PHP. Элементы языка.

- **Элементы языка:**
  - **Константы и выражения**
  - **Функции**
  - **Классы**
  - **Операторы**
  - **Регулярные выражения**

Константы и выражения

- Любой скрипт PHP состоит из последовательности операторов. Оператор может быть присваиванием, вызовом функции, циклом, условным выражением или пустым выражением.
- Операторы обычно заканчиваются точкой с запятой. Также операторы могут быть объединены в группу заключением группы операторов в фигурные скобки. Группа операторов также является оператором.
- PHP определяет несколько констант и предоставляет механизм для определения Ваших констант. Константы похожи на переменные, но они имеют иной синтаксис.

Константы

- Предопределенные константы - это __FILE__ and __LINE__, которые соответствуют имени файла и номеру строки, которая выполняется в настоящий момент.

```
<?php
function report ($file, $line,
  $message) {
    echo "An error occured in $file on
  line $line: $message.";}
report (__FILE__,__LINE__, "Something
  went wrong!");
?>
```

Константы

- Можно определить дополнительные константы с помощью функций **define()** и **undefine()**.

```
<?php
define("CONSTANT","Hello
  world.");
echo CONSTANT;
undefine ("CONSTANT");
?>
```

Арифметические операции

пример	название	результат
$a + $b	Сложение	Сумма $a и $b.
$a - $b	Вычитание	Вычитает $b из $a
$a * $b	Умножение	Произведение $a и $b.
$a / $b	Деление	Деление $a на $b.
$a % $b	Остаток деления	Остаток от деления $a на $b.

Оператор деления ("/") возвращает целую величину (результат целочисленного деления) если оба оператора - целые (или строка преобразованная в целое). Если каждый операнд является величиной с плавающей запятой, выполнится деление с плавающей запятой.

Операторы строк и присваивания

- **Операторы строк.** В действительности есть только один оператор - конкатенации (".").

```
$a = "Hello ";
$b = $a . "World!";
```

- **Операторы присваивания.** Основным оператором присваивания является "=".

```
$a = ($b = 4) + 5;
```

- есть дополнительные "операторы с присваиванием" для всех арифметических и строковых операторов:

```
$a = 3; $a += 5; $b = "Hello ";
$b .= "There!";
```

Бинарные Операторы

пример	название	результат
$a & $b	И	$a=5; /* 0101 */ $b=12; /*1100*/ $c=$a & $b; /* 4 (0100) */
$a | $b	Или	$a=5; /*0101*/ $b=12;/*1100*/ $c=$a|$b;/*1101*/
~ $a	Не	$a=5; /* 0101 */ ~ $a; /*1010 */

Логические операторы.

пример	имя	результат
$a and $b	И	Истина, если истинны $a и $b.
$a or $b	Или	Истина, если истинны $a или $b.
$a xor $b	Или	Истина, если истинны $a или $b, но не оба.
! $a	Не	Истина, если не истинно $a.
$a && $b	И	Истина, если истинны и $a и $b.
$a || $b	Или	Истина, если истинны $a или $b.

Разница двух различных вариантов операторов "and" и "&&" - в различии приоритетов.

Операторы Сравнения

пример	результат
$a = = $b	истина, если $a эквивалентно $b.
$a != $b	Истина, если $a не эквивалентно $b.
$a < $b	Истина если $a меньше чем $b
$a > $b	Истина если $a больше $b.
$a <= $b	Истина, если $a меньше или равно $b.
$a >= $b	Истина, если $a больше или равно $b.

Выражения

- В PHP почти всё является выражениями. Простейший пример, приходящий на ум - это константы и переменные. Когда вы печатаете "$a = 5", вы присваиваете значение '5' переменной $a.
- После этого присваивания вы считаете значением $a 5, также, если вы напишете $b = $a, вы будете ожидать того же как, если бы вы написали $b = 5. Другими словами, $a это также выражение со значением 5. Запись типа '$b = ($a = 5)' похожа на запись'$a = 5; $b = 5;' (точка с запятой отмечает конец выражения). Так как присваивания рассматриваются справа налево, вы также можете написать '$b = $a = 5'.

Выражения

- Более сложные примеры выражений - это функции:

```
function foo () { return 5; }
```

- Функции - это выражения с тем значением, которое они возвращают. Так как foo() возвращает 5, значение выражение 'foo()' - 5.
- PHP поддерживает 3 скалярных типа значений: целое, число с плавающей точкой и строки. PHP поддерживает 2 составных (нескалярных) типа: массивы и объекты. Каждое из таких значений может быть присвоено переменной или возвращено функцией.

Выражения

- Во многих случаях, в основном в условных операторах и операторах циклов, важно, являются ли их значения TRUE или FALSE (в PHP нет специального типа boolean).
- Любое не нулевое целое значение - это TRUE, ноль - это FALSE. Обратите внимание на то, что отрицательные значения - это не ноль и поэтому они считаются равными TRUE. Пустая строка и строка '0' это FALSE; все остальные строки - TRUE. И насчёт составных типов (массивы и объекты) - если значение такого типа не содержит элементов, то оно считается равным FALSE; иначе подразумевается TRUE.

Выражения

- **IF.** Возможности PHP по использованию выражения IF похожи на C:

```
if (expr) statement
```

- Вычисляется логический результат "expr" . Если expr равно TRUE, то PHP выполнит "statement", а если FALSE - проигнорирует.

```
if ($a > $b) { print "a is bigger than
  b"; $b = $a; }
```

- Выражение IF может иметь неограниченную степень вложенности в другие выражения IF, что позволяет Вам использовать выполнение по условию различных частей программы.

Выражения

- **ELSE.** ELSE расширяет возможности IF по части обработки вариантов выражения, когда оно равно FALSE. Данный пример выведет фразу 'a is bigger than b' если $a больше $b, и 'a is NOT bigger than b' в противном случае:

```
if ($a>$b){print "a is bigger than b";}
else {print "a is NOT bigger than b"; }
```

- Выражение ELSE выполняется только если выражение IF равно FALSE, а если есть конструкции ELSEIF - то если и они также равны FALSE.

Выражения

- **ELSEIF.** Является комбинацией IF и ELSE. ELSEIF, как и ELSE позволяет выполнить выражение, если значение IF равно FALSE, но в отличие от ELSE, оно выполнится только если выражение ELSEIF равно TRUE:

```
if ($a>$b){print "a is bigger than b";}
elseif ($a= =$b){print "a is equal b";}
else { print "a is smaller than b"; }
```

- Внутри одного выражения IF может быть несколько ELSEIF. Первое выражение ELSEIF (если таковые есть), которое будет равно TRUE, будет выполнено. В PHP вы можете написать 'else if' (два слова), что будет значить то же самое, что и 'elseif' (одно слово).

Выражения

- **IF(): ... ENDIF.** Наиболее часто это используется, когда вы внедряете блоки HTML внутрь оператора IF. Вместо использования фигурных скобок за "IF(выражение)" должно следовать двоеточие, одно или несколько выражений и завершающий ENDIF:

```
<? if ($a==5): ?> A = 5 <?php endif; ?>
```

- Блок HTML будет виден, если $a равно 5.
- Этот альтернативный синтаксис применим и к ELSE и ELSEIF (expr) :

```
if ($a == 5): print "a equals 5";
elseif ($a == 6): print "a equals 6";
else: print "a is neither 5 nor 6";
endif;
```

Выражения

- **`WHILE(expr) statement`** – предписывает выполнять **`statement`** до тех пор, пока **expr** равно **TRUE**. Значение **expr** проверяется в начале цикла, и если значение **expr** изменится внутри цикла, то он не прервётся до конца текущей итерации. Если изначально **expr** равно **FALSE**, цикл не выполняется ни разу.
- Можно сгруппировать несколько операторов внутри фигурных скобок или использовать альтернативный синтаксис:

```
WHILE(expr): выражения ... ENDWHILE;
```

- Следующие примеры идентичны:

```
$i = 1; while ($i <= 10) {print $i++;}
$i = 1;while ($i<= 10): print $i; $i++;
endwhile;
```

Выражения

- В DO..WHILE значение логического выражения проверяется не до, а после окончания итерации и он гарантировано выполнится хотя бы один раз. Для циклов DO..WHILE существует только один вид синтаксиса:

```
$i = 0; do { print $i; } while ($i>0);
```

- Программисты на C знакомы с иным использованием DO..WHILE:

```
do{if($i<5){print"I isn't big";break; }
    $i *= $factor;
    if ($i < $minimum_limit) { break; }
    print "i is ok";
} while(0);
```

Выражения

- **`FOR(expr1; expr2; expr3) statement`**
- expr1 безусловно вычисляется в начале цикла.
- В начале каждой итерации вычисляется expr2. Если оно равно TRUE, то выполняются **`statement.`** Если FALSE, то цикл заканчивается.
- В конце каждой итерации вычисляется expr3.
- Каждое из этих выражений может быть пустым. Если expr2 пусто, то цикл продолжается бесконечно.
- Примеры:

```
for ($i = 1; $i <= 10; $i++) print $i;
for ($i = 1;;$i++) {if ($i > 10) break;
   print $i;}
```

Выражения

- Примеры:

```
$i = 1; for (;;) {if ($i > 10) break;
print $i; $i++;}
for ($i = 1; $i <= 10; print $i, $i++);
```

- **PHP** также поддерживает альтернативный синтаксис **FOR :**

```
FOR (expr1; expr2; expr3): выражение;
  ...; endfor;
```

- Другие языки используют оператор **foreach** для того, чтобы обрабатывает массивы или списки. PHP использует для этого оператор **while** и функции **list()** и **each()**.

Выражения

- **BREAK.** Прерывает выполнение текущего цикла.

```
$i = 0; while ($i < 10) {
        if ($arr[$i] == "stop") { break; }
        $i++;
}
```

- **CONTINUE.** Переходит на начало ближайшего цикла.

```
while (list($key,$value) = each($arr))
  {
       if ($key % 2) {continue; }
      do_something_odd ($value);
}
```

Выражения

- Оператор SWITCH похож на группу операторов IF с одинаковым выражением.

```
switch ($i) {
    case 0: print "i = 0"; break;
     case 1: print "i = 1"; break;
     case 2: print "i = 2"; break;
   default: print "i!= 0, 1 or 2";
}
```

- Выражения в CASE могут быть любого скалярного типа, то есть целые числа или числа с плавающей запятой, а так же строки. Массивы и объекты не будут ошибкой.

REQUIRE и INCLUDE

- Оператор REQUIRE заменяет себя содержимым указанного файла, похоже на то, как в препроцессоре C работает #include.

```
require ('header.inc');
```

- Оператор INCLUDE вставляет и выполняет содержимое указанного файла. Это случается каждый раз, когда встречается оператор INCLUDE, так что вы можете включить этот оператор внутрь цикла, чтобы включить несколько файлов :

```
$files = array ('first.inc',
  'second.inc', 'third.inc');
for ($i = 0; $i < count($files); $i++)
  { include($files[$i]); }
```

REQUIRE и INCLUDE

- **include()** отличается от **require()** тем, что **include** выполняется каждый раз при его встрече, а **require()** заменяется на содержимое указанного файла безотносительно будет ли выполнено его содержимое или нет.
- include() - специальный оператор, его требуется заключать в фигурные скобки при использовании внутри условного оператора:

```
if ($condition) { include($file);}
else { include($other); }
```

- Когда файл исполняется, парсер пребывает в "режиме HTML", то есть будет выводить содержимое файла, пока не встретит первый стартовый тег PHP (<?).

FUNCTION

- Функция может быть объявлена так:

```
function foo ($arg_1,..., $arg_n) {
    echo "Example function.\n";
    return $retval; }
```

- Внутри функции может быть объявление другой функции или класса. Функции должны быть определены перед тем, как на них ссылаться.
- Результаты возвращаются через необязательный оператор **return**. Возвращаемый результат может быть любого типа, включая списки и объекты.

```
function my ($num) {return $num*$num; }
echo my (4);   // outputs '16'.
```

FUNCTION

- Множественные результаты не могут быть возвращены в качестве результата, но вы можете реализовать это путём возврата списка:

```
function foo() {
    return array (0, 1, 2); }
list ($zero, $one, $two) = foo();
```

- Информация может быть передана функции через список аргументов, которые являются разделенным запятыми списком переменных и/или констант.
- PHP поддерживает передачу аргументов по значению (по умолчанию), по ссылке, и значения по умолчанию.

FUNCTION

- Списки аргументов переменной длины не поддерживаются, но можно передать массивы:

```
function takes_array($input) {
    echo "$input[0] + $input[1] = ",
  $input[0]+$input[1];}
```

- По умолчанию аргументы передаются по значению. Если надо в функции модифицировать аргументы, передают их по ссылке, поставив амперсанд (&) перед именем аргумента в объявлении функции:

```
function foo( &$bar ) {
    $bar .= ' and something extra.'; }
$str = 'This is a string, ';
foo ($str); echo $str;
```

FUNCTION

- Функции могут определять значения по умолчанию для скалярных аргументов:

```
function makecoffee ($type = "cappucino") {
    echo "Making a cup of $type.\n";
}
echo makecoffee ();
echo makecoffee ("espresso");
```

- Этот пример выведет следующее :

```
Making a cup of cappucino.
Making a cup of espresso.
```

FUNCTION

- Значение по умолчанию должно быть константой, а не переменной или, к примеру, членом класса. Учтите, что когда вы объявляете аргументы по умолчанию, они должны быть справа от всех "неумолчиваемых" аргументов:

```
function makeyogurt ($flavour,
  $type = "acidophilus") {
     return "Making a bowl of
  $type $flavour.\n";
}
echo makeyogurt ("raspberry");
```

Классы

- Класс определяется следующим образом :

```
<?php class Cart {
      var $items;
function add_item ($artnr, $num) {
  $this->items[$artnr] += $num; }
function remove_item ($artnr, $num) {
  if($this->items[$artnr] > $num) {
          $this->items[$artnr]-= $num;
          return true; }
  else { return false; }
 }
}?>
```

Классы

- Вы должны создавать переменные класса, используя оператор new :

```
$cart = new Cart;
$cart->add_item("10", 1);
```

- Классы могут быть расширениями других классов. Расширенный класс обладает всеми переменными и функциями базового класса и тем, что вы определите при расширении класса. Это делается, используя ключевое слово extends:

```
class Named_Cart extends Cart {
    var $owner;
function set_owner ($name) {
        $this->owner = $name; }
}
```

Классы

- Внутри функций класса переменная **$this** означает сам объект. Вы должны использовать **$this->нечто** для доступа к переменной или функции с именем **'нечто'** внутри объекта.
- Конструкторы - это функции в классе, которые автоматически вызываются, когда вы создаёте новую переменную данного класса. Функция становится конструктором, когда она имеет такое же имя, как и сам класс.

```
class Auto_Cart extends Cart {
    function Auto_Cart () {
        $this->add_item ("10", 1); }
}
```

Классы

- Конструкторы также могут иметь аргументы, и эти аргументы могут быть необязательными, что делает конструктор более полезным:

```
class Constructor_Cart {
    function Constructor_Cart ($item =
  "10", $num = 1) {
        $this->add_item ($item, $num);
    }
}
$default_cart = new Constructor_Cart;
$different_cart = new Constructor_Cart
  ("20", 17);
```

Регулярные выражения

- Регулярные выражения используются для сложного манипулирования строками в PHP. Функции, которые поддерживают регулярные выражения:
  - `ereg()`
  - `ereg_replace()`
  - `eregi()`
  - `eregi_replace()`
  - `split()`
- Все эти функции принимают строку регулярного выражения как их первый параметр.

Регулярные выражения

- Примеры регулярных выражений
  - `ereg("abc",$string); /* Возвращает 'истина', если "abc" найдено в $string. */`
  - `ereg("^abc",$string);/* Возвращает 'истина', если "abc" найдено в начале $string. */`
  - `ereg("abc$",$string);/* Возвращает 'истина', если "abc" найдено в конце $string.*/`
  - `eregi("(ozilla.[23]|MSIE.3)",$HTTP_USER_AGENT);/*Возвращает 'истина', если браузер клиента - Netscape 2, 3 или MSIE 3.*/`

Регулярные выражения

- ereg("([[:alnum:]]+) ([[:alnum:]]+) ([[:alnum:]]+)", $string,$regs);
- /* Помещает три слова - $regs[1], $regs[2] и $regs[3], разделенные пробелом. */
- ereg_replace("^","
",$string); /* Устанавливает тег
 в начало строки $string. */
- ereg_replace("$","
",$string);
- /* Устанавливает тег
 в конец строки $string. */
- ereg_replace("\n","",$string);
- /* Отсекает символ "возврат каретки" в строке $string. */