VisiBone PHP Skinny Sheet

PHP versions 4 and 5

A light, lean quick-reference cheatsheet by topic: MySQL, Strings, Regular Expressions, Forms, Time & Date, Math, Arrays, Types, Output, and many other features and tools.

mysql (classic) example

```
<?php
require("secure/passwords.php");
$link = mysql_connect($HOST, $USER, $PASS);
if (FALSE===$link) die(mysql_error());
mysql_select_db('INFORMATION_SCHEMA');
$result = mysql_query("SHOW TABLE STATUS");
if (FALSE===$result) die(mysql_error());
echo "<table border=1><tr>";
for($i=0 ; $i < mysql_num_fields($result) ; $i++) {
    $field = mysql_fetch_field($result, $i);
    echo "<th>$field->name";
}
while($row = mysql_fetch_row($result)) {
    echo "<tr>";
    foreach ($row as $cell) echo "<td> $cell";
}
echo "</table>";
mysql_close($link);
?>
```

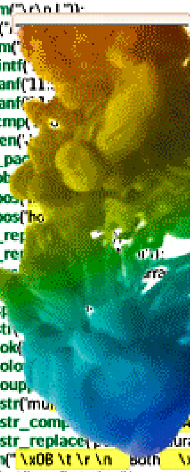
see visibone.com/php/mysql

mysqli (object) example (not PHP 4)

```
<?php
require("secure/passwords.php");
$mysqli = new mysqli($HOST, $USER, $PASS);
if (mysqli_connect_error() != '') die(mysqli_connect_error());
$mysqli->select_db('INFORMATION_SCHEMA');
$resulti = $mysqli->query("SHOW TABLE STATUS");
if ($mysqli->error != '') die($mysqli->error);
echo "<table border=1><tr>";
foreach ($resulti->fetch_fields() as $field) {
    echo "<th> $field->name";
}
while($row = $resulti->fetch_row()) {
    echo "<tr>";
    foreach ($row as $cell) echo "<td>$cell";
}
echo "</table>";
$mysqli->close();
?>
```

see visibone.com/php/mysqli

String



```
$s="abcd";  assert("c"==$s[2]);
assert("A"==chr(65));
assert(array('I', 'am')==explode(' ', 'I am'));
assert('I am'==implode(' ', array('I', 'am')));
assert('I am'==join(' ', array('I', 'am')));
assert("L"==ltrim("\r\n L"));
assert(65==ord("A ...
assert("R"==rtrim(" ...
assert('0255 255.0 FF fin'==sprintf( ... 255, 255, 'fini'));
assert(2==sscanf('11: ...
assert(array(11,59)==sscanf( ... (11==$h && 59==$m);
assert(strcmp( ...
assert(42==strlen(' ... ?'));
assert('~~x~~~'==str_pa ...
assert('7 ate 9.'==strpb ... digit & what follows)
assert(11==strpos( ... 'ar'));
assert(20==strpos('ho ... 'ar', 11+1));
assert('meimei'==str_rep ...
assert('Mayday'==str_re ...
assert(array('pet','pit')==str ... array('tex','tix'), $n) && $n==4);
assert(array('keen','edge')==str ...
assert(3==strsp ... )); // (consecutive digits)
assert('er after.'==strstr( ... 'er'));
assert('12'==strtok( ... && FALSE===strtok(' '));
assert('case'==strtolo ...
assert('CASE'==strtoup ...
assert('lie'==substr('mu ...
assert(0==substr_comp ... ARradiddle', 1, 3, TRUE)); // CcIi
assert('perdurable'==substr_replace( ... urabl', 3, 4));
assert("Both"==trim(" \x0B \t \r \n  Both   \x0B \t \r \n "));
assert('Cap first char'==ucfirst('cap first char'));
assert(' Cap All Words'==ucwords(' cap all words'));
assert('0255 255.0 FF fin'==vsprintf('%04d %5.1f %X %-3.3s', array(255, 255, 255, 'fini')));
```

Ink Drop Animation

```
$movie = new IDA_Movie(30);
$medium = IDA_Water(99.999);
$color = IDA_Rainbow(255,0,0);
$movie->setMedium($medium);
$movie->setColor($color);
$movie->play();
```

replace

output is an array if input is an array — can be arrays

	CScs	CcIi	**replace**
	`$sOut = str_replace($sOld, $sNew, $sIn);`	`$sOut = str_ireplace($sOld, $sNew, $sIn);`	by string match
	`$sOut = substr_replace($sIn, $sInsert, $iSkip, $iDelete);`		by numeric position

output is an array if input is an array

compare

CScs	CcIi	**compare** <=>
strcmp($s,$s)	strcasecmp($s,$s)	
strncmp($s,$s,$n)	strncasecmp($s,$s,$i)	length excerpt
substr_compare($s,$s,$i,$i)	substr_compare($s,$s,$i,$i,TRUE)	offset & length excerpt
strnatcmp($s,$s)	strnatcasecmp($s,$s)	natural order (9 < 22)
strcoll($s,$s)		for locale

search (===F)

CScs	CcIi	**search**
strpos($S,$s,$n)	stripos($S,$s,$n)	forward→
strrpos($S,$s,±$n)	strripos($S,$s,±$n)	←reverse

looong string (haystack) — short string (needle)

search & extract (match + what follows)

	CScs →	CcIi →	← CScs
identical {	strchr($S,$s) strstr($S,$s)	stristr($S,$s)	strrchr($S,$s)

Regular Expressions (Perl-like)

```
assert(array('it', 'is')==preg_grep('/i/', array('it', 'is', 'I')));
===F assert(1==preg_match('/i([st])/', 'If it is', $a));  assert($a==array('it','t'));  // strpos() is faster
===F assert(1==preg_match('/i([st])/', 'If it is', $a, PREG_OFFSET_CAPTURE));
     assert($a==array(array('it',3),array('t',4)));
===F assert(2==preg_match_all('/i([st])/', 'If it is', $a));  assert($a==array(array('it','is'),array('t','s')));
assert(40==strlen(preg_quote('\\.+*?[^]$(){}=!<>|:/', '/')));  // backslashify 19, plus slash
assert('ten ton tin'==preg_replace('/x/', 'n', 'tex tox tix'));
assert('ten ton tix'==preg_replace('/x/', 'n', 'tex tox tix', 2, $n)
       && $n == 2);
assert(array('If','it','is')===preg_split('/ /', 'If it is'));  // explode()
                                                                // is faster
assert(array('do','-','rue')==preg_split('/(-)/', 'do-rue', 2,
       PREG_SPLIT_DELIM_CAPTURE));
```

Regular Expression	what it means
^▭	starts with
▭$	ends with
.	any character (except LF)
(▭|▭)	either-or (use parens)
▭*	any number (0 or more)
▭?	optional (0 or 1)
▭+	etc. (1 or more)
▭{▭}	exactly (n)
▭{▭,}	at least (n or more)
▭{▭,▭}	range (n to n)
▭*? or ▭{...}? ...	make a quantifier ungreedy
[▭]	one of these characters
[▭-▭]	a character from X to X
[^▭]	any character except
(▭)	subpattern
(▭) \\1 \\2...	subpattern reference
(▭) \$1 \$2...	subpattern backreference
(? ☆)	many more features...
|▭| etc. ☆	different outer packaging
/▭/A	Anchor at string start
/▭$/D	Dollar: *don't* forgive final LF
/▭/e	eval replacee as PHP code
/▭/i	insensitive to case CcIi
/▭/m	^ multi-line edge match $
/▭/s	. subsume line separators
/▭/u	utf8 mode
/▭/U	UNGreedy (short) matches
/▭/x	extra space and comments
/▭/X	eXclaim illicit letter escape
\\d	digit or ordinal
\\D	non-digit
\\w	digit or letter or _
\\W	not a \\w
\\s	any white-space
\\S	not white-space
\\b	word boundary
\\B	inside or outside a word
/\\A▭/m	string-start anchor
/▭\\Z/m	string-end anchor
\\Q▭\\E	literal punctuation zone
e.g. \\n = \\x0A = \\cJ = \\012	specific control character
\\a \\f \\e	pattern has more character
\\(* \\. \\$...	backslashified literal punc's

String ⇔ String Conversion (Encode/Decode)

```
assert("a\\-list"==addcslashes("a-list", "-"));  // custom backslashify
assert(8==strlen(addslashes("'\"\0\\")));  // backslashify:  ' " NUL \
assert('6f6b'===bin2hex('ok'));
assert('line trim\r\nmer\r\n'===chunk_split('line trimmer', 9, "\r\n"));
assert("&auml;&copy;   &lt;&amp;&quot;&gt;"==htmlentities('ä©   <&">'));  // user-input sanitizer
assert('<&">   ä©'==html_entity_decode("&lt;&amp;&quot;&gt;   &auml;&copy;"));
assert("&lt;&amp;&quot;&gt;"==htmlspecialchars('<&">'));  // user-input sanitizer
assert('<&">'==htmlspecialchars_decode("&lt;&amp;&quot;&gt;"));
assert("line<br />\n"==nl2br("line\n"));
assert('MIME — RFC2045'==quoted_printable_decode('MIME =97 RFC2045'));
assert(22==strlen(quotemeta('.\\+*?[^]($)')));  // backslashify 11 chars
assert("\x5C\x22\x2A\x00\x0D\x0A"==stripcslashes('\\\\' . '\\"' . '\\*' . '\\000' . '\\x0D' . '\\n'));
assert("\x5C\x22\x2A\x00"==stripslashes('\\\\' . '\\"' . '\\*' . '\\0'));  // undo addslashes()
assert('no old bold pilots'==strip_tags('no <i>old</i>   <b>bold</b>   pilots'));
```

Legend

- <=> returning - 0 + to mean < = > (strcmp('a','z')<0 means 'a'<'z')
- gotcha – bugs are easy to write here
- ===F may return 0 or FALSE – so use === to test FALSE, not ==
- (use parens) – outside parentheses would be a good habit
- security alert, anticipate vulnerability
- user input issues, beware evil characters and bad puns
- diversity – Linux versus Windows, or web versus command-line
- ☆ obscure or redundant feature wants to go here – always optional
- = identical   contrast   unusual
- CcIi Case InsSensitive – functionNames(), ClassNames, TRUE, false, NulL, "/regular-expressions/i", stripos(), strcasecmp()
- CScs Case Sensitive – $variableNames, define('CONSTANTS'), class { const }, "/regex/", strpos(), strcmp() max(), and all operators: == === < <= > >=
- `output` web page content (HTML, etc.) or command-line console output
- PHP feature **PHP feature emphasized**
- PHP version 5.*latest* – no can do in version 4.*latest*
- the interesting part   html <tag>   sql statement
- $váriaбle_ňamė – CScs ISO 8859 letters, noninitial digits, underscore, 0x90-0xFF
- fûnçti0n_nåme() – CcIi letters, noninitial digits, underscore, 0x90-0xFF
- CONSTANT (define() and const) – CScs letters, noninitial digits, underscore, 0x90-0xFF – (ALL CAPS here by convention)
- ClassName { } – CcIi letters, noninitial digits, underscore, 0x90-0xFF – (Capitalized here by convention)
- assert(test) – this test should almost always be true
- guess(test) – plausible, could be true in special circumstances

Control Structures

```
function bold($text = '') {
    return "<b>$text</b>";
}
assert("<b>!</b>"==bold('!'));
```

```
if ($task instanceof Difficult) {
    $task->asif_Easy();
} elseif ($task instanceof Easy) {
    $task->asif_Difficult();
} else {
    confidence('sleep or dismay');
}
```

```
function couragePhillips() {
    while (grip('fear')) {
        actEffectively();
    }
}
```

```
function courageMcCarthy() {
    do {
        wiseChoices();
    while (feeling('fear'));
}
```

```
$gaußanswer = 0;
for ($i = 1 ; $i <= 100 ; $i++) {
    $gaußanswer += $i;
}
assert(5050 == $gaußanswer);
```

```
// comments
/* com-
ments */
# comments
```

```
switch ($genius) {
case 'intelligence':
    neither();
    break;
case 'imagination':
    nor();
    break;
default:
    love();
    love();
    love();
    break;
}
```

```
$times = array('minute', 'hour', 'day');
foreach ($times as $time) {
    sense("This $time, life is good.");
}

$efforts = array('idea' => 1, 'recon' => 9, 'act' => 90);
foreach ($efforts as $effort => $percent) {
    $efforts[$effort] = $percent/100.0;
}
```

PHP Skinny Sheet

VisiBone

```
array (
  'seconds' => 59,
  'minutes' => 59,
  'hours' => 23,
  'mday' => 31,
  'wday' => 5,
  'mon' => 12,
  'year' => 1999,
  'yday' => 364,
  'weekday' => 'Friday',
  'month' => 'December',
  0 => 946702799
)
```

```
class Voter extends Citizen {
    var $party;
    function Voter($name) {
        parent::Citizen($name);
    }
    function register($party) {
        $this->party = $party;
    }
}
$o = new Voter('John Q.');
```

```
interface Lawyer {
    public function argue(array $cases);
}
class Senator extends Voter implements Lawyer {
    static private $record = '';
    final public function argue(array $cases) {
        Senator::$record .= join($cases);
    }
    protected function influence(Lobbyist $x) {}
}
```

Time & Date

```
assert('23:59:59 Fri 31-Dec-1999'==date('H:i:s D d-M-Y',mktime(23,59,59,12,31,1999)));
assert('00:00:00 Sat 01-Jan-2000'==date('H:i:s D d-M-Y',mktime(23,59,59,12,31,1999)+1));
var_export(getdate(mktime(23,59,59,12,31,1999)));   // →
$a=getdate(time());  guess('Monday'==$a['weekday']);
guess('0.20934800 1200089159'==microtime());
guess(1200089159.209348==microtime(TRUE));
assert(1==mktime(0,0,0,1,1,2000)-mktime(23,59,59,12,31,1999));  // y2k moment
$t1=time();  dosomething();  $t2=time();  $nSeconds=$t2-$t1;
```

Function Tools

```
$f=create_function('$x', 'return 2*$x;');  assert(4==$f(2));
function f() { return func_get_args(); }  assert(array(1,'two')==f(1,'two'));
```

Array Simple

```
$a=array(2, 'b', 'or'=>'not', array('2b'), );  // trailing comma ok
assert(4==count($a)
   && 2==$a[0]
   && 'b'==$a[1]
   && 'not'==$a['or']
   && '2b'==$a[2][0]);
assert(3==count(array(7,8,9)));
assert(6==count(array(7,8,array(9,10,11)), COUNT_RECURSIVE));
assert(array('I', 'am')==explode(' ', 'I am'));
assert('I am'==implode(' ', array('I', 'am')));  // ≡ join()
assert(TRUE==in_array(333, array(1, 2, '333', 4)));
assert('I am'==join(' ', array('I', 'am')));
list($one,$two)=array(1,2);  assert(1==$one && 2==$two);
assert(array(1,2,3,4,5)==range(1,5));
$a=array(1,2,3,4,5);  assert(shuffle($a));  guess(array(2,5,3,1,4)==$a);
assert(3==sizeof(array(7,8,9)));
```

Simple Math

```
assert(5==abs(-5));
assert(2.0==ceil(1.1) && -1.0==ceil(-1.9));
assert(2.0==floor(2.9) && -3.0==floor(-2.1));
assert(2.5==fmod(12.5, 10.0));
assert(FALSE==is_finite(9e9999));
assert(TRUE==is_infinite(9e9999));
assert(0.0 <= lcg_value() && lcg_value() <= 1.0);  // random
assert(9==max(7,8,9) && 'y'==max('x','y','Z'));  // CScs
assert(1==min(3,2,1) && 'a'==min('c','b','a'));
assert(0 <= mt_rand() && mt_rand() <= mt_getrandmax());
$i = mt_rand(-1000,1000);  assert(abs($i) <= 1000);
mt_srand(0);  assert(963932192==mt_rand());
assert(0 <= rand() && rand() <= getrandmax());  // ✿
assert(-100 <= rand(-100,100) && rand(-100,100) <= 100);
assert(-2.67==round(-2.666, 2));
srand(0);  assert(12345==rand());
```

Operator Precedence

```
$o = new ClassName;
$o = new $sClassName;

$Z = $a[$i];  // array element by numeric index
$Z = $a[$s];  // array element by string key
$s = $s[$i];  // a character from a string ⊘

$n++;  ++$n;  // post- and pre-increment
$n--;  --$n;  // post- and pre-decrement

$i = ~$i;   // bitwise complement
$n = -$n;
$Z = @$Z;   // hide error messages
$b = (bool)$z;  $b=(boolean)$z;
$i = (int)$z;  $i=(integer)$z;
$x = (double)$z;  $x=(float)$z;  $x=(real)$z;
$s = (string)$z;
$a = (array)$Z;
$o = (object)$Z;
NULL==(unset)$Z;
```

Specify the class name, a string containing the class name, or an object instance of the class.

```
$b = $Z instanceof  C;
$b = $Z instanceof  $sClassName;
$b = $Z instanceof  $o;
```

Is $Z an object of the class? Or a class that extends or implements the class?

```
$b = !$b;  // boolean not

$n = $n * $n;
$n = $n / $n;
$i = $i % $i;  // integer modulo

$a = $a + $a;  // union by keys, not values
$n = $n + $n;
$n = $n - $n;
$s = $s . $s;  // string concatenate, à la Perl

$i = $i << $i;  // shift bits left
$i = $i >> $i;  // shift bits right

$z = $z < $z;   // less
$z = $z <= $z;  // less or equal
$z = $z >= $z;  // more or equal
$z = $z > $z;   // more
```

all these are CScs

```
$b = $Z==$Z;   // loose equal  not = assignment
$b = $Z != $Z;  // not loosely equal
$b = $Z <> $Z;  // inequal
$b = $Z===$Z;  // strictly equal (same type)
$b = $Z !== $Z;  // not strictly equal

$i = ($i & $i);  // bitwise 'and' ( use parens)
$Z = &$Z;  // assign (or pass or return) by reference

$i = ($i ^ $i);  // bitwise 'xor' ( use parens)

$i = ($i | $i);  // bitwise 'or' ( use parens)

$b = $b && $b;  // boolean logic  ≡ $b and $b

$b = $b || $b;  // boolean logic  ≡ $b or $b

$Z=($b ? $Z : $Z);  // if-else chooser ( use parens)

$Z = $Z;  // assign by value (copy)
$n += $n;  // add to, etc.
$n -= $n;
$n *= $n;
$n /= $n;
$i %= $i;
$s .= $s;  // concatenate to
$i &= $i;
$i |= $i;
$i ^= $i;
$i <<= $i;
$i >>= $i;

$b = $b and $b;  // boolean logic  ≡ $b && $b

$b = $b xor $b;  // boolean logic

$b = $b or $b;  // boolean logic  ≡ $b || $b
```

Integer ⇔ String

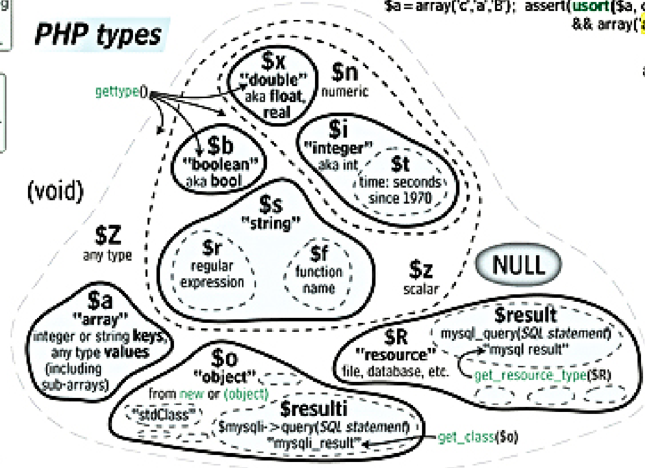
```
assert(44===(int)'44');
assert(44===(integer)'44');
assert(44==0+'44');
assert(2.0==doubleval('2'));
assert(-2.0==floatval('-2'));
assert(-3==intval('-3garbage'));
```

String ⇔ Integer

```
assert('44'===(string)44);
assert('44'==='' . 44);  // (concatenate empty string)
assert('3.14'==strval(3.140));
```

Array Sort

```
$a=array(2,1,1);  $b=array(0,8,9);  assert(array_multisort($a,$b) && array(1,1,2)==$a && array(8,9,0)==$b);
$a=array(7,8,9);  assert(arsort($a) && array(2=>9, 1=>8, 0=>7)==$a);
$a=array(9,8,7);  assert(asort($a) && array(2=>7, 1=>8, 0=>9)==$a);
$a=array(8=>1, 7=>1, 9=>1);  assert(krsort($a) && array(9=>1, 8=>1, 7=>1)==$a);
$a=array(8=>1, 7=>1, 9=>1);  assert(ksort($a) && array(7=>1, 8=>1, 9=>1)==$a);
$a=array('A222', 'a99');  assert(natcasesort($a) && array(1=>'a99', 0=>'A222')==$a);
$a=array('a222', 'a99');  assert(natsort($a) && array(1=>'a99', 0=>'a222')==$a);
$a=array(8,7,9);  assert(rsort($a) && array(9,8,7)==$a);
$a=array(8,7,9);  assert(sort($a) && array(7,8,9)==$a);
$a=array('K12','k9');  assert(uasort($a, 'strnatcasecmp') && array(1=>'k9', 0=>'K12')==$a);
$a=array('b12'=>0,'b4'=>0);  assert(uksort($a, 'strnatcmp') && array('b4'=>0,'b12'=>0)==$a);
$a=array('c','a','B');  assert(usort($a, create_function('$L,$R', 'return strcasecmp($L,$R);'))
                            && array('a','B','c')==$a);
```



Type Test

```
guess('stdClass'==get_class($o));
assert('mysql link'==get_resource_type($link));
assert('integer'==gettype(9));
assert(is_array(array(1,2,3)));
assert(is_bool(FALSE));
assert(is_callable('strlen'));
assert(is_double(7.5));
assert(is_float(7.5));
assert(is_integer(3) && is_int(3));
assert(is_null(NULL));
assert(is_numeric(2.55e2));
assert(is_object(new stdClass));
assert(is_real(7.5));
guess(is_resource(mysql_connect('x.com')));
assert(is_scalar('s') && !is_scalar(array()));
assert(is_string('abc'));
```

Variables are function-local. The two ways to use a global variable within a function:
global $variable; // once at function top
$GLOBALS['variable'] // anywhere.

Every function behaves as if it had these automatically:
global $GLOBALS, $_COOKIE, $_ENV, $_FILES, $_GET;
global $_POST, $_REQUEST, $_SERVER, $_SESSION;

Forms

```
guess('file.jpg'==$_FILES['pic']['name']);  // e.g. from <input type='file' name='pic'>
guess('value'==$_GET['field']);  // aka $HTTP_GET_VARS[]  e.g. <input name=\'field\'>
guess('value'==$_POST['field']);  // aka $HTTP_POST_VARS[]
guess('value'==$_REQUEST['field']);  // merged $_GET[] and $_POST[] etc.
```

Toolbox

```
assert(4==2+2);
assert('4==2+2');  // more informative failure
define('THREE', 3);  assert(3 == THREE);
define('TWO', 2);  assert(defined('TWO'));  // remember quotes
hopethisreturnstrue() or die('dashed hope');
$z=array();  assert(empty($z));  $z='';  assert(empty($z));
guess('linux'==$_ENV['TERM']);
if (errorlevel99()) { exit(99); } else { exit('message'); }
$a=getallheaders();  guess(preg_match(
                          '/Mozilla/', $a['User-Agent']));
global $z;  $z='one way';        // (inside a
$GLOBALS['z']='another way';     // function)
include 'localfile.php';
include_once('localfile.php');
$q=2;  assert(isset($q) && !isset($undefinedvariable));
```

Output

```
odd-looking output command        ?>odd-looking output command<?php
with parentheses                  echo("with "."parentheses");
without parentheses               echo "without ", "parentheses";
almost identical to echo()        print "almost identical to echo()";
3.14                              assert(4==printf('%4.2f', M_PI));
assert("Array\n(\n    [0] => 33\n)\n"==print_r(array(33), TRUE));
assert("array (\n 0 => 4,\n 1 => 'ensic',\n)"==var_export(array(4,'ensic'), TRUE));
3.14                              assert(4==vprintf('%4.2f', array(M_PI)));
```

print_r(array(4, 'ensic'));

```
Array
(
    [0] => 4
    [1] => ensic
)
```

var_dump(array(4, 'ensic'));

```
array(2) {
  [0]=>
  int(4)
  [1]=>
  string(5) "ensic"
}
```

var_export(array(4, 'ensic'));

```
array (
  0 => 4,
  1 => 'ensic',
)
```

(outputs PHP syntax)

```
$to="\"Reese E. Vuur\" <to@x.com>";
$subject="Unfiltered user input is dangerous";
$headers="From: \"S.N.Dürr\" <fm@x.com>\r\n";
$message="Multi-\nline\nmessage.\n";
assert(mail($to, $subject, $message, $headers));  //
ob_start();  echo 'output';  $s=ob_get_clean();  assert('output'==$s);
...a very very long report...  phpinfo();  // password-protect this report
guess('5.2.5'==phpversion());
require 'localfile.php';
require_once 'localfile.php';
guess('/home/username/htdocs'==$_SERVER['DOCUMENT_ROOT']);
guess('Mozilla/4.0 (compatible;  MSIE 6.0;  Windows NT 5.0)'==$_SERVER['HTTP_USER_AGENT']);
guess('http://host.com/frompage.html'==$_SERVER['HTTP_REFERER']);  // (sic)
guess('/path/file.php'==$_SERVER['PHP_SELF']);
guess('217.5.153.85'==$_SERVER['REMOTE_ADDR']);
guess('/home/username/htdocs/path/file.php'==$_SERVER['SCRIPT_FILENAME']);
$q=2;  unset($q);  assert(!isset($q));
```