**ПРОГРАММИРОВАНИЕ НА PHP**

[ЛЕКЦИЯ 1. ПОНЯТИЯ «КЛИЕНТ» И «СЕРВЕР». ИСТОРИЯ ПОЯВЛЕНИЯ PHP 2](#__RefHeading___Toc511251982)

[1.1. Понятия «Клиент» и «Сервер» 2](#__RefHeading___Toc511251983)

[1.2. Веб-сервер и интерфейс CGI 3](#__RefHeading___Toc511251984)

[1.3. История появления PHP 3](#__RefHeading___Toc511251985)

[ЛЕКЦИЯ 2. ТИПЫ ДАННЫХ И ОПЕРАТОРЫ 6](#__RefHeading___Toc511251986)

[2.1. Алфавит языка 6](#__RefHeading___Toc511251987)

[2.2. Операторы языка 7](#__RefHeading___Toc511251988)

[2.3. Типы данных 8](#__RefHeading___Toc511251989)

[ЛЕКЦИЯ 3. УПРАВЛЯЮЩИЕ СТРУКТУРЫ 11](#__RefHeading___Toc511251990)

[ЛЕКЦИЯ 4. ФУНКЦИИ, ОПРЕДЕЛЯЕМЫЕ ПОЛЬЗОВАТЕЛЕМ 15](#__RefHeading___Toc511251991)

[4.1. Понятие функции 15](#__RefHeading___Toc511251992)

[4.2. Аргументы функции 15](#__RefHeading___Toc511251993)

[4.3. Списки аргументов переменной длины 16](#__RefHeading___Toc511251994)

[4.4. Использование переменных внутри функции 17](#__RefHeading___Toc511251995)

[4.4.1. Глобальные переменные 17](#__RefHeading___Toc511251996)

[4.4.2. Статические переменные 18](#__RefHeading___Toc511251997)

[4.5. Оператор RETURN 18](#__RefHeading___Toc511251998)

[4.6. Возвращение ссылки 19](#__RefHeading___Toc511251999)

[4.7. Переменные функции 19](#__RefHeading___Toc511252000)

# ЛЕКЦИЯ 1. ПОНЯТИЯ «КЛИЕНТ» И «СЕРВЕР». ИСТОРИЯ ПОЯВЛЕНИЯ PHP

## 1.1. Понятия «Клиент» и «Сервер»

В общем случае под клиентом и сервером понимаются два взаимодействующих процесса, из которых один является поставщиком некоторого сервиса для другого.

**Сервер** – логический процесс, который обеспечивает некоторый сервис по запросу от клиента. Обычно сервер не только выполняет запрос, но и управляет очередностью запросов, буферами обмена, извещает своих клиентов о выполнении запроса и т. д.

**Клиент** – процесс, который запрашивает обслуживание от сервера. Процесс не является клиентом по каким-то параметрам своей структуры, он является клиентом только по отношению к серверу. При взаимодействии клиента и сервера инициатором диалога с сервером, как правило, является клиент, сервер сам не инициирует совместную работу. Это не исключает, однако, того, что сервер может извещать клиентов о каких-то зарегистрированных им событиях.

Типы серверов:

* *Сервер печати (принт-сервер)* — присоединение принтера к сети через специализированный узел обработки заданий на печать;
* *Сервер БД* — совместно используемая БД;
* *Файл-сервер* — хранилище данных, т.е. это серверы, разделяемым ресурсом которых является дисковая память;
* *Коммуникационный сервер* — управляет доступом к удалённым ресурсам, обеспечивает канал связи с глобальной вычислительной сетью;
* *Сервер приложений* — выполняются специальные вычислительные задания (обработка графики);
* *Веб-сервер* — Обеспечивает доступ к веб-страницам.

Компьютеры, использующие сетевые ресурсы сервера, называются клиентами. Взаимодействие с серверами прозрачно для пользователя, поскольку компьютер сам определяет место нахождения требуемого ресурса, и сам получает к нему доступ.

Каждый компьютер сети имеет уникальное сетевое имя, позволяющее однозначно его идентифицировать. Для каждого пользователя серверной сети необходимо иметь свое сетевое имя и сетевой пароль. Имена компьютеров, сетевые имена и пароли пользователей прописываются на сервере.

В функции «клиента» входит:

* Предоставление пользовательского интерфейса, ориентированного на определённые производственные обязанности и полномочия;
* Формирования запросов к серверу;
* Анализ ответов сервера на запросы и предъявление их пользователю.

Для удобства управления компьютерной сетью, несколько компьютеров, имеющих равные права доступа, объединяют в рабочие группы. Рабочая группа – группа компьютеров в локальной сети.

Совокупность приемов разделения и ограничения прав доступа участников компьютерной сети к ресурсам называется политикой сети. Обеспечением работоспособности сети и ее администрированием занимается системный администратор – человек, управляющий организацией работы компьютерной сети.

Рабочая станция — это индивидуальное рабочее место пользователя. На рабочих станциях устанавливается обычная операционная система. Кроме того, на рабочих станциях устанавливается клиентская часть сетевой операционной системы. Полноправным владельцем всех ресурсов рабочей станции является пользователь, тогда как ресурсы файл-сервера разделяются всеми пользователями. В качестве рабочей станции может использоваться компьютер практически любой конфигурации. Но, в конечном счете, все зависит от тех приложений, которые этот компьютер выполняет.

Основная функция сервера — выполнение специфический действий по запросам клиента (например, решение сложной математической задачи, поиск данных в базе, соединение клиента с другим клиентом и другое); при этом сам сервер не имеет никаких взаимодействий с клиентом. Если сервер, к которому обратился клиент, не в состоянии решить задачу из-за нехватки ресурсов, то в идеале он сам находит другой, более мощный, сервер и передаёт задачу ему, становясь, в свою очередь, клиентом, но не информируя об этом без нужды начального клиента.

## 1.2. Веб-сервер и интерфейс CGI

**Web-сервер** — сервер, принимающий HTTP-запросы от клиентов, обычно веб-браузеров, и выдающий им HTTP-ответы, как правило, вместе с HTML-страницей, изображением, файлом, медиа-потоком или другими данными. Веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно компьютер, на котором это программное обеспечение работает.

*Что в действительности происходит, когда пользователь вводит какой-либо URL в адресную строку браузера*?

Браузер извлекает из URL протокол (HTTP) и адрес web-сервера. Далее браузер обращается к серверу с HTTP-запросом, в котором указано, что необходимо получить документ */path/somefile.html*. Сервер просматривает содержимое каталога, и если находит документ *somefile.html*, то возвращает его браузеру в обыкновенном текстовом виде. Кроме текста сервер также возвращает тип документа (формат). Например, если web-сервер сообщает, что файл *somefile.html* является текстом, то в окне браузера отобразится HTML-код. Но обычно сервер сообщает браузеру, что *somefile.html* является гипертекстовым документом (html), поэтому пользователь увидит нормальную html-страницу. Если запрашиваемый браузером документ не будет найден, то сервер отправит сообщение об ошибке.

Термин CGI (Common Gateway Interface — общий шлюзовой интерфейс) обозначает набор соглашений, которые должны соблюдаться Web-серверами при выполнении ими различных Web-приложений. Интерфейс CGI выполняет функции шлюза между различными программами, установленными на web-сервере, и браузерами пользователей. Когда браузер клиента запрашивает определенный ресурс, web-сервер запускает CGI-программу, которая возвращает результат обратно web-серверу, после чего он отправляет ответ клиенту. Использование CGI-программ требуется, когда необходимо в браузере пользователя отобразить страницу, сформированную на основе каких-либо действий пользователя.

Если программа предназначена для CGI, то нужно позаботиться о взаимодействии с сервером. Простейшее взаимодействие заключается в том, что программист должен знать информацию о сервере. Данная информация передается с помощью переменных окружения.

## 1.3. История появления PHP

Язык PHP был разработан как инструмент для решения чисто практических задач. Его создатель, **Расмус Лердорф**, хотел знать, сколько человек читают его online-резюме, и написал для этого простенькую CGI-оболочку на языке Perl, т.е. это был набор Perl-скриптов, предназначенных исключительно для определенной цели – сбора статистики посещений.

Вскоре выяснилось, что оболочка обладает небольшой производительностью, и пришлось переписать ее заново, но уже на языке Си. После этого исходники были выложены на всеобщее обозрение для исправления ошибок и дополнения. Пользователи сервера, где располагался сайт с первой версией PHP, заинтересовались инструментом, появились желающие его использовать. Так что скоро PHP превратился в самостоятельный проект, и в начале 1995 года вышла первая известная версия продукта, называвшаяся Personal Home Page Tools (средства для персональной домашней страницы). Средства эти были более чем скромными: анализатор кода, понимающий всего лишь несколько специальных команд, и набор утилит, полезных для создания гостевой книги, счетчика посещений, чата и т.п.

К середине 1995 года после основательной переработки появилась вторая версия продукта, названная PHP/FI (Personal Home Page / FormsInterpreter – персональная домашняя страница/ интерпретатор форм). Она включала набор базовых возможностей сегодняшнего PHP, возможность автоматически обрабатывать html-формы и встраиваться в html-коды. Синтаксис PHP/FI сильно напоминал синтаксис Perl, но был более простым.

В 1997 вышла вторая версия Cи-реализации PHP – PHP/FI 2.0. К тому моменту PHP использовали уже несколько тысяч человек по всему миру, примерно с 50 тыс. доменов, что составляло около 1% всего числа доменов Internet. Число разработчиков PHP увеличилось до нескольких человек, но, несмотря на это, PHP/FI 2.0 все еще оставался крупным проектом одного человека. Официально PHP/FI 2.0 вышел только в ноябре 1997 года, просуществовав до этого в основном в бета-версиях. Вскоре после выхода его заменили альфа-версии PHP 3.0.

PHP 3.0 была первой версией, напоминающей PHP, каким мы знаем его сегодня. Он очень сильно отличался от PHP/FI 2.0 и появился опять же как инструмент для решения конкретной прикладной задачи. Его создатели, Энди Гутманс (Andi Gutmans) и Зив Сураски (Zeev Suraski), в 1997 году переписали заново код PHP/FI, поскольку он показался им непригодным для разработки приложения электронной коммерции, над которым они работали. Для того чтобы получить помощь в реализации проекта от разработчиков PHP/FI, Гутманс и Сураски решили объединиться с ними и объявить PHP3 официальным преемником PHP/FI. После объединения разработка PHP/FI была полностью прекращена.

Одной из сильных сторон PHP 3.0 была возможность расширения ядра. Именно свойство расширяемости PHP 3.0 привлекло вниманием множества разработчиков, желающих добавить свой модуль расширения. Кроме того, PHP 3.0 предоставляла широкие возможности для взаимодействия с базами данных, различными протоколами и API. Немаловажным шагом к успеху оказалась разработка нового, намного более мощного и полного синтаксиса с поддержкой ООП. С момента появления PHP 3.0 изменилась не только функциональность и внутреннее устройство языка, но и его название. В аббревиатуре PHP больше не было упоминания о персональном использовании, PHP стало сокращением (рекурсивным акронимом) от PHP: Hypertext Preprocessor, что значит "PHP: препроцессор гипертекста".

К концу 1998 года число пользователей PHP возросло до десятков тысяч. Сотни тысяч web-сайтов сообщали о том, что они работают с использованием этого языка. Почти на 10% серверов Internet был установлен PHP 3.0.

Официально PHP 3.0 вышел в июне 1998 года, после 9 месяцев публичного тестирования. А уже к зиме Энди Гутманс и Зив Сураски начали переработку ядра PHP. В их задачи входило увеличение производительности работы сложных приложений и улучшение модульности кода, лежащего в основе PHP.

Новое ядро было названо "Zend Engine" (от имен создателей: Zeev и Andi) и впервые представлено в середине 1999 года. PHP 4.0, основанный на этом ядре и принесший с собой набор дополнительных функций, официально вышел в мае 2000 года, почти через два года после своего предшественника, PHP 3.0. Помимо улучшения производительности, PHP 4.0 имел еще несколько ключевых нововведений, таких как поддержка сессий, буферизация вывода, более безопасные способы обработки вводимой пользователем информации и несколько новых языковых конструкций.

В настоящее время ведутся работы по улучшению Zend Engine и внедрению нововведений в PHP 5.0, первые бета-версии которого уже вышли в свет. Одно из существенных изменений произошло в объектной модели языка, ее основательно подлатали и добавили много новых возможностей.

Пятая версия PHP была выпущена разработчиками 13 июля 2004 года. Изменения включают обновление ядра Zend (Zend Engine 2), что существенно увеличило эффективность интерпретатора. Введена поддержка языка разметки XML. Полностью переработаны функции ООП, которые стали во многом схожи с моделью, используемой в Java. В частности, введён деструктор, открытые, закрытые и защищённые члены и методы, окончательные члены и методы, интерфейсы и клонирование объектов. В последующих версиях также были введены пространства имён, замыкания и целый ряд достаточно серьёзных изменений, количественно и качественно сравнимых с теми, которые появились при переходе на PHP 5.0.

Шестая версия PHP разрабатывалась с октября 2006 года. Было сделано множество нововведений, как, например, исключение из ядра регулярных выражений POSIX и "длинных" суперглобальных массивов, удаление директив safe\_mode, magic\_quotes\_gpc и register\_globals из конфигурационного файла php.ini. Одним из основных новшеств должна была стать поддержка Юникода. Однако в марте 2010 года разработка PHP6 была признана бесперспективной из-за сложностей с поддержкой Юникода. Исходный код PHP6 перемещён на ветвь, а основной линией разработки стала версия 5.4.

Возможности PHP:

* Создание CGI-приложений (скриптов), которые исполняются на стороне сервера. PHP наиболее широко используется именно для создания такого рода скриптов. Для того чтобы работать таким образом, понадобится PHP-парсер (т.е. обработчик php-скриптов), web-сервер для обработки скрипта, браузер для просмотра результатов работы скрипта, и, конечно, какой-либо текстовый редактор для написания самого php-кода. Парсер PHP распространяется в виде CGI-программы или серверного модуля.
* Создание скриптов, выполняющихся в командной строке. То есть с помощью PHP можно создавать такие скрипты, которые будут исполняться вне зависимости от web-сервера и браузера, на конкретной машине. Для такой работы потребуется только парсер PHP (в этом случае его называют интерпретатором командной строки (cli, command line interpreter)). Этот способ работы подходит, например, для скриптов, которые должны выполняться регулярно с помощью различных планировщиков задач или для решения задач простой обработки текста.
* Создание GUI-приложений (графических интерфейсов), выполняющихся на стороне клиента. Для применения PHP в этой области потребуется специальный инструмент – PHP-GTK, который является расширением PHP.

# ЛЕКЦИЯ 2. ТИПЫ ДАННЫХ И ОПЕРАТОРЫ

## 2.1. Алфавит языка

Алфавит языка включает в себя:

* Буквы (латинские и национальных алфавитов);
* Символ нижнего подчеркивания;
* Цифры;
* Специальные символы;
* Пробельные символы.

Лексема – минимальная единица языка, имеющая самостоятельный смысл. Виды лексем:

* Идентификаторы;
* Ключевые слова;
* Знаки операций;
* Разделители;
* Константы (литералы).

Переменная – поименованная область памяти, в которой в каждый момент времени может быть только одно значение. У каждой переменной есть имя, тип и текущее значение. Переменная в PHP обозначается знаком доллара, за которым следует ее имя. Имя переменной чувствительно к регистру. Правильное имя переменной должно начинаться с буквы или символа подчеркивания с последующими в любом количестве буквами, цифрами или символами подчеркивания.

Например:

$my\_var;

Переменные могут присваиваться *по значению* и *по ссылке*. В случае присвоения переменных по значению (присвоение одной переменной значения другой) изменение одной из них не влияет на значение другой. Для того чтобы присвоить значение переменной по ссылке, это значение должно иметь имя, т.е. оно должно быть представлено какой-либо переменной. Чтобы указать, что значение одной переменной присваивается другой переменной по ссылке, нужно перед именем первой переменной поставить знак амперсанд &.

<?php

$first = ' Text ';

$second = &$first;

$first = ' New text ';

echo "Переменная с именем first " .

"равна .$first <br>";

echo "Переменная с именем second " .

"равна $second";

?>

Переменная с именем first равна New text.

Переменная с именем second равна New text.

Для хранения постоянных величин, т.е. таких величин, значение которых не меняется в ходе выполнения скрипта, используются константы. Для определения констант используется функция define():

define("Имя\_константы", "Значение\_константы", [“Нечувствительность\_к\_регистру”])

Кроме констант, объявляемых пользователем, в PHP существует ряд констант, определяемых самим интерпретатором.

\_\_FILE\_\_ хранит имя файла программы (и путь к нему), которая выполняется в данный момент;

\_\_FUNCTION\_\_ содержит имя функции;

\_\_CLASS\_\_ содержит имя класса;

PHP\_VERSION хранит версию интерпретатора PHP.

## *2.2. Операторы языка*

Операторы позволяют выполнять различные действия с переменными, константами и выражениями.

Таблица 2.1. Арифметические операторы

|  |  |  |
| --- | --- | --- |
| **Обозначение** | **Название** | **Пример** |
| + | Сложение | $a + $b |
| - | Вычитание | $a - $b |
| \* | Умножение | $a \* $b |
| / | Деление | $a / $b |
| % | Остаток от деления | $a % $b |

«.» - строковый оператор, выполняет конкатенацию строк.

Пример:

$c = $a . $b;

Таблица 2.2. Логические операторы

|  |  |  |  |
| --- | --- | --- | --- |
| **Обозначение** | **Название** | **Описание** | **Пример** |
| and | И | $a и $b истинны (True) | $a and $b |
| && | И |  | $a && $b |
| or | ИЛИ | Хотя бы одна из переменных $a или $b истинна (возможно, что и обе) | $a or $b |
| || | ИЛИ |  | $a || $b |
| xor | Исключающее ИЛИ | Одна из переменных истинна. Случай, когда они обе истинны, исключается | $a xor $b |
| ! | Инверсия (NOT) | Если $a=True, то !$a=False и наоборот | !$a |

Таблица 2.3. Операторы сравнения

|  |  |  |  |
| --- | --- | --- | --- |
| **Обозначение** | **Название** | **Описание** | **Пример** |
| == | Равенство | Значения переменных равны | $a == $b |
| === | Эквивалентность | Значения и типы переменных равны | $a === $b |
| != | Неравенство | Значения переменных не равны | $a != $b |
| <> | Неравенство |  | $a <> $b |
| !== | Неэквивалентность | Переменные неэквивалентны | $a !== $b |
| < | Меньше |  | $a < $b |
| > | Больше |  | $a > $b |
| <= | Меньше или равно |  | $a <= $b |
| >= | Больше или равно |  | $a >= $b |

Таблица 2.4. Операторы инкремента и декремента

|  |  |  |  |
| --- | --- | --- | --- |
| **Обозначение** | **Название** | **Описание** | **Пример** |
| ++$a | Пре- инкремент | Увеличивает $a на единицу и возвращает $a | <?  $a=4;  echo "Должно быть 4:" .$a++;  echo "Должно быть 5:" .++$a;  ?> |
| $a++ | Пост- инкремент | Возвращает $a, затем увеличивает $a на единицу |
| --$a | Пре- декремент | Уменьшает $a на единицу и возвращает $a |
| $a-- | Пост- декремент | Возвращает $a, затем уменьшает $a на единицу |

## 2.3. Типы данных

PHP поддерживает восемь простых типов данных.

* Скалярные типы:
  + boolean (логический);
  + integer (целый);
  + float (с плавающей точкой);
  + string (строковый).
* Смешанные типы:
  + array (массив);
  + object (объект).
* Специальные типы:
  + resource (ресурс);
  + NULL.

В PHP не принято явное объявление типов переменных. Предпочтительнее, чтобы это делал сам интерпретатор во время выполнения программы в зависимости от контекста, в котором используется переменная.

**BOOLEAN** – простейший тип данных, выражает истинность значения. Может принимать одно из двух значений – TRUE или FALSE (регистронезависимы).

**INTEGER** – целый тип данных. Множество значений: {N; -N; 0}. Могут быть указаны в десятичной, шестнадцатеричной или восьмеричной системе счисления, по желанию с предшествующим знаком «-» или «+». В восьмеричной системе счисления нужно предварить число «0» (нулем), для использования шестнадцатеричной системы нужно поставить перед числом «0x».

Размер целого зависит от платформы, хотя, как правило, максимальное значение около двух миллиардов (это 32-битное знаковое).

**Замечание**. Беззнаковые целые PHP не поддерживает.

**Замечание**. Если определяется число, превышающее пределы целого типа, оно будет интерпретировано как число с плавающей точкой. Также если используется оператор, результатом работы которого будет число, превышающее пределы целого, вместо него будет возвращено число с плавающей точкой.

**Замечание**. Для преобразования переменной к конкретному типу нужно перед переменной указать в скобках нужный тип.

**FLOAT** – тип данных с плавающей точкой, они же числа с двойной точностью или действительные числа. Могут быть определены при помощи любого из следующих синтаксисов:

<?php

$a = 1.234;

$b = 1.2e3;

$c = 7E-10;

?>

Размер числа с плавающей точкой зависит от платформы (как правило, ~1.8e308 с точностью около 14 десятичных цифр).

**STRING** – набор символов. В PHP 1 символ равен 1 байту. Следовательно, существует ровно 256 различных символов. Это также означает, что PHP не имеет встроенной поддержки Unicode. В PHP практически не существует ограничений на размер строк.

Строка в PHP может быть определена тремя различными способами:

* с помощью одинарных кавычек;
* с помощью двойных кавычек (распознавание управляющих последовательностей);
* heredoc-синтаксисом.

Таблица 2.5. Управляющие последовательности

|  |  |
| --- | --- |
| **Последовательность** | **Значение** |
| \n | Новая строка (LF или 0x0A (10) в ASCII) |
| \r | Возврат каретки (CR или 0x0D (13) в ASCII) |
| \t | Горизонтальная табуляция (HT или 0x09 (9) в ASCII) |
| \\ | Обратная косая черта |
| \$ | Знак доллара |
| \" | Двойная кавычка |

**ARRAY** - это упорядоченное отображение, которое устанавливает соответствие между значением и ключом. Определить массив можно с помощью конструкции array() или непосредственно задавая значения его элементам.

array (key => value,

key1 => value1, ... )

Языковая конструкция array() принимает в качестве параметров пары ключ => значение, разделенные запятыми. Символ => устанавливает соответствие между значением и его ключом. Ключ может быть как целым числом, так и строкой, а значение может быть любого имеющегося в PHP типа. Числовой ключ массива часто называют индексом. Индексирование массива в PHP начинается с нуля. Значение элемента массива можно получить, указав после имени массива в квадратных скобках ключ искомого элемента. Если ключмассива представляет собой стандартную запись целого числа, то он рассматривается как число, в противном случае – как строка. Поэтому запись $a["1"] равносильна записи $a[1], так же как и $a["-1"] равносильно $a[-1].

Если для элемента ключ не задан, то в качестве ключа берется максимальный числовой ключ, увеличенный на единицу. Если указать ключ, которому уже было присвоено какое-то значение, то оно будет перезаписано. Начиная с PHP 4.3.0, если максимальный ключ – отрицательное число, то следующим ключом массива будет ноль (0).

<?php

$arr = array(5 => 43, 32, 56, "b" => 12);

$arr1 = array(5 => 43, 6 => 32,

7 => 56, "b" => 12);

?>

Для того чтобы изменить конкретный элемент массива, нужно присвоить ему с его ключом новое значение. Изменить ключ элемента нельзя, можно только удалить элемент (пару ключ-значение) и добавить новую. Чтобы удалить элементмассива, нужно использовать функцию unset().

<?php

$books = array ("php" =>

"PHP users guide",

12 => true);

$books[] =

"Book about Perl";

$books["lisp"] = 123456;

unset($books[12]);

unset ($books);

?>

Переиндексировать массив можно с помощью функции array\_values().

<?php

$arr = array ("a","b","c"); /\* Создаем массив со значениями "a", "b" и "c". Поскольку ключи не указаны, они будут 0,1,2 соответственно\*/

print\_r($arr); // выводим массив (и ключи, и значения)

unset($arr[0]);

unset($arr[1]);

unset($arr[2]);

// удаляем из него все значения

print\_r($arr); // выводим массив (и ключи, и значения)

$arr[] = "aa"; // добавляем новый элемент в массив. Его индексом (ключом) будет 3, а не 0

print\_r($arr);

$arr = array\_values($arr); // переиндексируем массив

$arr[] = "bb"; // ключом этого элемента будет 1

print\_r($arr);

?>

**OBJECT** – тип данных из объектно-ориентированного программирования (ООП). Согласно принципам ООП, класс – это набор объектов, обладающих определенными свойствами и методами работы с ним, а объект соответственно – экземпляр класса. В PHP для доступа к методам объекта используется оператор ->. Для инициализации объекта используется выражение new, создающее в переменной экземпляр объекта.

<?php

class Person

{

function know\_php()

{

echo "Теперь я знаю PHP";

}

}

$bob = new Person;

$bob -> know\_php;

?>

**RESOURCE** – это специальная переменная, содержащая ссылку на внешний ресурс (например, соединение с базой данных). Ресурсы создаются и используются специальными функциями (например, mysql\_connect(), pdf\_new() и т.п.).

**NULL** - говорит о том, что переменная не имеет значения. Существует только одно значение типа NULL – регистронезависимое ключевое слово NULL.

Переменная считается NULL, если:

* ей была присвоена константа NULL;
* ей еще не было присвоено какое-либо значение;
* она была удалена с помощью unset().

# ЛЕКЦИЯ 3. УПРАВЛЯЮЩИЕ СТРУКТУРЫ

**Условный оператор if.**

if(<условие>){

<блок\_выражений1>;

}

else{

<блок\_выражений2>;

}

либо:

if(<условие1>){

<блок\_выражений 1>;

}

elseif(<условие2>){

<блок\_выражений2>;

}

else{

<блок\_выражений3>;

}

**Оператор множественного выбора** **switch-case**.

switch (<выражение или переменная>){

case значение1:

<блок\_действий1>;

break;

case значение2:

<блок\_действий2>;

break;

...

default:

<блок\_действий\_по\_умолчанию>;

}

**Цикл «while»** – цикл с предусловием.

while (<выражение>) {<блок\_выполнения>;}

либо:

while (<выражение>): <блок\_выполнения> endwhile;

**Цикл** «**do-while»** – цикл с постусловием. Выполнится хотя бы 1 раз.

do {<блок\_выполнения>;} while (<выражение>);

**Цикл** «**foreach»** – предназначена исключительно для работы с массивами.

foreach ($array as $value) {<блок\_выполнения>;}

либо:

foreach ($array as $key => $value){<блок\_выполнения>;}

Пример:

<?php

$names = array("Иван","Петр","Семен");

foreach ($names as $val) {

echo "Привет, $val <br>";

}

foreach ($names as $k => $val) {

echo "Привет, $val ! Ты в списке под номером $k <br>";

}

?>

**Цикл** «**for»** - цикл с параметром. Выполняется определенное количество раз.

for (<выражение1>; <выражение2>; <выражение3>) {<блок\_выполнения>;}

либо:

for (<выражение1>; <выражение2>; <выражение3>): <блок\_выполнения> endfor;

Если опустить второе выражение (условие $i<10), то такую же задачу можно решить, останавливая цикл оператором break.

<?php

for ($i=0; ; $i++){

if ($i>=10) break;

if ($i % 2 == 0) print $i;

}

?>

Можно опустить все три выражения. В этом случае просто не будет задано начальное значение счетчика $i и оно не будет изменяться каждый раз в конце цикла. Все эти действия можно записать в виде отдельных команд либо в <блок\_выполнения>, либо перед циклом:

<?php

$i=2; // задаем начальное значение счетчика

for ( ; ; ){

if ($i>=10) break; // если $i больше или равно 10, то прекращаем цикл

if ($i % 2 == 0) print $i;

$i++; // увеличиваем счетчик на единицу

}

?>

Оператор **break** – завершает работу текущего цикла. Может использоваться с числовым аргументом, который говорит, работу скольких управляющих структур, содержащих его, нужно завершить.

<?php

$i=1;

while ($i) {

$n = rand(1,10);

switch ($n){

case 5:

echo "<font color=blue>Выход из switch (n=$n)</font>";

break 1; // прекращаем работу switch как 1 содержащего break цикла

case 10:

echo "<font color=red>Выход из switch и while (n=$n)</font>";

break 2; // прекращаем работу switch и while как 2-х содержащих break циклов

default:

echo "switch работает (n=$n), ";

}

echo " while работает – шаг $i <br>";

$i++;

}

echo "<br>Число итераций цикла $i ";

?>

Оператор **continue** – позволяет пропустить дальнейшие инструкции из <блок\_выполнения> любого цикла и продолжить выполнение с нового круга. Можно использовать с числовым аргументом, который указывает, сколько содержащих его управляющих конструкций должны завершить работу.

**Замечание**. Оператор continue в конструкциях switch работает так же, как и break. Если switch находится внутри цикла и нужно начать новую итерацию цикла, следует использовать continue 2.

Оператор **include** – позволяет включать код, содержащийся в указанном файле, и выполнять его столько раз, сколько программа встречает этот оператор. Включение может производиться любым из перечисленных способов:

include 'имя\_файла';

include $file\_name;

include ("имя\_файла");

Пример:

Файл params.inc

<?php

$user = "Вася";

$today = date("d.m.y");

?>

------------------

<?php

include("params.inc");

echo "Привет, $user!<br>";

echo "Сегодня $today";

?>

Поиск файла для вставки происходит по следующим правилам.

1. Сначала ведется поиск файла в include\_path относительно текущей рабочей директории.
2. Если файл не найден, то поиск производится в include\_path относительно директории текущего скрипта.
3. Параметр include\_path, определяемый в файле настроек PHP, задает имена директорий, в которых нужно искать включаемые файлы.

Если файл включен с помощью include, то содержащийся в нем код наследует область видимости переменных строки, где появился include. Любые переменные вызванного файла будут доступны в вызывающем файле с этой строки и далее. Соответственно, если include появляется внутри функции вызывающего файла, то код, содержащийся в вызываемом файле, будет вести себя так, как будто он был определен внутри функции. Таким образом, он унаследует область видимости этой функции.

Кроме локальных файлов, с помощью include можно включать и внешние файлы, указывая их url-адреса. Данная возможность контролируется директивой url\_fopen\_wrappers в файле настроек PHP и по умолчанию, как правило, включена. Но в версиях PHP для Windows до PHP 4.3.0 эта возможность не поддерживается совсем, вне зависимости от url\_fopen\_wrappers.

**Замечание**. include() – это специальная языковая конструкция, поэтому при использовании внутри условных блоков ее нужно заключать в фигурные скобки.

Оператор **require** – действует примерно так же, как оператор include в C++.

Основное отличие require и include заключается в том, как они реагируют на возникновение ошибки. Include выдает предупреждение, но работа скрипта продолжается. Ошибка в require вызывает фатальную ошибку работы скрипта и прекращает его выполнение.

**Замечание**. Условные операторы и циклы на require() не влияют.

**Замечание**. require, как и include, при использовании внутри условных блоков нужно заключать в фигурные скобки.

# ЛЕКЦИЯ 4. ФУНКЦИИ, ОПРЕДЕЛЯЕМЫЕ ПОЛЬЗОВАТЕЛЕМ

## 4.1. Понятие функции

Под функцией подразумевается отображение множества ее аргументов на множество ее значений. Пример:

<?php

function fact($n){

if ($n==0) return 1;

else return $fact = $n \* fact($n-1);

}

echo fact(3);

echo fact(50);

?>

Структура функции может быть задана следующим образом:

function <Имя\_функции> ([<параметр1>, <параметр2>,...<параметрN>]){

<Блок\_действий>;

[return "значение, возвращаемое функцией";]

}

Имена функций нечувствительны к регистру.

В PHP3 Функцию можно вызвать после ее определения. Но уже в PHP4 такого требования нет. Исключение составляют функции, определяемые условно (внутри условных операторов или других функций).

Если функция однажды определена в программе, то переопределить или удалить ее позже нельзя. Несмотря на то, что имена функций нечувствительны к регистру, лучше вызывать функцию по тому же имени, каким она была задана в определении.

## 4.2. Аргументы функции

Это переменные или константы, посредством которых в функцию передается различная информация. Способы передачи данных в функцию:

* передача аргументов по значению (используется по умолчанию);
* передача аргументов по ссылке;
* задание значения аргументов по умолчанию.

Когда аргумент передается в функцию по значению, изменение значения аргумента внутри функции не влияет на его значение вне функции. Чтобы позволить функции изменять ее аргументы, их нужно передавать по ссылке. Для этого в определении функции перед именем аргумента следует написать знак амперсанд "&".

<?php

function add\_label(&$data\_str){

$data\_str .= "checked";

}

$str = "<input type=radio name=article";

echo $str ."<br>";

add\_label($str);

echo $str ."><br>";

?>

В функции можно определять значения аргументов, используемые по умолчанию. Само значение по умолчанию должно быть константным выражением, а не переменной и не представителем класса или вызовом другой функции.

<?php

function Message($sign="Оргкомитет."){

echo "Следующее собрание состоится завтра.<br>";

echo $sign . "<br>";

}

Message();

Message("С уважением, Вася.");

?>

**Замечание**. Если у функции несколько параметров, то те аргументы, для которых задаются значения по умолчанию, должны быть записаны после всех остальных аргументов в определении функции. В противном случае появится ошибка, если эти аргументы будут опущены при вызове функции.

## 4.3. Списки аргументов переменной длины

В PHP4 можно создавать функции с переменным числом аргументов – создается функция, не зная заранее, со сколькими аргументами ее вызовут. Для этого используются встроенные функции **func\_num\_args(),** **func\_get\_arg(),** **func\_get\_args()**.

**func\_num\_args()** – возвращает число аргументов, переданных в текущую функцию. Может использоваться только внутри определения пользовательской функции. Если она появится вне функции, то интерпретатор выдаст предупреждение.

<?php

function DataCheck(){

$n = func\_num\_args();

echo "Число аргументов функции $n";

}

DataCheck();

DataCheck(1,2,3);

?>

**func\_get\_arg()** возвращает аргумент из списка переданных в функцию аргументов, порядковый номер которого задан параметром <номер\_аргумента>. Аргументы функции считаются начиная с нуля. Как и func\_num\_args(), эта функция может использоваться только внутри определения какой-либо функции. <Номер\_аргумента> не может превышать число аргументов, переданных в функцию. Иначе будет сгенерировано предупреждение, и функция func\_get\_arg() возвратит False.

<?

function DataCheck(){

$check = true;

$n = func\_num\_args();

if ($n>=1)

if (!is\_int(func\_get\_arg(0))) $check = false;

if ($n>=2)

if (!is\_string(func\_get\_arg(1))) $check = false;

return $check;

}

if (DataCheck(123,"text"))

echo "Проверка прошла успешно<br>";

else echo "Данные не удовлетворяют условиям<br>";

if (DataCheck(324))

echo "Проверка прошла успешно<br>";

else echo "Данные не удовлетворяют условиям<br>";

?>

**func\_get\_args()** возвращает массив, состоящий из списка аргументов, переданных функции. Каждый элемент массива соответствует аргументу, переданному функции. Если функция используется вне определения пользовательской функции, то генерируется предупреждение.

<?

function DataCheck(){

$check = true;

$n = func\_num\_args();

$args = func\_get\_args();

for ($i=0;$i<$n;$i++){

$v = $args[$i];

if ($i % 2 == 0){

if (!is\_int($v)) $check = false;

}

}

return $check;

}

if (DataCheck("text", 324)) echo "Проверка прошла успешно<br>";

else echo "Данные не удовлетворяют условиям<br>";

?>

В PHP3 для того, чтобы добиться подобного эффекта, можно использовать в качестве аргумента функции массив.

<?

function DataCheck($params){

$check =true;

$n = count($params);

for ($i=0;$i<$n;$i++){

$v = $params[$i];

if ($i % 2 != 0){

if (!is\_int($v)) $check = false;

}

}

return $check;

}

if (DataCheck("text", 324)) echo "Проверка прошла успешно<br>";

else echo "Данные не удовлетворяют условиям<br>";

?>

## 4.4. Использование переменных внутри функции

### 4.4.1. Глобальные переменные

Глобальные переменные – переменные, заданные вне подпрограмм, но которые нужно использовать внутри той или иной функции. Для этого в теле функции следует перечислить их имена после ключевого слова global.

Когда переменная объявляется как глобальная, фактически создается ссылка на глобальную переменную. Поэтому такая запись эквивалентна следующей (массив $GLOBALS содержит все переменные, глобальные относительно текущей области видимости):

$var1 = &$GLOBALS["var1"];

$var2 = &$GLOBALS["var2"];

Это значит, например, что удаление переменной $var1 не удаляет глобальной переменной $GLOBALS["var1"].

### 4.4.2. Статические переменные

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:

static $var1, $var2;

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

<?php

function Test\_s(){

static $a = 1;

$a = $a\*2;

echo $a;

}

Test\_s();// выведет 2

echo $a;// ничего не выведет, так как $a доступна только внутри функции

Test\_s();// внутри функции $a=2, поэтому результатом работы функции будет 4

?>

## 4.5. Оператор RETURN

Любая функция может возвращать как результат своей работы какое-нибудь значение с помощью оператора return. Возвращаемое значение может быть любого типа, включая списки и объекты. Когда интерпретатор встречает команду return в теле функции, он немедленно прекращает ее исполнение и переходит на ту строку, из которой была вызвана функция.

<?php

function Full\_age($b\_day, $b\_month, $b\_year)

{

$y = date("Y");

$m = intval(date("m"));

$d = intval(date("d"));

$b\_month = intval($b\_month);

$b\_day = intval($b\_day);

$b\_year = intval($b\_year);

$day = ($b\_day > $d ? 30 - $b\_day + $d : $d - $b\_day);

$tmpMonth = ($b\_day > $d ? -1 : 0);

$month = ($b\_month > $m + $tmpMonth ? 12 - $b\_month +

$tmpMonth + $m : $m+$tmpMonth - $b\_month);

$tmpYear = ($b\_month > $m + $tmpMonth ? -1 : 0);

if ($b\_year > $y + $tmpYear)

{

$year = 0; $month = 0; $day = 0;

}

else

{

$year = $y + $tmpYear - $b\_year;

}

return array ($day, $month, $year);

}

$age = Full\_age("29","06","1986");

echo "Вам $age[2] лет, $age[1] месяцев и $age[0] дней";

?>

Когда функция возвращает несколько значений для их обработки в программе, удобно использовать языковую конструкцию list (), которая позволяет одним действием присвоить значения сразу нескольким переменным.

list($day, $month, $year) = Full\_age("07", "08", "1974");

Конструкцию list () можно использовать для присвоения переменным значений элементов любого массива.

<?

$arr = array("first", "second");

list($a, $b) = $arr;

echo $a, " ", $b;

?>

## 4.6. Возвращение ссылки

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 амперсанд (&), и каждый раз при вызове функции перед ее именем тоже писать амперсанд (&). Обычно функция возвращает ссылку на какую-либо глобальную переменную (или ее часть – ссылку на элемент глобального массива), ссылку на статическую переменную (или ее часть) или ссылку на один из аргументов, если он был также передан по ссылке.

<?

$a = 3; $b = 2;

function & ref($par){

global $a, $b;

if ($par % 2 == 0) return $b;

else return $a;

}

$var =& ref(4);

echo $var, " и ", $b,"<br>";

$b = 10;

echo $var, " и ", $b,"<br>";

?>

## 4.7. Переменные функции

PHP поддерживает концепцию переменных функций. Это значит, что если имя переменной заканчивается круглыми скобками, то PHP ищет функцию с таким же именем значения и пытается ее выполнить.

<?

function Add\_sign($string, $sign="С уважением, Петр"){

echo $string ." ".$sign;

}

function Show\_text(){

echo "Отправить сообщение по почте<br>";

}

$func = "Show\_text";

$func();// это вызовет функцию Show\_text

$func = "Add\_sign";

$func("Привет всем <br>");// вызов функции Add\_sign с пар-ром "Привет всем"

?>

**Замечание**. Такие функции, как echo(), print(), unset(), include() и т.п. нельзя использовать в качестве переменных функций.