**Практическое занятие №21 Строки, Логический тип данных, приведение типов**

**Задание №1**

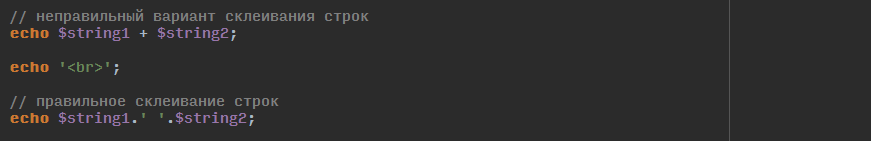
Проверьте запущен ли у Вас Open Server. Папка с материалами урока (php21.loc) должна быть размещена в директории /domains сервера. Заходим на <http://php21.loc> и начинаем урок.

**Задание №2**

Строка в PHP - это набор символов любой длины. Строки могут быть заключены в одинарные (') или двойные кавычки ("). Рассмотрим и протестируем код создания строк в PHP:



В отличии от JavaScript в PHP для конкатенации (склеивания) строк используется оператор «.», пример склеивания строк:



У оператора конкатенации есть особенности использования, так есть он находиться между двумя числами, то он будет разделителем целой и дробной части:

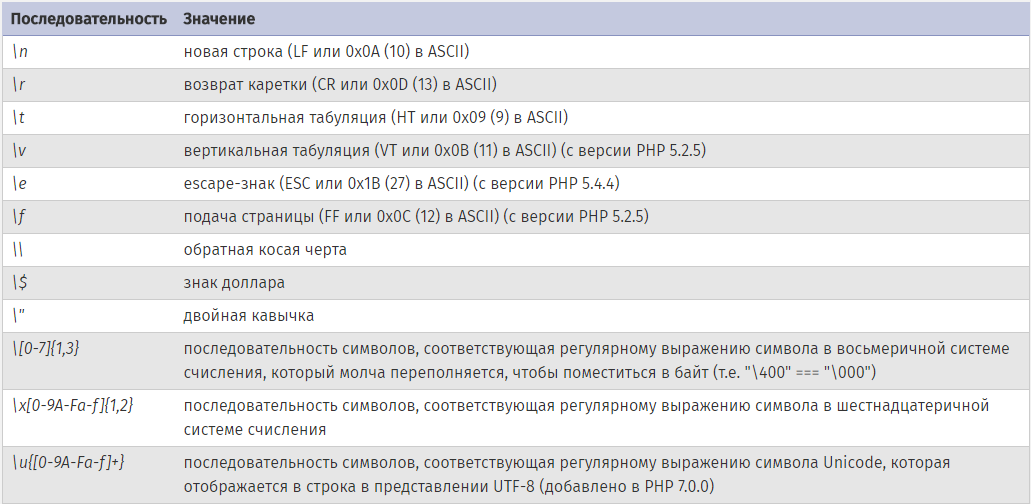


В последнем примере используется встроенная функция **var\_dump**, которая выводит значение и тип, чтобы показать Вам, что в результате мы получили строковый тип.

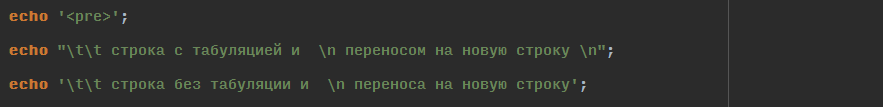
**Задание №3**

Управляющие последовательности в PHP очень похоже на JavaScript. В PHP есть особенность - строки в двойных кавычках «"» функционируют так же как и в JavaScript, строки в «'» выводятся «как есть».

Ниже представляем Вам таблицу управляющих последовательностей PHP:



Рассмотрим и протестируем код:



Единственным исключением экранирования строки в одинарных кавычках является сама одинарная кавычка:



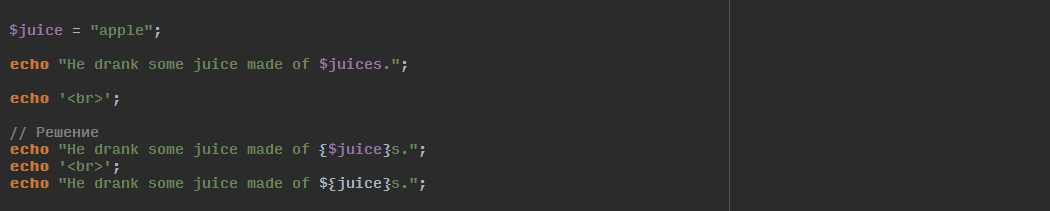
**Задание №4**

В PHP есть возможность, которой нет в JavaScript, это возможность подставлять переменные непосредственно в строку, разберем и протестируем пример ниже:



Какие выводы мы можем сделать?

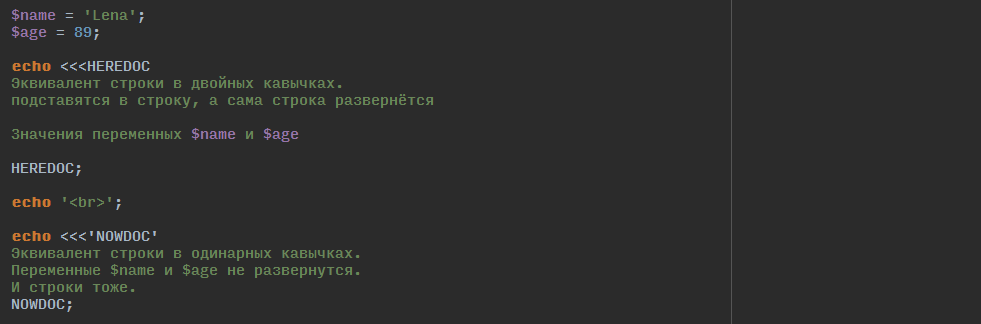
Возникает проблема, особенно характерна для Английского языка, когда нужно явно указать в тексте строки где начинается и заканчивается имя переменной. Рассмотрим и протестируем следующий пример:



Этот приём называется выделение (экранирование) переменных в строке.

**Задание №5**

Так же в PHP существует альтернативный синтаксис строк, он используется довольно редко, но его можно иногда встретить. Его могут использовать, например, для того что бы вывести большие куски текста в консоль операционной системы при вызове php скрипта. Рассмотри и протестируем следующий код:



**Задание №6**

Получить длину строки (кол-во символов) в PHP можно с помощью встроенной функции **strlen,** рассмотрим и протестируем пример кода:



Так же PHP позволяет получить произвольный доступ к символу в строке. Для этого используется следующий синтаксис:



Теперь самостоятельно напишите код, который в строке *$string* заменяет последний символ на «!».

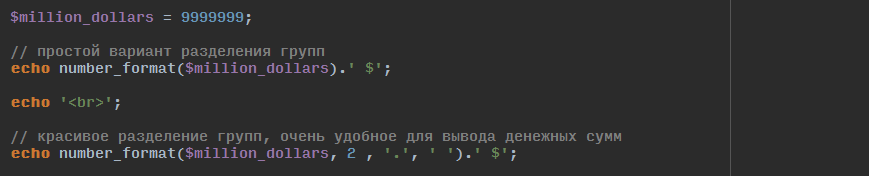
**Задание №7**

В PHP встроено множество функций для работы со строками. В этом задании мы разберем некоторые из них.

**number\_format** — форматирует число с разделением групп.

Подробнее <http://php.net/manual/ru/function.number-format.php> .

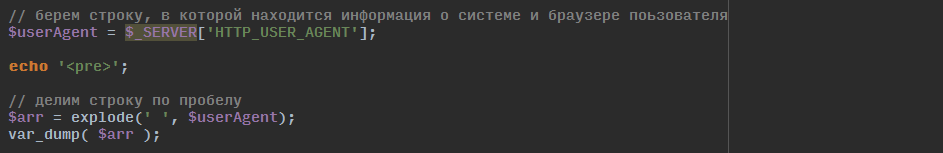
Рассмотрите и протестируйте пример кода:



**explode** — Разбивает строку с помощью разделителя (указанного символа). Возвращает массив PHP.

Подробнее тут <http://php.net/manual/ru/function.explode.php> .

Рассмотрите и протестируйте пример кода:



Обратная функция **implode** – она склеивает массив в строку:



**trim** – удаляет пробелы (или другие пробельные символы) из начала и конца строки.

Подробнее <http://php.net/manual/ru/function.trim.php> .

Рассмотрите и протестируйте пример кода:



**str\_replace** — Заменяет кусок строки на другой.

Подробнее <http://php.net/manual/ru/function.str-replace.php> .

Рассмотрите и протестируйте пример кода:



**Задание №8**

Логический тип данных в PHP такой же как и в JavaScript. Логические операторы автоматически приводят свои операнды в логический тип (об особенностях приведения типов в PHP мы поговорим позже).

Логические операторы PHP:



Обратите внимание что, логические операторы AND и OR в PHP имеют два варианта использования AND (&&) и OR (||). Оба варианта не одинаковы - они имеют разный приоритет выполнения (у AND и OR он ниже).

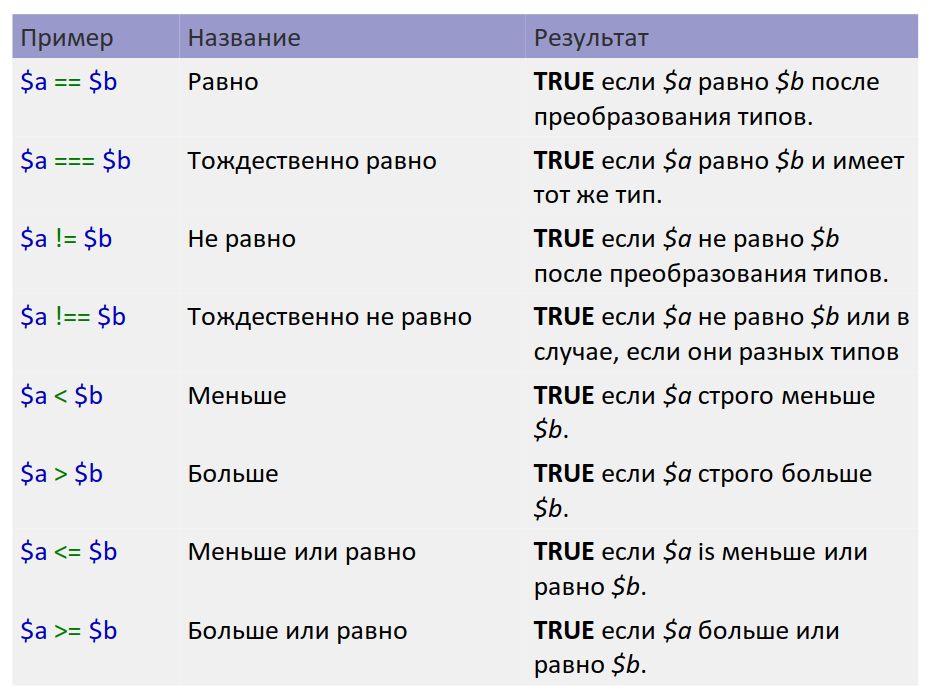
Приоритет выполнения операторов можно посмотреть на странице тут <http://php.net/manual/ru/language.operators.precedence.php>

Рассмотрим и протестируем следующий код:



Почему мы получили такие результаты?

Операторы сравнения служат для сравнения значений, операторы сравнения в PHP представлены в таблице ниже:

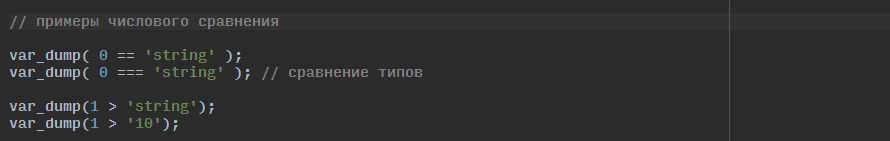
Операторы сравнения: 

Следует знать несколько особенностей сравнения в PHP.

Строки между собой сравниваются через механизм «лексического сравнения». PHP в этой роли используется алгоритм **Natural Order String Comparison** ( http://sourcefrog.net/projects/natsort/ ). Рассмотрим пример сравнения строк:



Если один из операндов число, то второй будет приведен к числу, протестируйте следующий пример:



Если один из операндов логическое значение, то второй будет приведен в логическое значение. Пример:



Детальнее о сравнении массивов и объектов можно посмотреть тут <http://php.net/manual/ru/language.operators.comparison.php>

**Задание №9**

Специальное значение NULL представляет собой переменную без значения. NULL - это единственно возможное значение типа null. Переменная считается null, если:

* ей была присвоена константа NULL.
* ей еще не было присвоено никакого значения.
* она была удалена с помощью unset().

Проверьте следующий код, посмотите с помощью var\_dump какие значения у переменных получились.

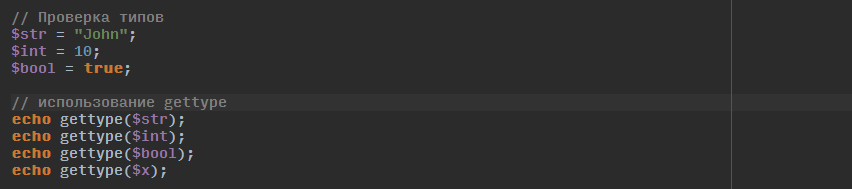


Встроенная функция unset() удаляет переменную.

**Задание №10**

Перед тем как рассмотреть и протестировать как PHP преобразует типы данных, мы разберём несколько важных встроенных функций для работы с типами данных:

**gettype($var)** – возвращает строку с названием типа переменной $var. Рассмотрите и протестируйте следующий код:



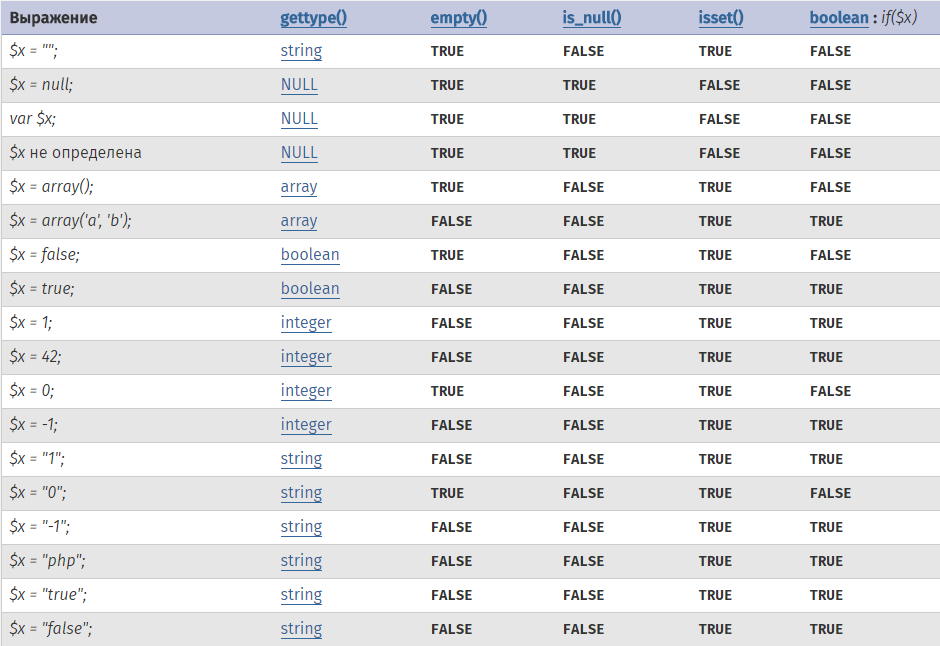
Так же существуют отдельные встроенные функции для проверки отдельных типов:



**isset($value)** - определяет, была ли установлена переменная значением отличным от NULL. Фактически это проверка на существование значения.

**empty($value)** - проверяет, пуста ли переменная.

Результаты функций связанных с типами представлены в таблице ниже:



**Задание №11**

В PHP допускаются следующие приведения типов:

* **(int), (integer)** - приведение к integer
* **(bool), (boolean)** - приведение к boolean
* **(float), (double), (real)** - приведение к float
* **(string)** - приведение к string
* **(array)** - приведение к array
* **(object)** - приведение к object
* **(unset)** - приведение к NULL (PHP 5+)

В этом задании Вы должны самостоятельно скрипт, который будет в табличной форме выводить информацию о переменной, включая:

* Имя переменной
* Значение
* Текущий тип
* результат empty для переменной
* результат is\_null для переменной
* результат isset для переменной
* вывести значения, которые мы получаем при приведении переменной к следующим типам: string, int, float, array, object, null, bool.

**Задание №11**

PHP позволяет очень гибко настроить оповещение об ошибках. В PHP все ошибки разбиты на различные уровни, каждому уровню соответствует определённая константа как идентификатор этого уровня. Сейчас мы разберём как некоторые из них.

**E\_PARSE** – ошибки на этапе первичного анализа кода, например классическая ошибка – пропущенный разделитель инструкций «;». Введите и запустите данный код:



Какое сообщение об ошибке мы получили? Что произошло с точки зрения PHP?

**E\_ERROR** – фатальные ошибки, которые прекращают выполнение сценария PHP, пример такой ошибки – вызов неопределенной функции:



**E\_WARNING** - предупреждения времени выполнения (не фатальные ошибки). Выполнение скрипта в таком случае не прекращается. Примером такой ошибки может служить попытка открыть несуществующий файл:



Главное, что Вы должны взять из этого примера это то, что выполнение программы после ошибки **E\_WARNING** продолжается и это может привести к «каскаду ошибок». Если такое уже произошло, то нужно всегда смотреть на первую ошибку.

**E\_NOTICE** – это что-то вроде извещение, PHP кажется, что Вы делаете что-то не так. Например, обращаетесь к не существующему элементу массива:



Сейчас на экране Вы возможно не увидите предупреждение об этой ошибке, потому что очень часто это оповещение отключено.

**E\_DEPRECATED** – ошибки, связанные с использование устаревших функций.

Полный список констант, описывающий уровни ошибок можно найти тут <http://php.net/manual/ru/errorfunc.constants.php>

Первоначально задать как будут отображаться ошибки можно в php.ini за это отвечают следующие директивы:

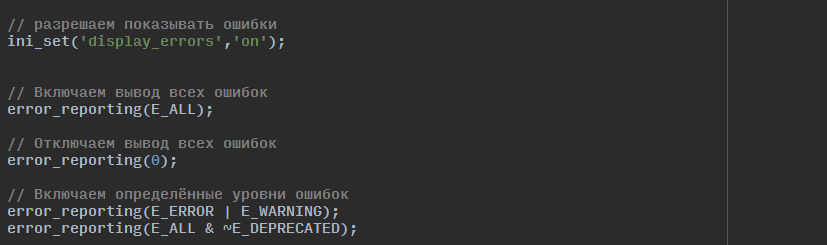
; on или off. Если стоит значение off то никакие ошибки не будут выводиться

display\_errors = on

; показывать все ошибки кроме E\_NOTICE

error\_reporting = E\_ALL & ~E\_NOTICE

Но если у Вас на проекте нет доступа к настройкам php.ini , то не отчаивайтесь есть замечательные встроенные функции, которые могут «на лету» изменить настройки PHP:

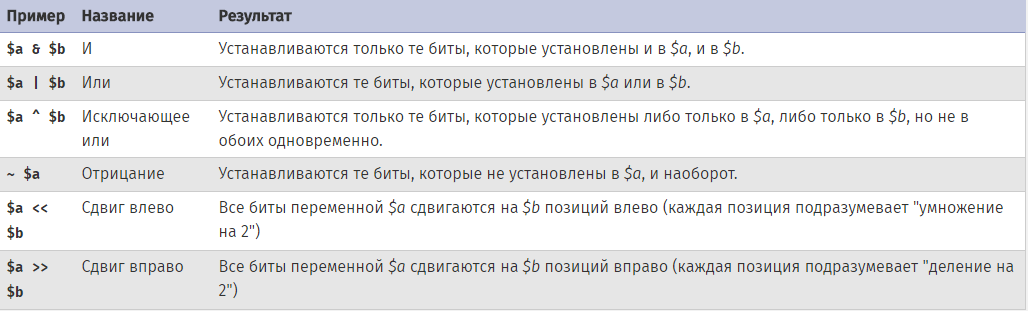


**ini\_set** - устанавливает значения настройки конфигурации в скрипте. Следует помнить что далеко не все настройки можно менять подобным образом.

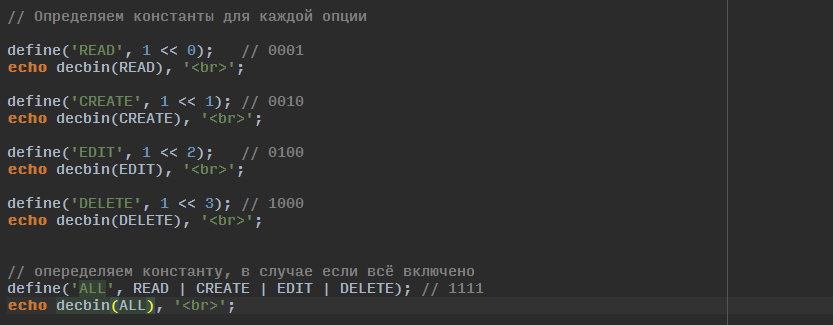
**error\_reporting** – задаёт какие ошибки следует отображать.

**Задание №12**

Последнее на сегодня задание — это побитовые операции в PHP. Побитовые операторы позволяют считывать и устанавливать конкретные биты целых чисел. Побитовые операции сведены в таблицу ниже:



Рассмотрим пример использования побитовых операций в PHP. Предположим, что нам нужно сгенерировать 4 бита данных, которые определяют модификаторы доступа к директории в следующем порядке: чтение, создание, изменение, удаление. Рассмотрим и протестируем пример кода, как это можно реализовать через побитовые операции:



Встроенная функция **decbin** переводит число из десятеричного в двоичный формат. Таким способом через созданные константы удобно управлять правами доступа. Чтобы комбинировать несколько прав используем побитовый оператор ИЛИ (|).

Другой пример использования побитовых операторов Вы уже встречали на этом уроке:



На самом деле функция error\_reporting принимает целое число, затем в двоичном формате, считывая биты, и по ним определяет какие настройки были ей переданы.

**Домашнее задание**

Ознакомитесь со встроенными функциями для обработки строк <http://php.net/manual/ru/ref.strings.php>