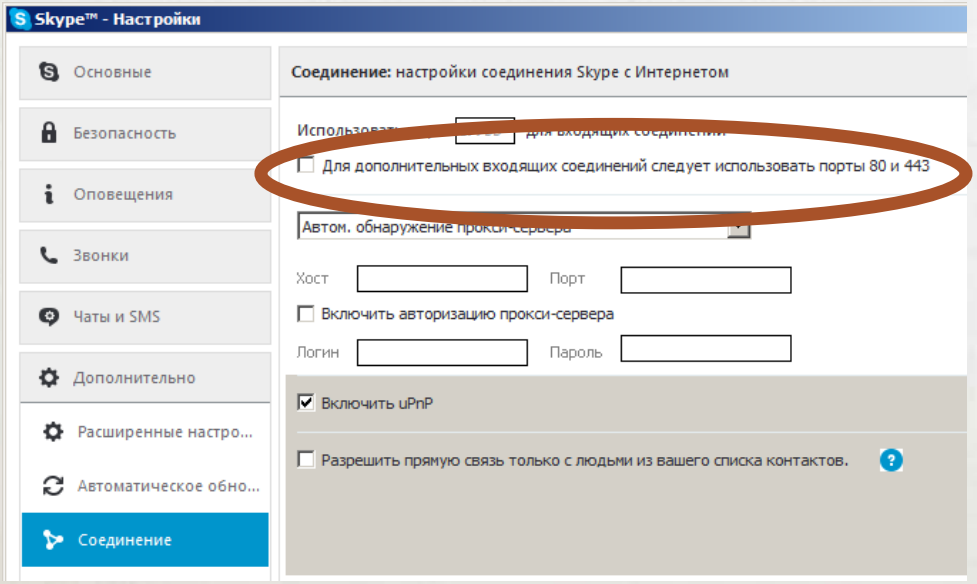
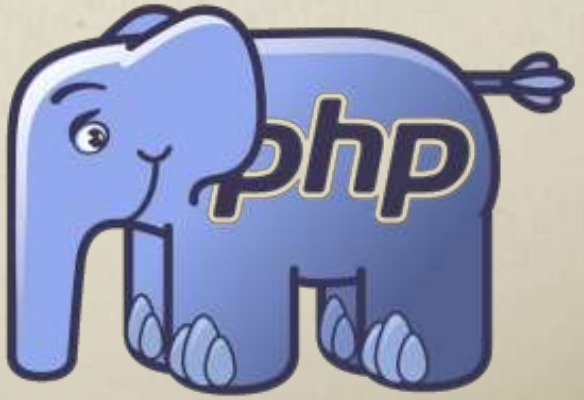
НАСТРАИВАЕМ СЕРВЕР

- **PHP** – работает на сервере. Сервером может быть как некий удалённый компьютер, так и программа на вашем компьютере, это неважно.

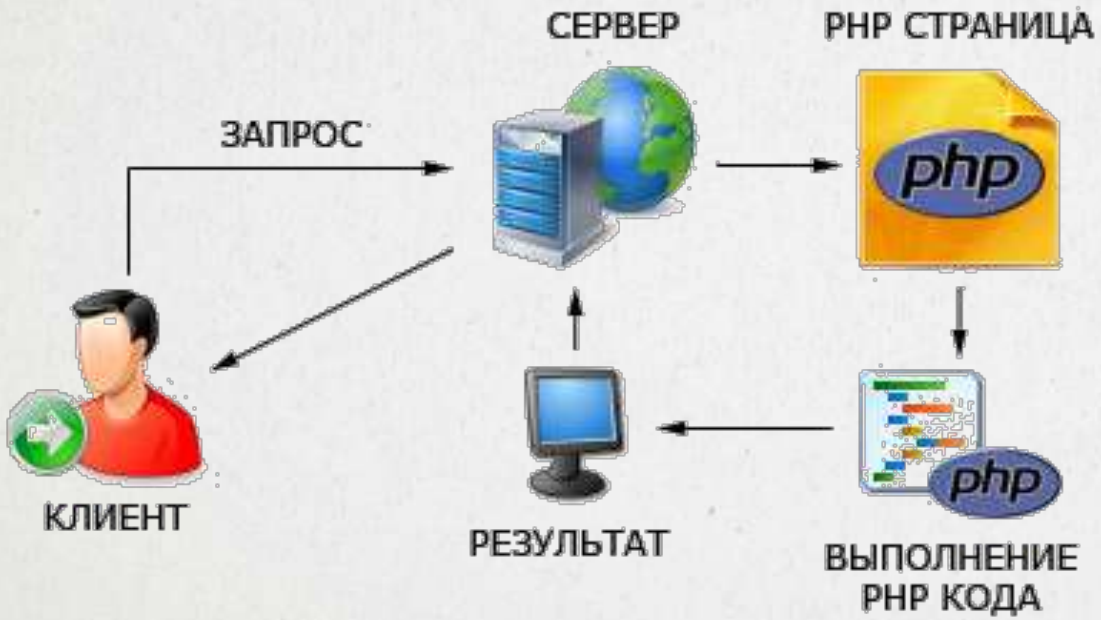
- **NB.** Если вы используете Skype – заставьте его освободить порт 80!





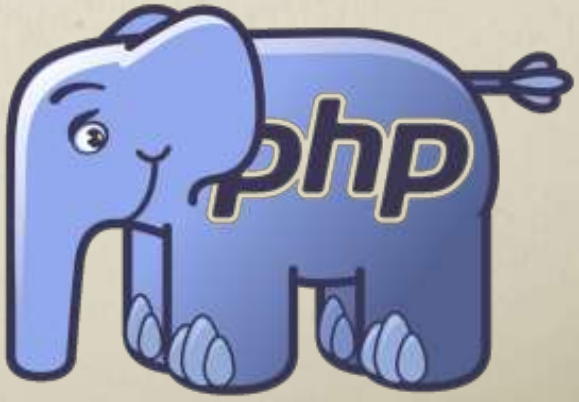
- **PHP –** это язык программирования. Не разметки, как HTML и не стилей, как CSS, а «настоящего» программирования
- Он предназначен для создания бизнес-логики вашего веб-приложения

PHP работает на сервере!



РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ НА PHP – ЭТО **ТЕКСТ**, который передается клиенту в ответ на его запрос

ГДЕ И КАК РАБОТАЕТ PHP?



ВЫВОД
или как увидеть работу программы?

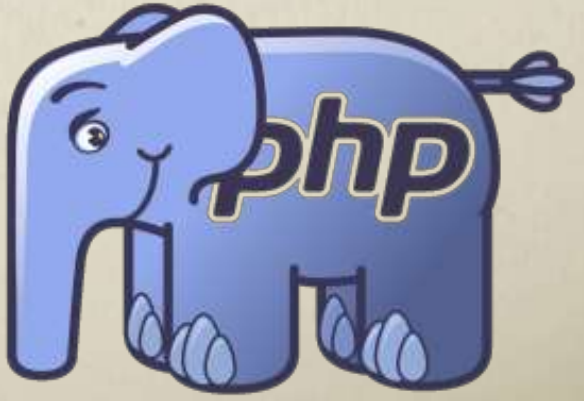
- Любое выражение можно вывести с помощью конструкции **echo**.

  При этом вы должны понимать, что вывод будет встроен в страницу, которая, после полной готовности, будет отправлена клиенту:

```
<html>
  <body>
    <p>
      Результат подсчета:
      <?php
        echo 2 * 2;
      ?>
    </p>
  </body>
</html>
```

NB. Не забудьте, что файл должен иметь расширение ".php", иначе чуда не произойдет!

ПЕРВАЯ ПРОГРАММА ☺



ВЫРАЖЕНИЯ

- **Выражение** – это некое значение (число, строка и т.д.), заданное в явном виде, или в виде вычислений. Например:

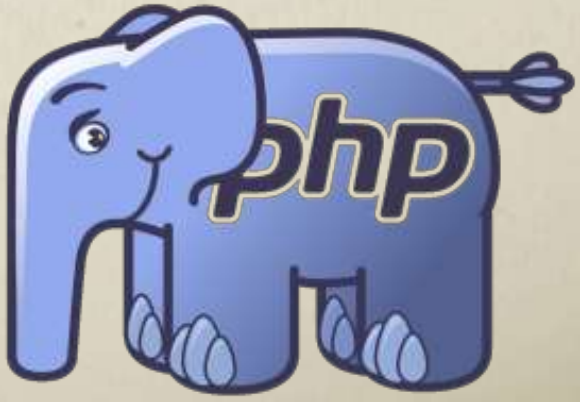
```
2     // это число
1.5   // это тоже число, но нецелое
'foo' // это строка из 3 символов
2+2   // это выражение, его результат =4
2*2   // не поверите, но тоже 4!
```

- Выражение имеет **значение** (то, чему оно равно)

Операции (операторы)

- Арифметические (+ - * /)
- Логические (&& || !)
- Сравнения (== != < >)
- Строковые
- Битовые
- Работы с массивами, проверки типа и другие, специфичные для PHP

ВЫРАЖЕНИЯ



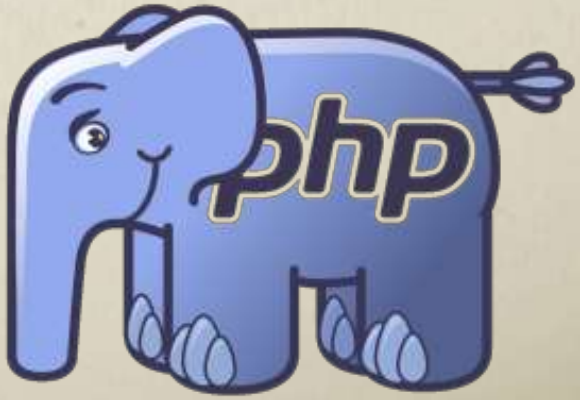
ПЕРЕМЕННЫЕ

- Переменная – это некое значение (выражение), сохраненное под понятным для нас именем. Например:

```
$email = 'test@example.com';
$age = 42;
$result = 2 + 2 * 2;
```

- Имя переменной в PHP начинается со знака «$». Обратите внимание, что $foo и $FOO – это разные переменные!
- В PHP объявление переменной (создание имени) и присваивание ей значения – это одна операция.

ПЕРЕМЕННЫЕ



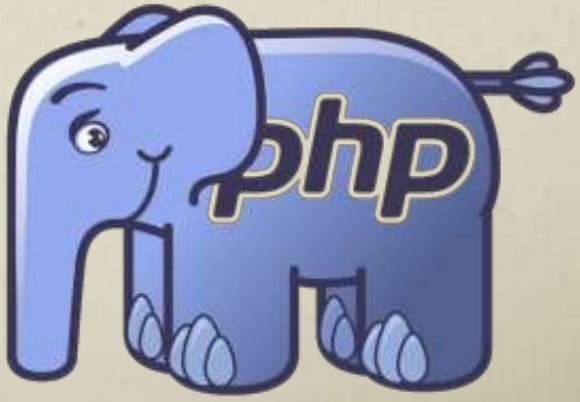
ТИП – это определение того, может быть значением

В PHP тип имеют значения, а не переменные. Переменная – это просто имя для значения, чтобы его сохранить на будущее.

Самое интересное, что тип выводится автоматически!

- `int, integer`
  Целые числа. Любые. Включая, конечно, ноль.
  Замкнуты относительно сложения, вычитания, умножения – результат снова будет целым числом. А вот при делении – не факт!
- `float`
  Тоже числа. Но не целые, а имеющие дробную часть. Иногда нулевую, но всё равно имеющие. Являются приближенными. Легко получить, написав число с десятичной точкой, или используя операции, дающие нецелый результат (деление, как пример)

ТИПЫ



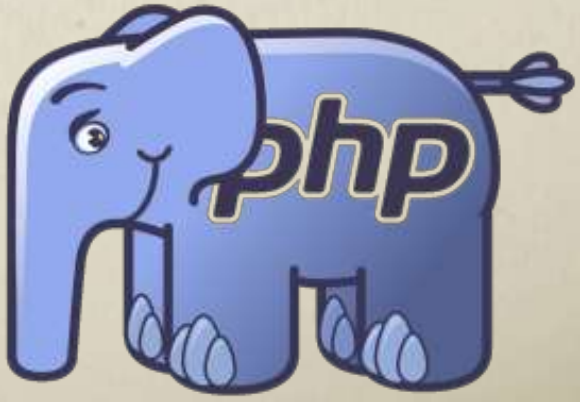
ТИП – это определение того, может быть значением

В PHP тип имеют значения, а не переменные. Переменная – это просто имя для значения, чтобы его сохранить на будущее.

Самое интересное, что тип выводится автоматически!

- `string`
  Строки. То, что состоит из символов. В PHP строки:
  - Могут быть любой длины (хоть вся «Война и мир»)
  - Могут быть нулевой длины (пустые)
    ```
    $str = '';
    ```
  - Могут заключаться в одинарные или двойные кавычки. Это почти одно и тоже.
    ```
    $str1 = 'test';
    $str2 = "test";
    ```
    но:
    ```
    $foo = 'Hello\n';
    $bar = "Hello\n";
    ```

ТИПЫ



ТИП – это определение того, может быть значением

Тип результата операции всегда определяется оператором!

Например, арифметические операции всегда производят числа:

```
$a = $x + $y;   // всегда число!
$a = 2 + '2';   // 4
$a = '2' * '2'; // тоже 4
```

А оператор «точка» (сложение строк) всегда производит строки:

```
$a = $x . $y;   // всегда строка!
$a = 2 . 2;     // '22'
$a = '2' . 2;   // тоже '22'
```

Этот механизм называется «приведением типов». Тип значения приводится к наиболее подходящему, чтобы соответствовать типу оператора.

ТИПЫ

ФУНКЦИЯ – это подпрограмма, которая принимает на вход параметры (аргументы) и возвращает результат

Одна из самых сильных сторон PHP – богатая стандартная библиотека функций

Функции можно использовать везде, где только можно использовать выражение.

Для использования функции нужно написать ее имя, а затем, в круглых скобках, список аргументов через запятую. Список можно быть и пустым!

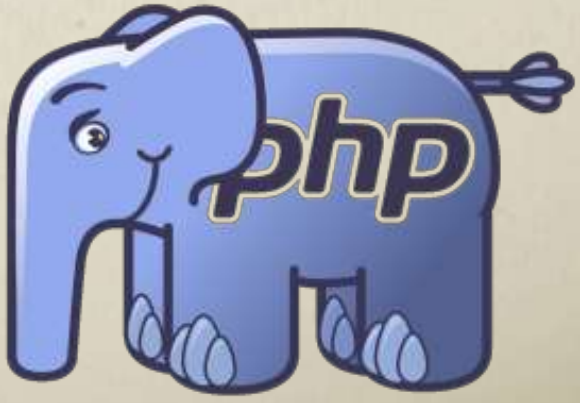
```
echo strlen('Hello');      // 5
$a = 2 + sqrt(2+2);        // 4.0

$x = str_replace('e', 'a', 'Hello');
  // 'Hallo'

'My number is ' . rand();
```

N.B. Невозможно знать наизусть ВСЕ функции стандартной библиотеки. Но возможно научиться пользоваться мануалом!

ФУНКЦИИ



boolean (bool) – тип, имеющий всего два значения:

- `true` – истина
- `false` - ложь

Например:

```
$a = true;
$b = false;

$x = (2 == 2);
$y = ($x != $a);
```

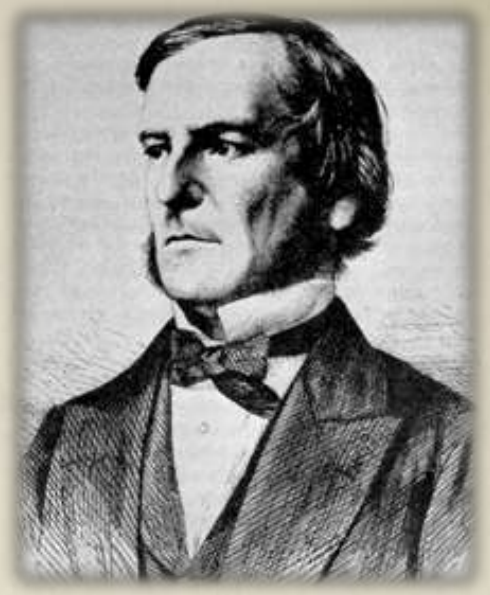
Булевы значения играют огромную роль в PHP. На них построены различные условия, многие функции принимают и возвращают булевы значения.

Приведение к boolean:

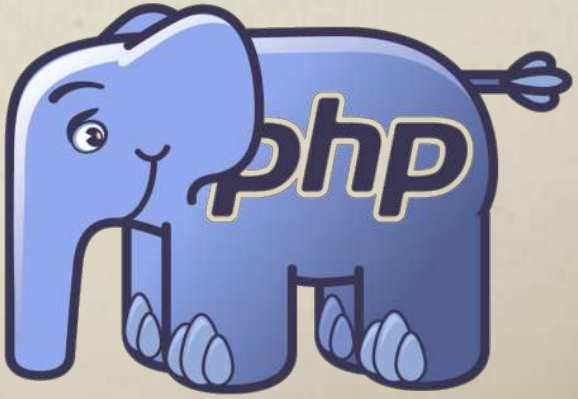
- `(bool)0 == false; (bool)'' == false`
- Ненулевое число, и непустая строка – это `true`
  - Исключение: `(bool)'0' == false`

Обратно:

- `(int)false == 0; (string)false == '';`
- `(int)true == 1; (string)true == '1';`



ТИП BOOLEAN



Булевы операторы – операторы, которые работают с булевыми значениями:

- **&&** – логическое «И»

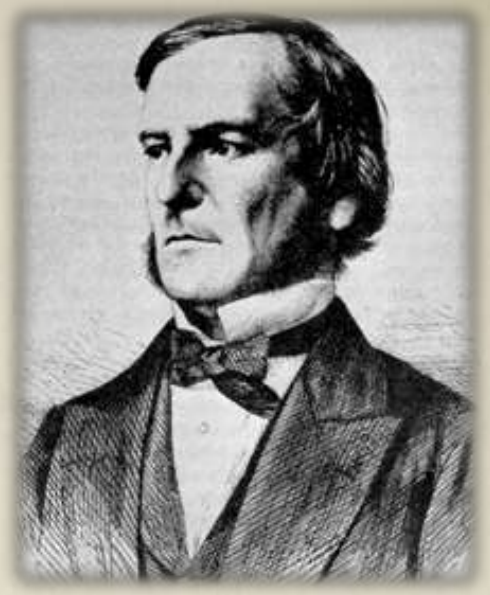
&&	0	1
0	0	0
1	0	1

- **||** – логическое «ИЛИ»

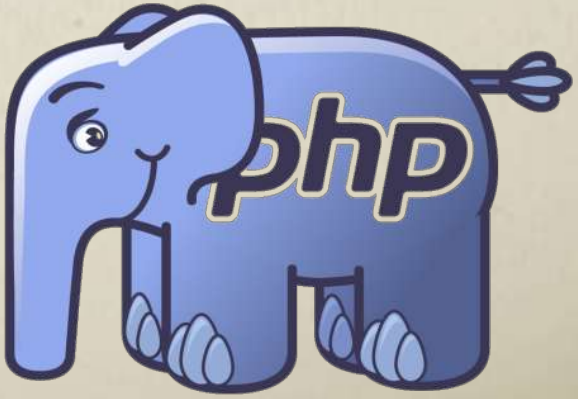
||	0	1
0	0	1
1	1	1

- **XOR** – логическое «ИСКЛЮЧАЮЩЕЕ ИЛИ»

xor	0	1
0	0	1
1	1	0



ТИП BOOLEAN



УСЛОВИЕ – это оператор PHP, определяющий, должен ли выполняться тот или иной код в зависимости от чего-либо

```
if ($x > y) {
  $m = $x;
} else {
  $m = y;
}
```

Сокращенная форма:

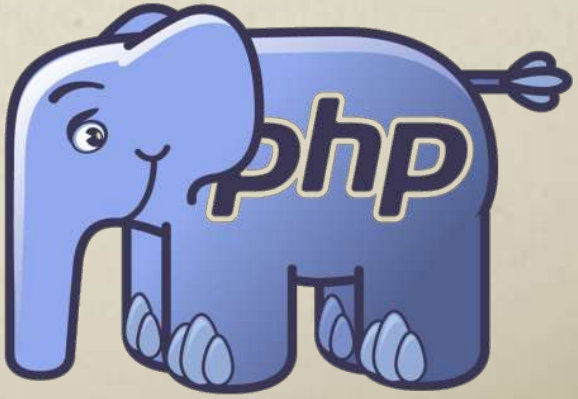
```
if ($stm) {
...
}
```

Форма с перебором условий:

```
if ($stm1) {
  ...
} elseif ($stm2) {
  ...
} else {
  ...
}
```

Важно! Всегда пишите фигурные скобки, даже если вам кажется, что это необязательно!

УСЛОВИЯ

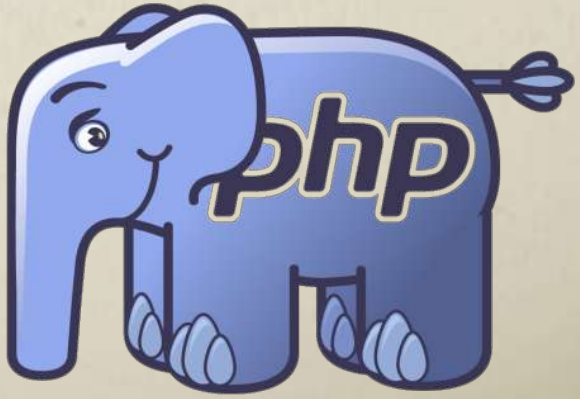


УСЛОВИЕ – это оператор PHP, определяющий, должен ли выполняться тот или иной код в зависимости от чего-либо

```
switch ($stm) {
  case 1:
     run1();
     break;
  case 2:
     run2();
     break;
  default:
     run();
     break;
}
```

- **break** прерывает перебор условий, не забывайте писать этот оператор!
- Секция **default** выполнится, если не совпало ни одно сравнение, ее может и не быть
- **Обратите внимание, что $stm и значения в case могут быть любыми – проверяется просто их равенство (нестрогое)**
- Несколько case могут быть записаны один за другим и отвечать за один и тот же случай

УСЛОВИЯ



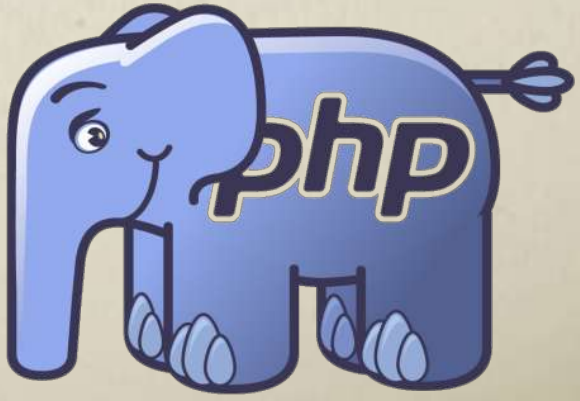
ФУНКЦИЯ – это подпрограмма, имеющая имя, набор входных значений (аргументов) и возвращаемое значение (необязательно)

```
function maxnum($a, $b) {
  if ($a > $b) {
    return $a;
  } else {
    return $b;
  }
}

$m = maxnum(3, 5);
echo maxnum(-1, 1);
```

- Имя функции должно быть уникальным и не совпадать с именем библиотечной функции
- **return** – это оператор возврата значения из функции. Их может быть несколько или не быть вообще. Оператор прерывает выполнение функции!
- Все переменные, объявленные в функции, будут локальными и перестанут существовать после выхода из нее
- Глобальные переменные недоступны в функции

ФУНКЦИИ



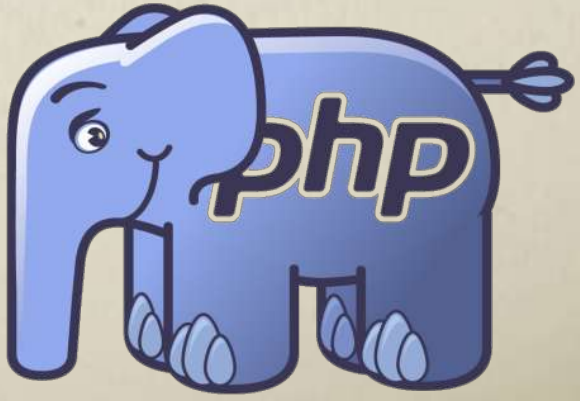
АРХИТЕКТУРНО ВЕРНО разделять свою программу на отдельные файлы.

В качестве первого шага к хорошей архитектуре можно собрать в один файл функции, а в другие файлы тот код, который их использует.

```
include __DIR__ . '/functions.php';
```

- **`__DIR__`** – это «магическая» константа. Она всегда содержит в себе полный путь в ФС до папки с текущим файлом
- **`include`** подключает файл, в случае его недоступности – выведет предупреждение, но продолжит работу программы
- **`require`** подключает файл, но в случае его недоступности вызовет фатальную ошибку и завершит программу
- **`include_once` и `require_once`** не будут подключать указанный файл, если он уже ранее был подключен

ВКЛЮЧЕНИЕ ФАЙЛОВ



МОДУЛЬНЫЙ ТЕСТ – это код, содержащий утверждения о том, как должны работать отдельные модули (части) вашей программы

В PHP принято использовать модульные тесты для того, чтобы доказать, что ваша программа работает верно, и для того, чтобы отслеживать возможные ошибки при изменении кода.

```
function maxnumber($a, $b) {
...
}

assert( 1 == maxnumber(1,1) );
assert( 1 == maxnumber(-1,1) );
assert( 1 == maxnumber(1,-1) );
```

- Старайтесь писать тесты до написания кода
- Используйте конструкцию assert, она встроена в язык
- Установите настройки:
  - `display_errors On`
  - `error_reporting E_ALL`

ТЕСТЫ