**PHP по мотивам лекций Специалист PHP Борисова**

**Настройка**

<?php

phpinfo(); // вывод инфы о текущей конфигурации

?>

В открывшемся окне Loaded Configuration File – указывает на текущий php.ini файл. Если меняем настройки в php.ini и они не меняются – надо посмотреть, куда ссылается эта строка

**ini\_get("post\_max\_size")** получить значение директивы из php.ini

**ОСНОВЫ PHP**

<?php …code… ?> - рекомендуемый

<? ..code.. ?> - не рекомендуется

<script language=”php”> </script> - устаревший

<% code %> - устаревший, asp\_tags (php.ini)

Ту или иную опцию можно включить локально в какой-то папке, прописав в **.htaccess** команду, например так включается asp\_tags для локального использования:

php\_flag asp\_tags on

Что будет видно на страничке с выводом функции **phpinfo():**

C:\Users\User-HPi3A\Downloads\asp_tags.png

Закрывающая **точка с запятой** в конце инструкции - обязательна

**echo** "hello", "22", 111 -может выводить конкатенацию строк

**print** "hello" - выводит только одиночную строку

**Переменные** создаются через знак доллара $. Регистрозависимы

<? echo $name; // первый вариант вывода ?>

<?= $name // второй вариант вывода, короткий ?>

**unset($name) -**  удаление переменной $name

**iconv("windows-1251", "UTF-8", $name)** - перевод кодировки из win в UTF

**Константы (запись только для чтения)**

Контанты регистрозависимы. Что держать в константах:

* имена файлов

define("USER\_NAME", "Вася") // запись значения константы USER\_NAME

const TWO\_HUNDRED = 200 // тоже самое, как и запись сверху. До PHP 5.6 этой записи нельзя присваивать результат выражения, например так: const THREE = 2 + 1;

**Ссылки в php**

$x = 10;

$y = &$x; // сослались на переменную, теперь у них одна область памяти

$y = 20;

echo $x; // 20;

echo $y; // 20;

**Переменные переменных**

$x = 'name';

$$x = "Вася"

echo "Привет, ", $name; // Привет , Вася

**Экранирование переменных**

$juice = "apple"

echo "He drank some $juice juice." // he drank some apple juice

echo "He drank some juice made of {$juice}s" //экранировали для приставки s

**Ошибки в PHP**

Ошибки имеют несколько уровней:

* Parse error - ошибка парсинга. Парсит код и проверяет на наличие синтаксических ошибок, если ошибок нет, начинает выполнение
* Fatal error - ошибка во время исполнения кода. После этой ошибки код дальше исполняться не будет
* Warning - код исполнятся дальше будет, но ошибки могут лезть где-то дальше
* Notice - легкое предупреждение
* Deprecated - устаревший метод

За вывод ошибок в php.ini отвечает две команды:

display\_errors = on // вкл\выкл вывод ошибок

error\_reporting = E\_ALL & ~E\_NOTICE // показывает все ошибки кроме E\_NOTICE

error\_reporting(E\_ALL) - показывает все ошибки

**ТИПЫ ДАННЫХ**

**Строки**

**Разница между двойными (" ") и одинарными (' ') символами:**

Одинарные кавычки не интерпретируют внутренние команды: переменные не подставляют значение, и не работают символы переноса строки, например

echo "Это будет выведенно в \n две строчки"

echo 'А это не будет выведенно в \n две строчки'

$name = "John";

echo 'Hello $name'; // Hello $name , не работает с одинарными кавычками

echo "Hello $name"; // Hello John, вывод переменных прямо в строке

/n переносит строку только в исходном коде (код страницы), для переноса в браузере следует использовать <br>

**<<<HEREDOC** - аналог pre в HTML. Строка парсится в таком виде, в котором забирается из окна ввода:

echo <<<ANY\_NAME

Значения переменных $name и $age

подставятся в строку, а сама строка перенесется

ANY\_NAME;

Правила использования:

* после <<<HEREDOC, в той же строке не должно быть других символов, даже пробелов.
* перед после закрывающий HEREDOC; тоже не должно быть ничего

Ошибка с этими идентификаторами ведет к трудноуловимой ошибки: парсер говорит о то, что ошибка в закрытии блока где-то есть, но не указывает, где именно. Потому перед началом разработки в блоках, лучше все блоки сразу правильно открыть и правильно закрыть, убедится, что нету ошибок, и только потом писать код внутри.

**Работа со строками**

$str = 'This is some string'

$first = $str{0} // получить первый символ в строке

echo $str[0] // тоже первый элемент строки, можно брать как через {}, так и []

$last = $str{mb\_strlen($str) -1} // получить последний символ

**strlen($strName)** - получить длину строки. Отсчет идет в байтах, а не сиволах

**mb\_strlen($strName)** - получает длину строки в байтах и переводит их в правильное значение

$name = "Вася"

strlen($name) // 8, каждый символ в рус. идет по 2 байта

mb\_strlen($name) // 4

**Конкатенация строк** - в PHP это точка . либо написание переменных в одной строке:

$a = "Hello";

$b = $a . "World"; // "Hello World"

$d = "$a $b"; // "Hello World

Константы не конкатенируются, как в последнем примере;

**strtoupper($x)** - перевести строку в верхний регистр

**Числа**

Тип integer = все целочислненные значения

Тип float (double) = все числа с плавающей запятой

Базовые операции такие же, как в JS

echo "10" + 10 // 20, привел к типу "число"

echo "string" + 10 // 10, не нашел число в строке

echo "15string" + 10 // 25, прочел число слева направо, пока не уперся в символ

**Булевы значения**

Булевые true/false в PHP переводятся в число 1\0, причем false переводится в 0 только при приведению к числу, иначе будет просто пустая строка:

$x = true;

$y = false;

echo $x; // 1

echo $y; // "" - пустая строка

echo $y + 10; // 10 , false было переведено в 0 и суммировано с 10

При проверке на **true \ false** к false приводится в следующих слуачаях:

$str = ""; // false - пустая строка

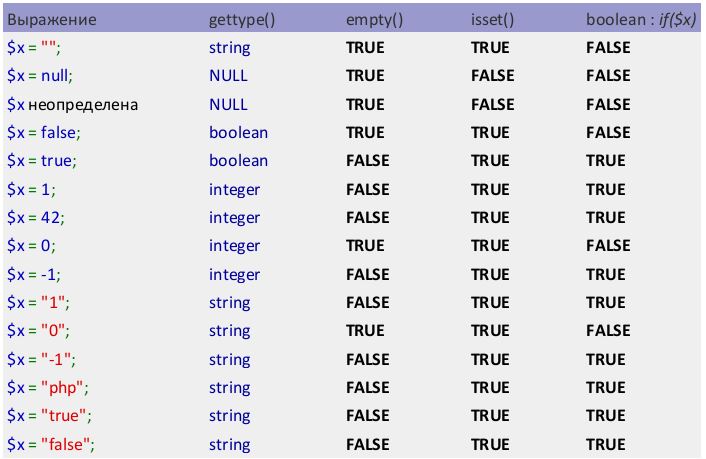
$str = "0"; // false - строка с 0 внутри

$str = 0; // false - число 0

$arr = []; // false - пустой массив

**isset($x)** - выдает false только в двух слуаях: переменная не определена или переменная = null

**empty($x)** - проверяет, "заполненна" ли переменная. Полная противоположность проверке логическим оператором **if($x)**

****

**Специальный тип**

null появляется в 3-х случаях:

* переменной не было присовено значения:

echo $x;

* переменной специально было присоваенно значение null:

$x = null;

echo $x;

* переменная была удалена:

$x = 10;

unset($x);

echo $x;

**Проверка типов**

**gettype($variable)** - показывает тип, возвращает тип в строке

**is\_type** - проверяет нужный тип, возвращает 1 \ 0

$str = "John";

$int = 10;

$bool = true

echo gettype($str); // "string"

echo gettype($int); // "integer"

echo gettype($bool); // "boolean"

echo gettype($x); // "NULL"

echo is\_string($str); // 1;

echo is\_integer($int); // 1;

echo is\_boolean($bool); // 1;

echo is\_null($x); // "";

echo is\_array($arr); //

**Преобразование типов**

Можно окончательно преобразовать тип через **settype(type):**

$x = "10 apples";

settype($x, "integer"); // 10, слово apples потерялось навсегда

Использование settype не рекомендуется

**(int)$variable**, **(string)$variable**, **(bool)$variable**, **(double)$variable** - приведение к типу копии переменной

echo (int)$x \* 10 // 100, было произведено преобразование копии переменной

echo $x; // 10 apples - сама переменная не изменена

**==** проверка с преобразованием к единому типу

**===** проверка без преобразования типов

echo 1 == "1" // true

echo 1 === "1" // false

**Логические операторы**

Такие же, как в JS, пара нюансов:

**&& и || = and и or**, только у and и or приоритет ниже

if ( ... ){

...

} elseif ( ... ) {

...

} else { }

//альтернатива, двое точие (:) как открывающая скобка и endif как конечная закрывающая

if ( ... ):

...

elseif ( ... ) :

...

else :

...

endif

**Массивы**

$arr = []; // обьявление массива. Пустой массив - false

$arr[] = "John" // пушим в массив значение. Добавляет к максимальному индексу +1

$arr = [10=>"John", true, 5=>"lol"];// дырки до позиции 5, на 5 lol, а потом отсчет с 10

ehco $arr[10] // выведет значение под номером 10

**count($arr)** - посчитать длинну массива

**var\_dump($arr)** - вывести подробное описание элементов массива, с типами

**print\_r($arr)** - вывести элемент массива. Не видно типов элементов.

**Ассоциативный массив -** в JS идет как обьект. В PHP это массив в именованными ключами. Доступ к значениям можно получить только по именнованому ключу.

$arr = [

"gender" => "male", //доступ только так: echo $arr["gender"];

"name"=>"John",

"age" => 23

]

$arr[] = "lol"; // добавит в индекс 0, т.к. до этого были именнованые ключи

**Многомерный массив.**  Массив так же может содержать вложенные массивы

$arr = [];

$arr[0] = [

"login" => "john"

"pass" => "1234"

];

$arr[1] = [

"login" => "mike"

"pass" => "5678"

];

echo arr[1]["login"]; // mike

**Циклы**

**for( Часть А ; Часть B ; Часть C );**

* **Часть А -** инструкция, которая будет выполнена с самого начала. Там может быть выражение, функция, присваивание и т.д., что угодно.
* **Часть B -** встроенный if\else, который будет выполнять инструкции внутри for, если вычисленное значение, приведенное к булевому типу дает true
* **Часть C -** исполняется каждый раз после отработки тела цикла. Часть А и часть C похожи, разница только в том, что C исполняется в конце каждый раз после отработки тела цикла.

Если в условии цикла используется вычисленная длинна (массива, строки и т.д.), лучше предварительно ее занести в отдельную переменную, что бы не гонять вызов функции по несколько раз:

$greeting = ["H", "E", "L", "L", "O"];   
  
$count = count($greeting);  // сохранили длину в отдельной переменной  
for($i = 0; $i < $count; $i++){  //теперь количество итераций не зависит от длины  
 echo $greeting[$i];   
 $greeting[] = "new";  // добавляем новый элемент массива, цикл уже не бесконечный  
}

**for** используется чаще, когда есть четко обозначенное начало и конец

**break** - прерывает цикл. Так же можно указать число, которое служит как указатель на то, сколько уровней вложенности должен прервать break

$i = 1; $j = 1;

while ($j <= 10) {

while ($i <= 10) {

echo $i++;

if($i ===5)

break 2; // прервет выполнение текущего цикла и одного цикла сверху

}

$j++;

}

**continue** - прерывает текущую итерацию, но не цикл. Число после continue тоже указывает уровни вложености

**foreach($array as $key => $value){}** цикл для перебора массива

* **$array** - массив, который будем перебирать
* **$key** - ключ каждой ячейки, присваемый при переборе
* **$value** - значение каждой ячейки, присваемой при переборе

$user = [ // дан ассоциативный массив

"name" => "John",

"login" => "root",

"pass" => ""

]

foreach($user as $key => $val){

echo "$key: $val"; // выводим пары ключ значение каждого элемента

}

Цикл берет копию элемента, поэтому изменение внутри foreach не повлияет на оригинальный массив. Что бы изменить оригинальный массив, надо использовать ссылку на переменную **&$var**

$nums = [1, 2, 3, 4, 5];

foreach($nums as &$val){ //ключ опускаем, присваиваются сразу значения

$val \*= 10; // изменяется оригинальный массив благодаря ссылке в условии

}

unset($val) // поскольку переменная в памяти осталась, во избежание ошибок удаляем ее

Подставляем динамически меню через foreach:

$mainMenu = [ // дано меню со ссылкой в <a> следющим за <a> текстом

["href" => 'index.php', "header" => "Домой"],

["href" => 'about.php', "header" => "О нас"],

];

echo "<ul>";

foreach($mainMenu as $val){

// можно было конкатерировать строки через . а можно и выделить переменные через {}

echo "<li><a href='{$val["href"]}'>{$val["header"]}</a></li>";

}

echo "</ul>";

Стандартная ошибка foreach - **Invalid argument**. Это значит, что первым аргументов в него передали НЕ массив, надо сразу проверить тип аргумента через var\_dump например.

**Функции**

Функции лучше делать много мелких, чем одну большую. Все что повторяется больше одного раза, надо вынести в функцию.

Имя функции в PHP регистронезависимо

function say($name, $h = 3){ // второй аргумент - параметр по умолчанию

echo "<h$h> Hello, $name!<h$h>";

}

say("John", 1); // выведется прл тэгом <h1>

say("Mike"); // вызывается без второго арумента, выведется прл тэгом <h3>

Сначала надо передавать обязательные параметры, а после - необязательные. Передать дырку в аргументах при вызове функции say( , 2) невозможно.

**Возвращаемое значение**

Функция всегда возвращает значение. return если не задан явно, возвращает NULL

return возвращает вычисленные значения. Считается хорошим тоном, если функция возвращает значения одного типа, что бы не нужно было в последствии еще проверять тип: число, строку, массив или булево значение

Если нужно вернуть несколько значений, можно передать их в массив, который будет возвращаться. Для извлечения из возвращаемого массива есть несколько способов

function nums() {

return [1, 2, 3];

}

// list присваивает указанным переменным соответствущие значения внутри массива

list($one, $two, $three) = nums(); // \* первый способ

echo $two; // 2

$arr = nums(); // \* второй способ

echo $arr[2] // 2

$two = nums()[1] // 2 \* третий способ - разименование массива

Поскольку лучше, что бы функция возвращала значения одного типа, вместо возврата массива со значениями, можно сделать ссылку на внешнюю переменную:

function validation( $statement, &$error ){ // сделали ссылку на внешнюю переменную

if("условие, которое показывает, что все хорошо"){

return true; //возвращает тру, если проверка пройдена без ошибок

} else {

$error = "Сообщение об ошибке"; // передаем значение сообщение об ошибке по ссылке

return false; // и к тому же возвращаем false

}

}

validation( submitForm, $err ){

if(err){ ... валидация не пройдена ...}; // достаточно проверить только error message

**Именованная функция**

Мы можем присвоить функцию переменной и в последствии вызвать ее:

function say(){...}

$strFunc = "say"; // назначили переменной strFunc функцию say

$strFunc(); // вызываем

Вызов именнованной функции можно совершить только послее ее присваивания, иначе будет ошибка:

$strFunc(); // Fatal error: Function name must be a string

$strFunc = "say";

**Аргументы функций**

Если в функцию передается много аргументов, их можно извлечь разными способами:

* **func\_num\_args() -** выводит количество переданных аргементов
* **func\_get\_args() -** получает все аргументы в виде массива
* **func\_get\_arg(2) -** получает аргумент по заданному индексу

function foo(){

echo func\_num\_args(); // 5

print\_r( func\_get\_args() ); // [1,2,3,4,5]

echo func\_get\_arg(2); // 3

}

foo(1, 2, 3, 4, 5);

**Типизация передаваемых аргументов**

Массив. Можно указать, что передаваемый аргумент должен быть только массивом:

function foo(array $arr){};

foo("sdf"); // ошибка! должен передаваться массив

foo([]); // все ок

Вызываемый аргумент

function bar(){};

function foo(callable $param, $x){

$param($x)

};

foo("sdf"); // ошибка! должен передаваться вызывамый тип - функция, класс, метод

foo("bar"); // все ок, в строке передалось имя функции

**Области видимости (scope)**

PHP хранит все глобальные переменные в глобальном **массиве $GLOBALS:**

$GLOBALS["name"] = "John"; // идентично обычному $name = "John"

Функции, как и константы, всегда глобальны, где бы не были обьявлены

**Область видимости внутри функций.** Переменные внутри функции не видны снаружи. Что бы взаимодейстовать с глобальными переменными, есть три способа:

* Использовать ключевое слово **global.**

Первый недостаток - уничтожается локальная переменная.

Второй недостаток - если функция большая, в конце становится непонятно, испльзуется локальная переменная или глобальная

Третий недостаток - это ссылка на глобальную переменную, потому удалить ее изнутри функции не получится

$name = "Mike"

function say($name){

global $name; // указываем, что далее $name - глобальная переменная

echo $name; // будет использована глобальная переменная $name="Mike"

};

say("John");

* Обьявить явно:

function say($name){

$GLOBALS["name"] = "John"

}

* Сделать ссылку на глобальную переменную через &:

$name = "Mike"

function say(&$name){

echo $name; // будеи использована глобальная переменная $name="Mike"

};

say("John"); // тут будет Fatal Error, т.к. передаем строку, а не переменную

**Видимая локальная переменная**

Иногда нужно, что бы локальная переменная сохраняла свое значение и после завершения функции. Это делается благодаря слову **static.** static сохраняется в отдельном массиве, так же, как это делается для $GLOBALS

function foo(){

static $x = 0; // обьявляем переменную и присваиваем ей значение, в отличии от global

echo $x++;

}

foo(); // 0

foo(); // 1

foo(); // 2

**Дата**

**getdate()** - возвращает массив с разными параментрами

**time()** - возвращает метку времени, с 1 января 1970г., в секундах

**mktime(hours, minutes, seconds, mouth, day, year)** - календарь, который может считать как в прошлое, так и в будущее. Все параметры необязательны. Если какой-то параметр не передан, подставляется текущий параметр.

$x = mktime(0, 0, 0, 1, 1, 2017);

print\_r(getdate($x));

**strftime("")** - можно задавать формат в роли аргументов, вторым аргументом принимает метку времени, в секундах,

**date("")** - очень похоже на strftime, только буквы передаются напрямую. Не умеет работать с локали (перевод в русскую расскладку)

echo strftime("%d - %y", 1234567890); // 14-2009

echo date("d-m-Y H:i:s", 1234567890); // 14-02-2009 02:31:30

Строка любого возвращаемого месяца идет с заглавной буквы и без склонений. Если нужно сделать правильные окончания месяцев (2 января, 30 марта и т.д.), лучшее решение - сделать массив с нужными окончаниями [1=> "января", "февраля" ...] и вытаскивать оттуда, чем править строку.

**Особенности языка PHP**

**Языковые конструкции**

Языковые конструкции в php похожи на функции, но с отличиями. Их можно вызывать без скобок.

echo не возвращает вообще ничего (не NULL!)

print возвращает 1

return

include

require

**Псевдоконстанты**  
\_\_LINE\_\_ - выводит текущий номер строки

\_\_FILE\_\_ - выводит полный физический путь к файлу

\_\_DIR\_\_ - полный физический путь к текущей директории

\_\_FUNCTION\_\_ - выводит название функции, в которой вызвана

**Суперглобальные переменные**

$\_ENV - хранится информация о операционной системе

$\_SERVER - информация о настройках сервера

$\_COOKIE - инфа о куках

$\_SESSION - работа с сеансами

$\_FILES - работа с файлами, закачиваемыми по HTTP протоколу

$\_GET

$\_POST

$\_REQUEST

**Документация**

Официальная документация - [php.net](http://php.net) => documentation

Полезные разделы - Справочник и Appendicses, Справочник функций

Директивы php.ini:

* PHP\_INI\_SYSTEM - можно менять только в конфигурационном файле, на хостинге изменить не получится
* PHP\_INI\_PERDIR - можно изменять в htaccess
* PHP\_INI\_ALL - можно изменять и в конфиге, и в htaccess. Более того, можно изменять прямо в коде php во времея выполнения, через

ini\_set( ... )

Через search можно сразу искать функцию по названию:

**count()**

int\* count ( mixed\*\* $array\_or\_countable [, int\*\*\* $mode \*\*\*\* = COUNT\_NORMAL] )

* \* - тип возвращаемого значения. В данном случае - integer
* \*\* mixed - указывает, что нет строго определенного типа. В данном случае массив или что-то, что можно подсчитать
* \*\*\* - указывает тип необязательного аргумента, тут iteger
* \*\*\*\* указывает, что тут по умолчанию присвоена какая-то переменная

**Подключение файлов**

Подключение файлов происходит через языковую конструкцию **include:**

<? include "test.php"; ?>

Если имя файла неверно будет Warning

**require -** работает так же, как **include,** но если ошибка в подключении файла, будет Fatal Error

**include\_once, require\_once -** тоже самое, но подключает только один раз. Во избежание повторения одинаковых частей кода. Никаких ошибок не выдает, просто не подключает

**Обработка ошибок**

При старте сайта в продакш ошибки отключаются во избежание взломов.

error\_reporting(0); // отключает вывод всех ошибок в головном index-файле

Вместо стандартных сообщений PHP необходимо сделать свои сообщения. Делаются они путем проверки функций и возвращением false в случае ошибки в самом начале функции или return true в самом конце в случае успешного выполнения.

Затем в месте вызова функции ставится проверка: если функция возвращает false, выводим сообщение об ошибке.

function drawMenu ($menu, $horisontal = false){

if ( !is\_array($menu)) return false; // выход сразу, если не массив

//... function code ....

return true;

}

//вызов функции в нужном месте html-файла

<?php if(!drawMenu($mainMenu)) echo "ERR\_DRAW\_MENU"; ?> // вызов константы с сообщением

**Перехват ошибок.** Можно сделать отдельную функцию, которая будет перехватывать ошибки

function myError($no, $msg, $file, $line){

}

$no - все ошибки имеют свой номер, сюда он будет передаваться

$msg - сообщение об ошибке (invalid, argument и т.д.)

$file - файл, в котором произошла ошибка

$line - в какой строке произошла ошибка

Что бы не выводить встроенные сообщения PHP об ошибке, а только мои, дано 3 константы:

E\_USER\_ERROR, E\_USER\_WARNING, E\_USER\_NOTICE

Передавая их в качестве значения функции-обработчику, PHP будет знать, что произошла именно пользовательская ошибка:

function myError($no, $msg, $file, $line){

$dt = date("d-m-Y H:i:s"); //создаем время ошибки

$str = "[$dt] - $msg in $file:$line\n"; // записываем время в удобночитаемый вид

switch ($no) { // если переданый номер - пользовательская ошибка

case E\_USER\_ERROR:

case E\_USER\_WARNING:

case E\_USER\_NOTICE:

echo $msg; // выводим сообщение на сайт

}

// записываем ошибку в error.log, как свои, так и PHP

error\_log("$str", 3, "error.log");

}

**error\_log("message", "destanation", file) -** создает файл с ошибками в той директории, где была вызвана функция

message - сообщение, которое будет записано

destanation - куда будет отправлено. 0 - системный регистр PHP 1 - на почту, 3 - записан в файл

file - как будет называться файл для записи

**set\_error\_handler("myError") -** назначает мой обработчик ошибок вместо встроенного

set\_error\_handler("myError"); // пишем это вверху индексного файла

**trigger\_error("сообщение об ошибке", E\_USER\_ERROR) -** вызывает ошибку там, где она должны быть с моей точки зрения. Первым параметром передается сообщение об ошибке, вторым номер ошибки

//вызов функции в нужном месте html-файла

<?php

if(!drawMenu($mainMenu))

trigger\_error(ERR\_DRAW\_MENU, E\_USER\_ERROR) ?> // вызов константы с сообщением

**HTTP: работы с формами**

Клиент может передавать данные через два метода: **GET и POST**

**GET метод.**

Передает запрос через адресную строку браузера. В первой строке указывается метод, после идет пробел, вторым параметром - где он был вызван. Третий параметр стандартый - HTTP/1.1

Заголовок HOST является обязательным

GET /folder/index.html HTTP/1.1

Host: [www.example.com](http://www.example.com)

**Используется только для поисковых запросов (search) или если с сылкой передается какой-то параметр**

<a href="test.php?x=100">link</a>

**Передача данных из веб-формы с помощью GET.** В вебформе важно указывать:

action - что будет передаваться на страничку после нажатия submit

method - какой метод используется, GET или POST

name - данные из полей достаются по name

<form action="action.php" method="get">

<!--Логин-->

<input name="login" type="text">

<!--Пароль-->

<input name="pwd" type="password">

<input type="submit">

</form>

На выходе получается такая строка, которая конкатенируется через **&**

GET /action.php?login=John&pwd=1234 HTTP/1.1

$\_SERVER["QUERY\_STRING"] - тут сохраняется конкретный запрос из адресной строки браузера, который находится после знака вопроса (**?**)

echo $\_SERVER["QUERY\_STRING"] // login=John&pwd=1234

$\_SERVER["REQUEST\_URI"] - сохраняет полную текущую строку запроса:

echo $\_SERVER["REQUEST\_URI"] // info.php?login=John&pwd=1234

**Прием параметров переданных через GET.** После передачи запроса методом GET, данные сохраняются в суперглобальном массиве $\_GET, каждому name присваивается своя переменная

echo $\_GET["login"]; // John

echo $\_GET["pwd"]; // 1234

**Минусы передачи через GET**

* запрос может быть очень большим, и адресная строка браузера будет замусорена
* Строка запроса попадает в логи
* нельзя сделать некоторые вещи, например невозможно загрузить файл

**POST метод.**

Передает всю информацию в заголовках. Обязательными являются 3 верхних заголовка. Далее идет пустая строка, которая означает конец заголовка. Далее идет тело метода с указанной строкой запроса.

* В первой строке указан метод (POST) и действие, которое должно произойти
* Referer: поскольку запрос невозможно сделать анонимным из адресной строки, этот заголовок обязателен
* Content-Type: обычно используется значение по умолчанию
* Content-Length - длинна строки

POST /action.php HTTP/1.1

Referer: http://example.com/page-with-form.html

Content-Type: application/x-www-form-urlencoded

Content-Length: 19

login=John&pwd=1234

**Прием параметров переданных через POST.** После передачи запроса методом POST, данные сохраняются в суперглобальном массиве $\_POST, каждому name присваивается своя переменная

echo $\_POST["login"];

echo $\_POST["pwd"];

**Безопасность обработки переданных данных.**

Никогда не использовать "сырые" принятые данные, всегда пропускать через всевозможные проверки. "Проверок много не бывает":

// обрезаем, что бы удалить пробелы, если чел копировал логин и вырезаем любые теги

$name = trim( strip\_tags( $\_POST['name'] ) );

// проверяем, что бы там было число

$age = (int) $\_POST['age'];

**Проверка инициализации метода (при submit)**

if( $\_SERVER['REQUEST\_METHOD'] == 'POST' ){ ... }

Пример работающей формы - постройка таблицы умножения, пользователь задает колонки и ряды

<?php

if ($\_SERVER["REQUEST\_METHOD"] == "POST") { // запуск функции постройки таблицы

$cols = abs( (int) $\_POST["cols"] );

$rows = abs( (int) $\_POST["rows"] );

}

$cols = ($cols) ? $cols : 5;

$rows = ($rows) ? $rows : 5;

?>

<form action="<?= $\_SERVER["REQUEST\_URI"]?>" method="POST"> //возвращает текущий адрес

<label>Количество колонок: </label>

<br/>

<input name='cols' type='text' value="<?=$cols?>"/> //пользователь задает колонки

<br/>

<label>Количество строк: </label>

<br/>

<input name='rows' type='text' value="<?=$rows?>""/> // задает ряды

<br/>

<input type='submit' value='Создать'/>

</form>

<!-- Таблица -->

<?php

//таблица умножения

drawTable($cols, $rows); // сама функция была подключена ранее

?>

<!-- Таблица -->