Урок №2

Операторы ветвления

Условный оператор

```
#выполнить фрагмент кода при соблюдении определенного условия

if (логическое_выражение) {
    команда1
} [ else if (логическое_выражение2) {
    команда2
} [ else {
    команда3
} ]]
```

Блок else не является обязательным, поэтому может быть упущен.

Условные структуры могут быть вложенными.

Если используется только одна команда, то фигурные скобки так же можно упустить.

Если нужно выполнить один из многих фрагментов кода:

elseif => else if

Задача №2.1

Создать переменную.

Присвоить переменной значение - любой год. Например, 2003.

С помощью if..else выяснить, является ли год високосным.

```php
<?php
    $year   2003

    if ($year   4) {
        echo 'Год не является високосным';
    } else {
        echo 'Год является високосным'
    }
?>
```

Тернарный оператор

```
(логическое_выражение) ? команда1 : команда2

(expr1) ? (expr2) : (expr3)
```

Если логическое_выражение истинно, то выполнить команда1,

иначе выполнить команда2.

```php
# Пример использования тернарного оператора
$action = empty($_GET['action']) ? 'default' : $_GET['action'];

# Аналогичен следующему блоку с использованием if/else
if (empty($_GET['action'])) {
    $action = 'default';
} else {
    $action = $_GET['action'];
}
```

Оператор выбора switch

Проверка одной переменной на соответствие разным значениям

```
switch (выражение) {
    [ case Константное_выражение1:
        Список_операторов1 ]
    [ case Константное_выражение2:
        Список_операторов2 ]
    [ case Константное_выражение3:
        Список_операторов3 ]
    …
    [ default :
        Список_операторов ]
}
```

Когда обнаруживается соответствие метке case, исполняются все последующие строки, пока не встретится инструкция break.

default создает блок, который выполняется по умолчанию.

Задача №2.2

Создать несколько констант с названием операций. Например, ADD_NEW_PRODUCT, EDIT_PRODUCT и т.д.

Создать переменную и присвоить ей значение одной из операций.

Используя оператор выбора switch вывести на экран выбранную операцию.

```php
<?php
    define('ADD_NEW_PRODUCT'  1)
    define('EDIT_PRODUCT', 2);
    define('DELETE_PRODUCT', 3);

    $cmd   ADD_NEW_PRODUCT

    switch ($cmd) {
        case ADD_NEW_PRODUCT:
            echo 'Добавить новый продукт';
            break;
        case EDIT_PRODUCT:
            echo 'Отредактировать продукт';
            break;
        case DELETE_PRODUCT:
            echo 'Удалить продукт';
            break;
    }
?>
```

Урок №2

Пользовательские функции

Определение функции

Функция – это блок программного кода, который определяется один раз и может вызываться многократно.

```
function имя_функции([аргумент1[, аргумент2[, …, аргументN]]]) {
    блок_кода_функции
}
```

К имени функции применяются те же ограничения, что и к именам других идентификаторов.

Аргументы функции – локальные переменные, значения которых определяются при вызове функции.

В теле программы функция вызывается по имени, после которого в круглых скобках указываются аргументы функции.

```
function square($x) {
   return $x * $x;
}
square(2);
```

Передача функциям аргументов

Информация передается функциям через набор их аргументов, представленных последовательностью значений, разделенных запятыми.

```
function sum($x, $y) {
   return $x + $y;
}

sum(2, 5);
```

Разработчик может использовать:

1. передачу значений аргументов «по значению»;
2. передачу значений аргументов «по ссылке»;
3. значения аргументов по умолчанию;
4. переменное количество аргументов.

Передача значений аргументов по ссылке

По умолчанию значения аргументов передаются в функцию «*по значению*», т. е. в виде копий.

Объекты всегда передаются по ссылке!

Если необходимо изменять сами аргументы, то их необходимо передавать *по ссылке:*

```
function concat(&$output, $str) {
    $output .= $str;
}
```

Если необходимо передать аргумент по ссылке в функцию, которая принимает аргумент по значению:

```
function concat($output, $str) {
    $output .= $str;
}
$str = 'Hello';
concat(&$str, ' world!');
```

Значения аргументов по умолчанию

В определении функции можно использовать синтаксис C++ для задания *значений аргументов по умолчанию*.

```
function fee($type='Яблоко') {
    return 'Имеем фрукт: ' . $type;
}

fee();          #выводит: "Имеем фрукт: Яблоко"
fee('Груша');   #выводит: "Имеем фрукт: Груша"
```

Значения по умолчанию должны быть константными.

Можно задавать в качестве значений по умолчанию массивы и NULL.

Аргументы, имеющие значения по умолчанию должны стоять в объявлении функции после обязательных аргументов.

Неопределенное количество аргументов

int func_num_args (void)

Возвращает количество аргументов, переданных функции.

mixed func_get_arg (int arg_num)

Возвращает указанный аргумент из списка аргументов пользовательской функции.

array func_get_args (void)

Возвращает массив, содержащий аргументы функции.

Возвращение функциями значений

Значения возвращаются функциями при помощи инструкции *return*.

Возвращаться может значение любого типа.

Инструкция return прекращает исполнение функции,

даже если в теле функции остались другие инструкции.

```
function square($x) {
   return $x * $x;
}

square(4); #выводит: 16
```

Инструкция return может располагаться только в теле функции.

Инструкция return может использоваться без выражения, тогда она

просто прерывает исполнение функции, не возвращая значения.

Задача №2.3

Написать функцию, которая выводит количество дней, оставшихся до нового года. Функция должна корректно работать при запуске в любом году, т. е. грядущий год должен вычисляться программно.

```
function getDaysToNewYear() {
    # В PHP выше 5.1 вместо mktime() без аргументов
    # использовать time()
    $now = mktime();
    $newYear   mktime(0  0, 0, 1, 1, date('Y')   1);

    return (int) ($newYear   $now)   (60   60   24)
}

echo getDaysToNewYear()
```

Рекурсивная функция

Рекурсивная функция – это функция, которая в ходе выполнения вызывает саму себя.

Прямая рекурсия – содержит в своем теле вызов самой себя.

```
function factorial($x) {
   return $x === 0 ? 1 : $x * factorial($x - 1);
}
factorial(7); #выводит: 5040
```

Косвенная рекурсия – функция вызывает другую функцию, которая в свою очередь вызывает первую.

```
function A() {
   B();
}

function B() {
   A();
}
```

Урок №2

Область видимости
и
Время жизни переменной

Область видимости переменной

Область видимости переменной (scope) – это контекст, в котором эта переменная определена.

Область видимости ограничена текущим файлом, но распространяется на файлы, присоединяемые директивами include и require.

Локальной областью считается код функций и определения классов.

К *глобальной области* относится весь код сценария PHP, кроме кода функций и определения классов.

Суперглобальный массив $GLOBALS

Содержит все глобальные переменные, используемые вне функций.

Статические переменные.

Пространство имен.

Только классы, интерфейсы, функции и константы зависят от него.

Проверка существования переменной

bool isset (mixed var [, mixed …])

Определяет была ли установлена переменная значением, отличным от NULL.

Если передать более одного аргумента – True только если все переменные определены.

Проверка происходит слева направо до первой неопределенной переменной.

Уничтожение переменной

void unset (mixed var [, mixed …])

Удаляет перечисленные переменные.

Если переменная, объявленная глобальной или передается ПО ССЫЛКЕ, удаляется внутри функции, то будет удалена только локальная переменная.

Переменная в области видимости вызова функции сохранит то же значение, что и до вызова unset().

Время жизни переменной

Время жизни переменной — интервал от объявления (декларации) переменной до ее уничтожения.

Время жизни глобальных переменных начинается с момента объявления и заканчивается в двух случаях:

1. уничтожили непосредственно в самой программе с помощью unset();
2. завершилась работа сценария.

Время жизни локальных переменных ограничено временем выполнения функции.

Время жизни статических переменных аналогично времени жизни глобальных переменных.

Задача №2.4

Мы начинаем разрабатывать основу для небольшого фреймворка.

Необходимо написать функцию, которая в качестве аргумента принимает псевдоним файлового пути и возвращает реальный файловый путь.

Псевдоним файлового пути – ассоциируется с путем к файлу, где разделителем директорий служит точка.

```
# пример использования
getPathOfAlias('root.components')

# функция вернет root/components в Linux
# функция вернет root\components в Windows
```

Конец