array1);
array_walk($array2, function(&$x) {$x = $x*$x;});
var_dump($array2);
$sqr = function (&$x) {
    $x = $x * $x;
};
array_walk($array3, $sqr);
var_dump($array3);
```

Стрелочные функции

Стрелочные функции появились в PHP 7.4, как более лаконичный синтаксис для анонимных функций. Они также реализованы с использованием класса Closure.

Основной вид записи стрелочных функций:

fn (argument_list) => expr

Стрелочные функции аналогичны анонимным функциям, за исключением того, что использование переменных из родительской области всегда выполняется автоматически по значению без указания use.

$a = 5;

$b = function($x) => $x + $a;//возвращает $x + 5

Функции, определяемые пользователем

Допускается объявление функций внутри других функций, однако, вложенное объявление ничем не отличается от обычного ее объявления. Вложенная функция не наследует параметров родительской функции, параметры должны передаваться ей точно так же, как и любой другой функции.

Можно использовать **ОБЪЯВЛЕНИЕ ТИПА** параметров функции, указывая его перед именем аргумента функции, а также тип возвращаемого значения через двоеточие

function myfunc(int $a, float $b): string

В случае несоответствия типов в PHP 5 это будет обрабатываемая фатальная ошибка, а в PHP 7 будет выбрасываться исключение TypeError

Объявление типа

Тип возвращаемого значения можно указывать с версии PHP 7.0 и с версии 7.1 **void** для функций, не возвращающих значений.

В случае несоответствия типов в PHP 5 это будет обрабатываемая фатальная ошибка, а в PHP 7 будет выбрасываться исключение TypeError

Начиная с PHP 7.1, объявления типов могут быть помечены как обнуляемые, путём добавления префикса в виде знака вопроса(?). Это означает, что значение может быть как объявленного типа, так и быть равным null, например **?int**.

С версии PHP 8.0 можно указывать объединённые типы, которые позволяют использовать несколько типов, а не только один. Например **int|string|null**

Типы данных аргумента

Тип данных	Версия PHP
Имя класса или интерфейса	5.0
self (объект собственного класса)	5.0
array	5.1
callable	5.4
bool	7.0
float	7.0
int	7.0
string	7.0
iterable (массив или объект с интерфейсом Traversable)	7.1
object (любой объект)	7.2
mixed (array|bool|callable|int|float |object|resource|string|null)	8.0

Тип возвращаемого значения можно указывать с версии PHP 7.0

Задание. Использовать пользовательские функции

Для текста, например

$text =

Laravel is a web application framework with expressive, elegant syntax. We believe development must be an enjoyable, creative experience to be truly fulfilling. Laravel attempts to take the pain out of development by easing common tasks used in the majority of web projects, such as authentication, routing, sessions, and caching.

Laravel aims to make the development process a pleasing one for the developer without sacrificing application functionality. Happy developers make the best code. To this end, we've attempted to combine the very best of what we have seen in other web frameworks, including frameworks implemented in other languages, such as Ruby on Rails, ASP.NET MVC, and Sinatra.

Laravel is accessible, yet powerful, providing powerful tools needed for large, robust applications. A superb inversion of control container, expressive migration system, and tightly integrated unit testing support give you the tools you need to build any application with which you are tasked.'

Задание

1. Определить, сколько раз встречается в тексте слово Laravel и вывести текст, выделив его в тексте цветом.
2. Вывести в браузер статистику файла - количество абзацев, предложений, слов, символов.
3. Найти самое длинное слово. Если таких несколько, вывести в браузер их все.
4. Для каждого символа, имеющегося в тексте подсчитать, сколько раз он там встречается, символы расположить в возрастающем порядке.

Успехов в учёбе