MÃ NGUỒN MỞ

PHẦN III – XÂY DỰNG VÀ PHÁT TRIỂN PHẦN MỀM NGUỒN MỞ

PHAN TRỌNG TIẾN
BM Công nghệ phần mềm
Khoa Công nghệ thông tin, VNUA
Email: phantien84@gmail.com
Website: http://timoday.edu.vn

Nội dung chính

1. Giới thiệu Lập trình PHP
2. Giới thiệu Hệ quản trị cơ sở dữ liệu MySQL

PHP

Personal Home Page

Lịch sử ra đời của PHP

- 1994, Rasmus Lerdorf tạo ra ngôn ngữ PHP và được tiếp tục phát triển bởi nhiều người khác.
- Thường sử dụng PHP xây dựng ứng dụng thương mại điện tử
- 2001 đã có 5 triệu tên miền sử dụng PHP
- PHP là *Open Source, bạn có thể làm việc trên mã nguồn, thêm, sửa, sử dụng và phân phối chúng.*

Nhúng PHP trong HTML

- Khóa PHP đầy đủ

  <?php

  … PHP code …

  ?>
- Khóa PHP rút gọn

  <?

  … PHP code …

  ?>

Lệnh và chú thích

- Lập trình PHP phải tuân theo chuẩn
- Mỗi lệnh phải kết thúc bằng dấu ; (trừ lệnh cuối trước khóa ?>)
- Khối (nhiều) lệnh được đặt trong cặp { }
- Có ba cách ghi chú thích:
  1. // chú thích có giá trị đến cuối dòng
  2. # chú thích có giá trị đến cuối dòng
  3. /*

     chú thích trên nhiều dòng

     */

Ví dụ

```
1
2  <?
3  Echo "PHP is simple"; //day la vi du ve code PHP
4  /* Voi cu phap nay chung ta
5  Co the  chu thich 1 cum ma lenh */
6  ?>
7
```

```
1  <?php
2  Echo "Hello word";
3  Printf"<br><font color=red>Who Are You ?</font>";
4  ?>
5
```


Các kiểu dữ liệu cơ bản

Kiểu	Ví dụ	Mô tả
integer	99	Số nguyên
double	25.76	Số thực
string	”hello” ’ xin chào’ <<<HI chào buổi sáng. HI;	Chuỗi ký tự
boolean	True	true hoặc false

Biến

- Cách dùng:
  - $tên_biến
  - Không cần khai báo trước khi dùng
  - Gán giá trị bằng toán tử =
- Quy ước về cách đặt tên:
  - Bắt đầu bằng *chữ cái hoặc gạch dưới (_)*
  - *Không chứa ký tự trắng* (space, tab)
  - *Phân biệt in hoa – thường*

Ví dụ sử dụng biến

- Gán giá trị cho biến

```
<?php
    $qty = 30;
    $price = 20;
    $total = $qty * $price;
    echo "Tong tien :" . $total;
?>
```

- Thay đổi biến

```
<?php
    $qty = "soluong";
    echo "qty:" . $qty ."<br>";
    $$qty = 40;
    echo "so luong :" . $soluong;
?>
```

Chú ý: Toán tử "." dùng để nối chuỗi

Sự chuyển đổi kiểu dữ liệu

Có hai hình thức ép kiểu chính

- *Ép kiểu ngầm định*

  Xảy ra tự động khi thực hiện các toán tử đòi hỏi hai biểu thức cùng kiểu

- *Ép kiểu chỉ định*

  Chỉ định một kiểu dữ liệu cụ thể đặt trong cặp () trước biểu thức cần ép kiểu

Một số hàm liên quan đến ép kiểu

- bool is_*type* ($tên_biến hay biểu thức):

  is_integer, is_float, is_numeric, is_string, is_bool, is_array, is_double, is_real, is_int, is_object

  - *Kiểm tra dữ liệu của một biến, kết quả trả về true hoặc false*

- string gettype($tên_biến hay biểu thức)
  - *Trả về loại kiểu dữ liệu như: integer, double, long ...*

- int settype($tên_biến, "kiểu_dữ_liệu")
  - *Gán kiểu dữ liệu cho tên biến*

Trị và tham chiếu

- Khi thực hiện phép gán biến cho biến thì mặc định giá trị được sao chép từ biến nguồn sang biến đích

  Ví dụ: $a = $b (Giá trị của $b được sao chép sang $a)

- Dùng tham chiếu khi muốn đặt thêm một tên cho một biến có sẵn

  Ví dụ: $x = &$y (lúc này $x và $y là hai tên của cùng một biến)

Phạm vi của biến

- Có ba mức phạm vi:
  - Biến hàm:
    - Được khai báo và sử dụng cục bộ trong phạm vi hàm
  - Biến toàn cục (không nằm trong hàm):
    - Được khai báo và sử dụng bên trong một script, mặc định là không thể sử dụng bên trong các hàm
  - Biến siêu toàn cục:
    - Có thể sử dụng ở mọi nơi, không thể định nghĩa bởi người dùng

Một số biến siêu toàn cục

- $GLOBALS
- $_SERVER
- $_GET, $_POST
- $_SESSION, $_COOKIE
- $_REQUEST
- $_ENV
- $php_errormsg

Biến $GLOBAL

- PHP coi 1 biến có một giới hạn. Để xác định một biến toàn cục (global) có tác dụng trong một hàm , ta cần khai báo lại. Nếu không giá trị của biến sẽ được coi như là biến cục bộ.
- Ví dụ

```
<?
    $a = 1;
    $b = 2;

    function Sum ()
        {
          global $a, $b;
          $b = $a + $b;
        }

    Sum ();
    echo $b;
?>
```

Biến $GLOBAL

- Một cách khác để dùng biến toàn cục trong 1 hàm là ta dùng mảng $GLOBAL của PHP
- Ví dụ

```
<?
    $a = 1;
    $b = 2;

    function Sum ()
    {
      $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
    }

    Sum ();
    echo $b;
?>
```


Biến $REQUEST

- Lấy các giá trị của GET, POST, COOKIE … theo thứ tự EGPCS (Enviroment,Get, Post, Cookie, Server)
- Tuy nhiên , các phần tử trong mảng REQUEST là hoàn toàn độc lập với các phần tử trong mảng GET , POST vvv... Bạn có thể thay thế bằng giá trị khác với mảng REQUEST như giá trị trong GET,POST thì không đổi.
- Ví dụ:

```
<?
 $_POST['username'] = "cottonbelly";
 $_GET['username'] = "snoopy0877";

 echo $_POST['username']; // sẽ in ra : cottonbelly
 echo $_GET['username']; // sẽ in ra : snoopy0877
 echo $_REQUEST['username']; // sẽ in ra : snoopy0877

 $_REQUEST['username'] = "lambada";

 echo $_POST['username']; // sẽ in ra : cottonbelly
 echo $_GET['username']; // sẽ in ra : snoopy0877
 echo $_REQUEST['username']; // sẽ in ra : lambada thay vì snoopy0877
?>
```

Hằng

- Định nghĩa:
  - *define* ('tên_hằng', giá trị)
  - Giá trị hằng chỉ được dùng các kiểu dữ liệu cơ bản
  - Bắt buộc định nghĩa trước khi dùng
- Quy ước về cách đặt tên:
  - *Giống* cách đặt tên biến
  - Không sử dụng ký hiệu *$*
  - Thường đặt tên bằng chữ *in hoa*

Ví dụ

```
<?
$a= 100 // biến a ở đây có giá trị là 100.
$a= "PHP is easy" // Biến a ở đây có giá trị "PHP Is easy".
Biena=123 //Có lỗi vì bắt đầu 1 biến phải có dấu "$"
$123a="PHP" //Có lỗi vì phần tên bắt đầu của biến là dạng số.
?>
```

```
<?
define ("C", "COMPANY");
define ("YELLOW", "#ffff00");
echo "Gia tri cua C la". C;
?>
```

Toán tử: gán và số học

Gán	Số học	Kết hợp
=	+	+=
	-	-=
	*	*=
	/	/=
	%	%=

Toán tử: so sánh

Ký hiệu	Ý nghĩa
==	Bằng giá trị
===	Bằng giá trị và cùng kiểu
!=	Khác giá trị
<>	Khác giá trị
!==	Khác giá trị hoặc khác kiểu
<	Nhỏ hơn
>	Lớn hơn
<=	Nhỏ hơn hoặc bằng
>=	Lớn hơn hoặc bằng

Toán tử: logic

Ký hiệu	Ý nghĩa
and	And
&&	And
or	Or
||	Or
xor	Xor
!	Not

Toán tử: bitwise

Ký hiệu	Ý nghĩa
&	And
|	Or
^	Xor
~	Not
<<	Dịch trái
>>	Dịch phải

Toán tử: tăng giảm 1

Ký hiệu	Ý nghĩa
++	Tăng 1
--	Giảm 1

Thứ tự ưu tiên phép toán

Qui tắc liên kết	Toán tử	Ghi chú
	new	Tạo 1 đối tượng từ 1 class, toán tử này chỉ áp dụng trên 1 toán hạng nên không có qui tắc liên kết
Bên phải trước	[	Toán tử truy cập 1 phần tử trong mảng
	++ --	Tăng/Giảm 1 đơn vị, toán tử này chỉ áp dụng trên 1 toán hạng nên không có qui tắc liên kết
	! ~ - (int) (float) (string) (array) (object) @	Các toán tử này chỉ áp dụng trên 1 toán hạng nên không có qui tắc liên kết
Bên trái trước	* / %	
Bên trái trước	+ - .	
Bên trái trước	<< >>	
	== != === !===	Toán tử so sánh, chỉ áp dụng trên 2 toán hạng nên không có qui tắc liên kết
Bên trái trước	&	
Bên trái trước	^	
Bên trái trước	|	
Bên trái trước	&&	
Bên trái trước	||	
Bên trái trước	? :	
Bên phải trước	= += -= *= /= .= %= &= |= ^= <<= >>=	
Bên trái trước	and	
Bên trái trước	xor	
Bên trái trước	or	
Bên trái trước	,	

Các câu lệnh điều khiển PHP

- Câu lệnh *If*
- Câu lệnh *Switch*
- Vòng lặp *While/Do...While*
- Vòng lặp *For*
- Vòng lặp *Foreach*
- Câu lệnh *Break*
- Câu lệnh *Continue*
- Câu lệnh *Return*
- Câu lệnh *Include*

Câu lệnh *If*

```
if (biểu thức điều kiện)
    khối lệnh 1;
else
    khối lệnh 2;
```

- Các câu lệnh If có thể lồng nhau

Ví dụ

```php
<?php
$a=5;
$b=7;
if($a < $b)
{
    echo "Bien A co gia tri nho hon bien B";
}
else
{
    echo "Bien A co gia tri lon hon bien B";
}
?>
```


Câu lệnh *Switch*

```
switch (biểu thức)
{
   case biểu thức 1:
    khối lệnh 1;
   case biểu thức 2:
    khối lệnh 2;
   ...
   case biểu thức n:
    khối lệnh n;
   default:
    khối lệnh cuối;
}
```

Vòng lặp *While/Do...While*

- while (biểu thức điều kiện)
  khối lệnh;

- do
  khối lệnh;
  while (biểu thức điều kiện);

Ví dụ *While/Do...While*

```
<?php
   $i = 1;
   while ( $i <= 10 )
   {
     echo $i, "\n";
     $i++;
   } //end while
?>

<?php
    $i = 1;
    do
    {
      echo $i, "\n";
      $i++;
    }
    while ( $i < 10 );
?>
```

Vòng lặp *For*

for (biểu thức 1; biểu thức 2; biểu thức 3)
khối lệnh;

- biểu thức 1: thực hiện 1 lần khi bắt đầu vòng lặp
- biểu thức 2: điều kiện lặp, được xem xét trước mỗi lần lặp
- biểu thức 3: thực hiện sau mỗi lần lặp

```
<?php
    for ( $i = 0; $i < 10; $i++ )
    {
        echo $i, "\n";
    } //end for
?>
```


Vòng lặp *Foreach*

- Câu lệnh foreach chỉ làm việc với mảng.
- foreach ($array as $value)
  câu lệnh;
- foreach ($array as $key => $value)
  câu lệnh;

```
<?php
 $a = array('a' => 1, 'b' => '2', 'c' => '3');
 foreach ( $a as $value )
 {
        echo $value, "\n";
 } //end foreach
?>
```

Các lệnh ngắt lặp

- *Break*

  Dừng và thoát ra khỏi vòng lặp *for, foreach, while, do-while* và *switch*

- *Continue*

  Dừng thực hiện lần lặp hiện hành để chuyển sang lần lặp tiếo theo

Câu lệnh *Return*

- Trong một hàm, câu lệnh ***Return*** kết thúc việc thực thi hàm và trả về kết quả. Nó cũng kết thúc thực hiện script.

```
<?php
  function test() {
    return;
  }

  echo gettype(test()) . "\n";
  echo (test()?'true':'false') . "\n";
  echo (!test()?'true':'false') . "\n";
  echo (test() === false?'true':'false') . "\n";
?>
```

Câu lệnh *Include*

- Chèn code của một file khác vào trang PHP hiện tại.

File *vars.php*

```
<?php

$color = 'green';
$fruit = 'apple';

?>
```

File test.php

```
<?php

echo ’A $color $fruit’; // A

include 'vars.php';

echo ’A $color $fruit’; // A green apple

?>
```


Bài tập

- **Bài tập 1:**

  Viết 1 trang web có giá trị từ 1->20. Hãy xuất ra trình duyệt những số chẵn nằm trong khoảng 1->20 đó.

- **Bài tập 2:**

  Xây dựng 1 website thỏa yêu cầu xuất ra bảng cửu chương từ 2 ->10.

Xử lý giá trị form trong PHP

- Giúp tương tác xử lý dữ liệu trên form của người sử dụng.
- Cú pháp:

- **Action:** hành động chuyển tiếp đến link xử lý.
-

```
<form name="Tên form" action="link xử lý" method="Phương thức">
```


Ví dụ

- User nhập vào username rồi kích Sumit thì dữ liệu được chuyển tới trang check.php để tiến hành xử lý thông tin. Trên phương thức POST, với tên form là reg. Giá trị mà chúng ta gởi là username.

```
<form action = "check.php" method = post name=reg>
Please type your name here : <BR>
<input type = text name = "username"><BR><BR>
<input type = submit value = "Submit Data" name="add">
</form>
```

Cách lấy được giá trị vừa nhập

- PHP cho phép ta lấy giá trị dựa vào 2 phương thức POST và GET.
  - Đối với **POST** ta có : **$_POST[‘Giá trị’]**
  - Đối với **GET** ta có : **$_GET[‘Giá trị’]**
- Ví dụ trên

```
2  <?
3  echo "welcome, " . $_POST['username'] . "!";
4  ?>
5
```


Phương thức GET:

- Phương thức này cũng được dùng để lấy dữ liệu từ form nhập liệu. Tuy nhiên nhiệm vụ chính của nó vẫn là lấy nội dung trang dữ liệu từ web server.

- Ví dụ:

  Với url sau: shownews.php?id=50

  Vậy với trang shownews ta dùng hàm $_GET[‘id’] sẽ được giá trị là 50.

Phương thức POST

- Phương thức này được sử dụng để lấy dữ liệu từ form nhập liệu. Và chuyển chúng lên trình chủ webserver.

Bài tập

- Bài tập 1

  Xây dựng 1 trang HTML với nội dung gồm form nhập liệu họ và tên. Sau đó dùng 1 file php để xuất ra thông tin họ và tên mà người sử dụng vừa nhập liệu.

- Bài tập 2

  Tạo 1 trang web với hộp thoại nhập liệu username và password. Nếu người sử dụng nhập thông tin username/ password là admin/12345 thì xuất ra thông báo "welcome, admin" với kiểu chữ Tahoma, màu đỏ. Ngược lại nếu nhập sai thì xuất thông báo "Username hoặc password sai. Vui lòng nhập lại".

Hàm

- Mục đích xây dựng hàm
  - Tái sử dụng lại những đoạn mã giống nhau.
  - Tăng tính mềm dẻo, nhất quán trong ứng dụng, thời gian xây dựng và thiết kế ứng dụng.
- Các lợi ích
  - Chi phí
  - Độ tin cậy
  - Tính nhất quán

Sử dụng lại các Hàm

- Sử dụng hai hàm ***require()*** và ***include()*** để chèn các tệp *PHP, text, HTML và cả class PHP*
- Sự khác nhau giữa hàm ***require()*** và ***include()*** là gì?

Sự khác nhau giữa hàm ***require()*** và ***include()***

- Dùng Require:
  - Thông báo lỗi *"fatal"* và dừng thực thi script.
- Dùng Include:
  - Thông báo lỗi và tiếp tục thực thi.

Hàm

- Định nghĩa

```
<?php
    function tên_hàm([danh sách tham số ...])
    {
        [thân hàm ...]
    }
?>
```

- Gọi hàm

  Nhập tên_hàm (không phân biệt chữ in hoa-thường) và cung cấp đầy đủ các tham số cần thiết trong cặp dấu ()

Ví dụ Hàm

```
<?php
  function testing($a)
  {
                echo "Tham số là $a";
  } //end testing
  //Gọi hàm
  testing(123);
  testing("abc");
?>
```


Hàm

Kết thúc và trả kết quả

Lệnh ***return*** dùng để kết thúc và trả kết quả cũng như quyền điều khiển lại cho nơi đã gọi hàm. Nếu không có lệnh ***return*** thì mặc định hàm trả về giá trị NULL.

Muốn trả về ***hơn một giá trị*** thì phải ***dùng mảng***

Truyền tham số

Mặc định các tham số được truyền vào bên trong hàm theo phương pháp ***tham trị***. Trường hợp ***muốn thay đổi trực tiếp trên các tham số truyền*** thì người ta dùng phương pháp ***tham chiếu***, thêm dấu *&* trước tên tham số (khi định nghĩa) cũng như tên biến được truyền làm tham số(khi gọi hàm)

Ví dụ

```
<?php
   function binh_phuong($a)
       {
       $ketqua = $a * $a;
       return $ketqua;
       } //end testing

   echo binh_phuong(2);
?>
<?php
    function testing($a="mặc định")
    {
          echo "Tham số là $a";
    } //end testing
    testing();
?>
```


Hàm

- Tham số có giá trị mặc định

  Tương tự cách khai báo và gán giá trị đầu tiên cho biến, thông thường ***loại tham số này nên đặt cuối trong danh sách*** tham số

  Khi gọi hàm ***nếu bỏ trống tại vị trí tham số*** có giá trị mặc định thì mặc nhiên ***giá trị mặc định được dùng cho tham số đó***

- Hàm có số lượng tham số không xác định

  Khai báo danh sách tham số rỗng ()

  Sử dụng các hàm sau để lấy danh sách các tham số:

  func_num_args(): số lượng tham số khi hàm được gọi

  func_get_arg(i): giá trị các tham số thứ i được truyền (bắt đầu từ 0)

  func_get_args(): danh sách tất cả các tham số

Ví dụ

```
<?php
    function makecoffee($type = "cappuccino")
    {
        return "Making a cup of $type.\n";
    }

    echo makecoffee();
    echo makecoffee(null);
    echo makecoffee("espresso");
?>
```

Kết quả:

```
Making a cup of cappuccino.
Making a cup of .
Making a cup of espresso.
```


Ví dụ

```
<?php
    function foo()
    {
        $numargs = func_num_args();
        echo "Number of arguments: $numargs\n";
    }

    foo(1, 2, 3);
?>
```

Hàm

- Biến tĩnh
  Thêm từ khóa ***static*** khi khai báo biến
  Được khởi tạo (và gán giá trị) một lần đầu tiên duy nhất trong suốt quá trình thực thi của script

- Sử dụng biến toàn cục
  Khai báo lại biến toàn cục với từ khóa global (bên trong hàm) để có thể sử dụng được biến toàn cục này bên trong hàm
  Sử dụng các hàm sau để lấy danh sách các tham số:
  func_num_args(): số lượng tham số khi hàm được gọi
  func_get_arg(i): giá trị các tham số thứ i được truyền (bắt đầu từ 0)
  func_get_args(): danh sách tất cả các tham số

Ví dụ

```
function Test ()
  {
    static $a = 0;
    echo $a;
    $a++;
  }
```

- Với khai báo như trên , $a sẽ không mất đi giá trị sau khi gọi hàm Test() mà $a sẽ được tăng lên 1 sau mỗi lần gọi hàm Test().

Hàm

- **Phạm vi**
  Có giá trị sử dụng trong toàn script, ngay cả trước và sau khi định nghĩa
- **Lồng hàm**
  Cho phép định nghĩa lồng hàm, thậm chí lồng bên trong một cấu trúc điều khiển (if, switch, while/do, while…)
  Loại hàm này có phạm vi trong toàn script và không thể định nghĩa lại

Hàm

- **Hàm biến**
  Khi một biến kiểu chuỗi được khai báo và gán giá trị trùng khớp với tên một hàm được định nghĩa thì tên biến đó có thể được dùng như một cách gọi hàm khác với cách gọi hàm bình thường bằng tên hàm.
- **Một số hàm không thể dùng như hàm biến**
  - echo
  - print
  - var_dump
  - print_r
  - isset
  - unset
  - is_null
  - is_type

Ví dụ

```php
<?php
    function foo() {
       echo "In foo()<br />\n";
    }

    function bar($arg = '')
    {
       echo "In bar(); argument was '$arg'.<br />\n";
    }

    function echoit($string)
    {
       echo $string;
    }

    $func = 'foo';
    $func();        // This calls foo()

    $func = 'bar';
    $func('test');  // This calls bar()

    $func = 'echoit';
    $func('test');  // This calls echoit()
?>
```


Ví dụ hàm *echo*

*void **echo** (string $arg1 [, string $...])*

```php
<?php
// You can use variables inside of an echo statement
$foo = "foobar";
$bar = "barbaz";
echo "foo is $foo"; // foo is foobar
// You can also use arrays
$baz = array("value" => "foo");
echo "this is {$baz['value']} !"; // this is foo !
// Using single quotes will print the variable name, not the value
echo 'foo is $foo'; // foo is $foo
// Some people prefer passing multiple parameters to echo over concatenation.
echo 'This ', 'string ', 'was ', 'made ', 'with multiple parameters.', chr(10);
echo 'This ' . 'string ' . 'was ' . 'made ' . 'with concatenation.' . "\n";
echo <<<END
This uses the "here document" syntax to output
multiple lines with $variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon. no extra whitespace!
END;
// Because echo does not behave like a function, the following code is invalid.
// ($some_var) ? echo 'true' : echo 'false';
// However, the following examples will work:
($some_var) ? print 'true' : print 'false'; // print is also a construct, but
                                            // it behaves like a function, so
                                            // it may be used in this context.
echo $some_var ? 'true': 'false'; // changing the statement around
?>
```

Ví dụ hàm *print*

```php
<?php
print("Hello World");
print "print('abc'); also works without parentheses.";
print "This spans
multiple lines.
The newlines will be
output as well";
print "This spans\nmultiple lines. The newlines will be\noutput as well.";
print "escaping characters is done \"Like this\".";
// You can use variables inside of a print statement
$foo = "foobar";
$bar = "barbaz";
print "foo is $foo"; // foo is foobar
// You can also use arrays
$bar = array("value" => "foo");
print "this is {$bar['value']} !"; // this is foo !
// Using single quotes will print the variable name, not the value
print 'foo is $foo'; // foo is $foo
// If you are not using any other characters, you can just print variables
print $foo;          // foobar
print <<<END
This uses the "here document" syntax to output
multiple lines with $variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon no extra whitespace!
END;
?>
```


Ví dụ hàm *var_dump*

- Hiển thị kiểu dữ liệu và giá trị của nó

```php
<?php
$a = array(1, 2, array("a", "b", "c"));
var_dump($a);
?>
```

The above example will output:

```
array(3) {
  [0]=>
  int(1)
  [1]=>
  int(2)
  [2]=>
  array(3) {
    [0]=>
    string(1) "a"
    [1]=>
    string(1) "b"
    [2]=>
    string(1) "c"
  }
}
```

Ví dụ hàm *print_r*

- Hiển thị thông tin về một biến
- Cú pháp

  mixed **print_r** (mixed $expression [, bool $return = false])
  - Mixed : chỉ định như một biến chấp nhận nhiều kiểu dữ liệu
  - Nếu bạn muốn giữ lại kết quả đầu ra thì trả về cho một biến, tham số $return đặt là *True*.

Ví dụ

```
<pre>
<?php
$a = array ('a' => 'apple', 'b' => 'banana', 'c' => array ('x', 'y', 'z'));
print_r ($a);
?>
</pre>
```

The above example will output:

```
<pre>
Array
(
    [a] => apple
    [b] => banana
    [c] => Array
        (
            [0] => x
            [1] => y
            [2] => z
        )
)
</pre>
```

Hàm *isset* và *unset*

```php
<?php
$var = '';
// This will evaluate to TRUE so the text will be printed.
if (isset($var)) {
    echo "This var is set so I will print.";
}
// In the next examples we'll use var_dump to output
// the return value of isset().
$a = "test";
$b = "anothertest";
var_dump(isset($a));      // TRUE
var_dump(isset($a, $b)); // TRUE

unset ($a);
var_dump(isset($a));     // FALSE
var_dump(isset($a, $b)); // FALSE

$foo = NULL;
var_dump(isset($foo));   // FALSE
?>
```


Hàm *is_null*

- Trả về *True* nếu biến là Null, *False* trong các trường hợp khác.

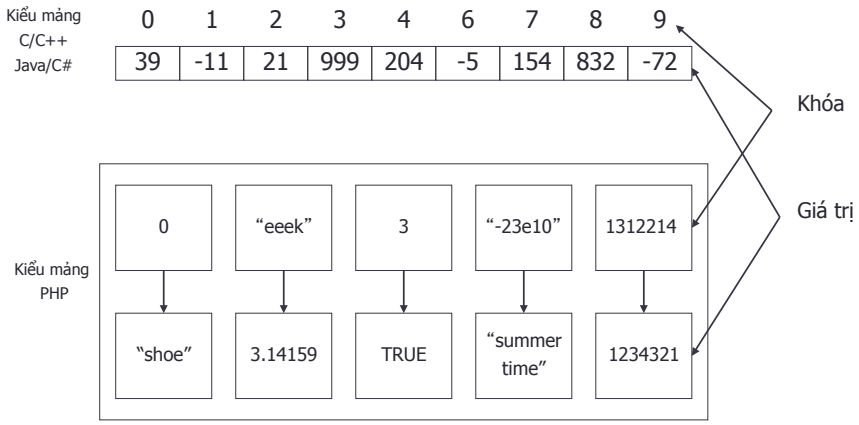
```php
<?php

error_reporting(E_ALL);

$foo = NULL;
var_dump(is_null($inexistent), is_null($foo));

?>
```

Mảng (*Array*)



Mảng

- Tạo mảng và gán giá trị

  $tên_biến = array([khóa =>]giá trị, [khóa =>]giá trị,…)

  Trường hợp không định nghĩa các khóa thì mảng sẽ được gán khóa mặc định theo kiểu số nguyên tăng dần bắt đầu từ 0

- Ví dụ
  - $a=array("Kenny","Maria","Julia","Kenvin");
  - $a= array (name => "Kenny", job => "Teacher", age=>"45", email => "webmaster@vietchuyen.com.vn")
- Tạo mảng từ một mảng có sẵn

  $tên_mảng_mới = $tên_mảng_cũ

Mảng

- Thêm một phần tử vào mảng
  $tên_mảng[khóa] = giá trị
  - Phần tử luôn được thêm vào cuối mảng
  - Nếu khóa đã tồn tại thì không có phần tử nào được thêm
  - Trường hợp không chỉ định khóa thì khóa sẽ được chọn bằng khóa có giá trị số nguyên lớn nhất cộng 1
- Xóa một phần tử khỏi mảng
  unset($tên_mảng[khóa])
- Đếm số phần tử của mảng
  count($tên_mảng)

Ví dụ

```php
<?php
$arr = array("foo" => "bar", 12 => true);

echo $arr["foo"]; // bar
echo $arr[12];    // 1
?>
```

```php
<?php
$arr = array("somearray" => array(6 => 5, 13 => 9, "a" => 42));

echo $arr["somearray"][6];    // 5
echo $arr["somearray"][13];   // 9
echo $arr["somearray"]["a"];  // 42
?>
```

Mảng

- Truy xuất
  $mảng[khóa]

  Khi dùng khóa chuỗi bên trong một chuỗi, không được dùng cặp ‘ ’ hoặc “”, nếu không thì phải đặt truy xuất bên trong cặp {}

  vd:
  // sai lỗi cú pháp
  echo "My PC has a $computer['processor'] processor
\n";
  echo "My PC has a $computer[""processor""] processor
\n";

  // đúng cú pháp nhưng không nên dùng
  echo "My PC has a $computer[processor] processor
\n";

  // cách dùng tốt nhất
  echo "My PC has a {$computer['processor']} processor
\n";

Ví dụ

```php
<?php
$arr = array(5 => 1, 12 => 2);

$arr[] = 56;    // This is the same as $arr[13] = 56;
                // at this point of the script

$arr["x"] = 42; // This adds a new element to
                // the array with key "x"

unset($arr[5]); // This removes the element from the array

unset($arr);    // This deletes the whole array
?>
```

Mảng

- Duyệt mảng với vòng lặp foreach

  foreach (mảng as [khóa =>] giá trị)

  Khối lệnh;

Ví dụ

```php
<?php
// Create a simple array.
$array = array(1, 2, 3, 4, 5);
print_r($array);

// Now delete every item, but leave the array itself intact:
foreach ($array as $i => $value) {
    unset($array[$i]);
}
print_r($array);

// Append an item (note that the new key is 5, instead of 0).
$array[] = 6;
print_r($array);

// Re-index:
$array = array_values($array);
$array[] = 7;
print_r($array);
?>
```

Mảng

Duyệt mảng với vòng lặp for

Tạo mảng khóa số nguyên trung gian

$mảng_khóa = array_keys($mảng)

Truy xuất thông qua mảng khóa

$mảng[$mảng_khóa[i]]

Mảng

Duyệt mảng với con trỏ mảng

Con trỏ mảng trỏ vào phần tử đầu tiên khi mảng được tạo ra

Các hàm di chuyển con trỏ mảng

- reset($mảng)
- end ($mảng)
- current($mảng) / pos($mảng)
- each($mảng)/next($mảng): di chuyển con trỏ tới vị trí tiếp theo.
- prev($mảng)

Mảng đa chiều

- Khai báo

$mảng_đa_chiều = array(khóa_1 => array(khóa_11 => …),
khóa_2 => array(khóa_12 => …),
…
khóa_n => array(khóa_1n => …))

- Truy xuất

$mảng_đa_chiều[khóa_1][khóa_2][…][khóa_n]

Ví dụ

```php
<?php
$fruits = array ( "fruits"  => array ( "a" => "orange",
                                       "b" => "banana",
                                       "c" => "apple"
                                     ),
                  "numbers" => array ( 1,
                                       2,
                                       3,
                                       4,
                                       5,
                                       6
                                     ),
                  "holes"   => array (      "first",
                                       5 => "second",
                                            "third"
                                     )
                );

// Some examples to address values in the array above
echo $fruits["holes"][5];    // prints "second"
echo $fruits["fruits"]["a"]; // prints "orange"
unset($fruits["holes"][0]);  // remove "first"

// Create a new multi-dimensional array
$juices["apple"]["green"] = "good";
?>
```

Các hàm xử lý mảng

- Sắp xếp

  Theo giá trị

  - sort($mảng) / asort($mảng) // tăng dần
  - rsort($mảng) / arsort($mảng)// giảm dần
  - natsort($mảng) / natcasesort($mảng) // tăng dần, dùng cho chuỗi
  - usort($mảng, "hàm_so_sánh") // tự định nghĩa thứ tự
  - uasort($mảng, "hàm_so_sánh") // tự định nghĩa thứ tự

  Theo khóa

  - ksort($mảng) // tăng dần
  - krsort($mảng) // giảm dần
  - uksort($mảng, "hàm_so_sánh") // tự định nghĩa thứ tự

Ví dụ *sort*

```php
<?php

$fruits = array("lemon", "orange", "banana", "apple");
sort($fruits);
foreach ($fruits as $key => $val) {
    echo "fruits[" . $key . "] = " . $val . "\n";
}

?>
```

The above example will output:

```
fruits[0] = apple
fruits[1] = banana
fruits[2] = lemon
fruits[3] = orange
```

Ví dụ *asort*

```php
<?php
$fruits = array("d" => "lemon", "a" => "orange", "b" => "banana", "c" => "apple");
asort($fruits);
foreach ($fruits as $key => $val) {
    echo "$key = $val\n";
}
?>
```

The above example will output:

```
c = apple
b = banana
d = lemon
a = orange
```


Các hàm xử lý mảng

- Nối ghép hai mảng
  array_merge($mảng1, $mảng2)
  array_combine($mảng1, $mảng2)
  array_intersect($mảng1, $mảng2)
- Tìm kiếm
  array_search($giá_trị, $mảng)

Chuỗi (*String*)

- Một chuỗi là một dãy các ký tự
- Một ký tự giống như một byte
- PHP không giới hạn kích thước kiểu chuỗi, nó chỉ phụ thuộc vào bộ nhớ mà PHP đang chạy.
- Có 4 cách để biểu diễn một chuỗi
  - Dùng dấu nháy đơn (')
  - Dùng dấu nháy kép (")
  - Dùng câu lệnh *"heredoc"*
  - Dùng câu lệnh *"nowdoc"* (PHP 5.3.0)

Dùng dấu nháy đơn (')

- Đây là cách đơn giản nhất để thể hiện chuỗi, đặt chuỗi cần hiện trong dấu nháy đơn
- Để hiển thị dấu nháy đơn trong chuỗi in ra thì dùng ký tự \ (backsplash) trước ký tự '
- Chú ý:
  - Các biến trong dấu nháy đơn và các ký tự đặc biệt cho xuống dòng sẽ không được PHP biên dịch trong dấu nháy đơn.

Ví dụ

```php
<?php
$expand = "variable";
echo 'this is a simple string';

echo 'You can also have embedded newlines in
strings this way as it is
okay to do';

// Outputs: Arnold once said: "I'll be back"
echo 'Arnold once said: "I\'ll be back"';

// Outputs: You deleted C:\*.*?
echo 'You deleted C:\\*.*?';

// Outputs: You deleted C:\*.*?
echo 'You deleted C:\*.*?';

// Outputs: This will not expand: \n a newline
echo 'This will not expand: \n a newline';

// Outputs: Variables do not $expand $either
echo 'Variables do not $expand $either';
?>
```


Dùng dấu nháy kép (“”)

- PHP sẽ biên dịch các biến và các ký tự đặc biệt ở trong dấu ngoặc kép.

Sequence	Meaning
\n	linefeed (LF or 0x0A (10) in ASCII)
\r	carriage return (CR or 0x0D (13) in ASCII)
\t	horizontal tab (HT or 0x09 (9) in ASCII)
\v	vertical tab (VT or 0x0B (11) in ASCII) (since PHP 5.2.5)
\f	form feed (FF or 0x0C (12) in ASCII) (since PHP 5.2.5)
\\	backslash
\$	dollar sign
\"	double-quote
\[0-7]{1,3}	the sequence of characters matching the regular expression is a character in octal notation
\x[0-9A-Fa-f]{1,2}	the sequence of characters matching the regular expression is a character in hexadecimal notation

Ví dụ

```
<?php
$ex= "variable";
echo "This is a simple string";

// Outputs: This will  expand: \n a newline
echo "This will not expand: \n a newline";

// Outputs: Variables do  $expand $either
echo "Variables do not $expand $either";
?>
```

```
This is a simple stringThis will not expand:
 a newlineVariables do not
```


Dùng câu lệnh *"heredoc"*

- Là cách thứ 3 phân định một chuỗi.
- Câu lệnh:
  - <<< từ_khóa_nhận_diện

    Các dòng văn bản ở đây

    từ_khóa_nhận_diện;

- Ví dụ

```
<?php
$str = <<<EOD
        Example of string
        spanning multiple lines
        using heredoc syntax.
EOD;
echo $str;
?>
```

Dùng câu lệnh *"nowdoc"*

- Cú pháp giống *"heredoc", nhưng ký tự nhận diện được thêm ''*
- Chức năng: giống như dùng dấu nháy đơn (' ')
- Ví dụ:

```
<?php
$str = <<<'EOD'
Example of string
spanning multiple lines
using nowdoc syntax.
EOD;
echo $str;
?>
```


Các hàm xử lý chuỗi

- Các xử lý cơ bản
  - trim($chuỗi, [' ký tự muốn cắt'])
  - ltrim($chuỗi, [' ký tự muốn cắt'])
  - rtrim($chuỗi, [' ký tự muốn cắt'])
  - strlen($chuỗi)
  - substr($chuỗi, $vị trí, $chiều_dài)
  - strtoupper ($chuỗi)
  - strtolower ($chuỗi)
  - iconv(mã nguồn, mã đích, $chuỗi)

Ví dụ hàm trim()

```php
<?php

$text   = "\t\tThese are a few words :) ...  ";
$binary = "\x09Example string\x0A";
$hello  = "Hello World";
var_dump($text, $binary, $hello);

print "\n";

$trimmed = trim($text);
var_dump($trimmed);

$trimmed = trim($text, " \t.");
var_dump($trimmed);

$trimmed = trim($hello, "Hdle");
var_dump($trimmed);

// trim the ASCII control characters at the beginning and end of $binary
// (from 0 to 31 inclusive)
$clean = trim($binary, "\x00..\x1F");
var_dump($clean);

?>
```


Ví dụ hàm iconv() -> convert chuỗi theo loại mã nào đó

```php
<?php
$text = "This is the Euro symbol '€'.";

echo 'Original : ', $text, PHP_EOL;
echo 'TRANSLIT : ', iconv("UTF-8", "ISO-8859-1//TRANSLIT", $text), PHP_EOL;
echo 'IGNORE   : ', iconv("UTF-8", "ISO-8859-1//IGNORE", $text), PHP_EOL;
echo 'Plain    : ', iconv("UTF-8", "ISO-8859-1", $text), PHP_EOL;

?>
```

The above example will output something similar to:

```
Original : This is the Euro symbol '€'.
TRANSLIT : This is the Euro symbol 'EUR'.
IGNORE   : This is the Euro symbol ''.
Plain    :
Notice: iconv(): Detected an illegal character in input string in .\iconv-example.php on line 7
This is the Euro symbol '
```

Các hàm xử lý chuỗi

- Tìm kiếm
  - strpos($chuỗi, $chuỗi_con, [$vị_trí_bắt đầu])
  - strrpos ($chuỗi, $chuỗi_con, [$vị_trí_bắt đầu])
- So sánh
  - strcmp($chuỗi_1, $chuỗi_2)
  - strncmp($chuỗi_1, $chuỗi_2, $chiều_dài)
  - strcasecmp($chuỗi_1, $chuỗi_2)
  - strncasecmp($chuỗi_1, $chuỗi_2, $chiều_dài)
  - strnatcmp($chuỗi_1, $chuỗi_2)
  - strnatcasecmp($chuỗi_1, $chuỗi_2)

Ví dụ sử dụng hàm strpos()

```php
<?php
$mystring = 'abc';
$findme   = 'a';
$pos = strpos($mystring, $findme);

// Note our use of ===.  Simply == would not work as expected
// because the position of 'a' was the 0th (first) character.
if ($pos === false) {
    echo "The string '$findme' was not found in the string '$mystring'";
} else {
    echo "The string '$findme' was found in the string '$mystring'";
    echo " and exists at position $pos";
}
?>
```

PHP và Unicode

Thiết lập môi trường trong php.ini

Thiết lập	Giá trị
extension	php_mbstring.dll
extension_dir	"đường dẫn đến thư mục extension của php"
mbstring.language	Neutral
mbstring.internal_encoding	UTF-8
mbstring.encoding_translation	
mbstring.http_input	UTF-8
mbstring.http_output	UTF-8
mbstring.substitute_character	?
mbstring.func_overload	7

Lập trình hướng đối tượng: *class* và *object*

- ***Lập trình hướng thủ tục***
  - *là cách lập trình để giải quyết vấn đề theo hướng giải quyết từng bước một đến khi đạt được kết quả.*
  - Kiểu lập trình hướng thủ tục còn được gọi là kiểu lập trình từ trên xuống hoặc lập trình theo hàm (function)
- ***Lập trình hướng thủ tục gồm 2 bước:***
  - Xử lý vấn đề
  - Xây dựng hàm và tối ưu mã nguồn

Lập trình hướng đối tượng: *class* và *object*

- ***Lập trình hướng đối tượng***
  - Là kiểu lập trình dựa trên một nền tảng các class đã được xây dựng sẵn
  - Chúng ta phải xác định trước những gì sẽ phải làm, những trường hợp sẽ xảy ra để xây dựng lớp có những chức năng cần thiết cho quá trình xây dựng ứng dụng.

Lập trình hướng đối tượng: *class* và *object*

- Khai báo lớp

```
class tên_lớp
{
        các thuộc tính và phương thức
}
```

- Tạo và hủy một đối tượng

```
$tên_biến = new tên_lớp();
```

Đối tượng sẽ tự động bị hủy khi không còn tham chiếu nào đến nó

```
$tên_biến = NULL;
```

Đặc trưng OOP

- Tính kế thừa
- Tính đa hình
- Tính đóng gói
- Tính trừu tượng

Ví dụ 1

```php
<?php
class SimpleClass
{
    // property declaration
    public $var = 'a default value';

    // method declaration
    public function displayVar() {
        echo $this->var;
    }
}
?>
```

Ví dụ 2

```php
<?php
class A
{
    function foo()
    {
        if (isset($this)) {
            echo '$this is defined (';
            echo get_class($this);
            echo ")\n";
        } else {
            echo "\$this is not defined.\n";
        }
    }
}

class B
{
    function bar()
    {
        // Note: the next line will issue a warning if E_STRICT is enabled.
        A::foo();
    }
}
```


Ví dụ 2 (tiếp)

```php
$a = new A();
$a->foo();

// Note: the next line will issue a warning if E_STRICT is enabled.
A::foo();
$b = new B();
$b->bar();

// Note: the next line will issue a warning if E_STRICT is enabled.
B::bar();
?>
```

The above example will output:

```
$this is defined (A)
$this is not defined.
$this is defined (B)
$this is not defined.
```

Lập trình hướng đối tượng: ***member*** và ***method***

- Các từ khóa khai báo

  public: có thể sử dụng bên ngoài lớp
  private: chỉ sử dụng cục bộ bên trong lớp
  protected: sử dụng được bởi các lớp kế thừa

- Một số quy tắc chung

  Không thể khai báo hai method trùng tên
  Method phải được khai báo ngay bên trong khai báo lớp
  Dùng biến giả **$this** để truy xuất các member và method trong lớp
  Dùng toán tử -> để truy xuất đến ***member*** và method

Ví dụ 1

```
<?php
    class Cart {
      var $items;  // Items in our shopping cart

      // Add $num articles of $artnr to the cart

      function add_item ($artnr, $num) {
          $this->items[$artnr] += $num;
      }

      // Take $num articles of $artnr out of the cart
    function remove_item ($artnr, $num) {
        if ($this->items[$artnr] > $num) {
            $this->items[$artnr] -= $num;
            return true;
        } else {
            return false;
        }
      }
    }
?>
```

- Lớp Cart ở đây là một kiểu dữ liệu, vì vậy bạn có thể tạo một biến có kiểu này với toán tử new

```
$cart = new Cart;
$cart->add_item("10", 1);
```

Ví dụ 2

```php
<?php
class SimpleClass
{
   // invalid member declarations:
   public $var1 = 'hello ' . 'world';
   public $var2 = <<<EOD
hello world
EOD;
   public $var3 = 1+2;
   public $var4 = self::myStaticMethod();
   public $var5 = $myVar;

   // valid member declarations:
   public $var6 = myConstant;
   public $var7 = array(true, false);

   // This is allowed only in PHP 5.3.0 and later.
   public $var8 = <<<'EOD'
hello world
EOD;
}
?>
```


Lập trình hướng đối tượng: ***constructor*** và ***detructor***

- Khai báo

public function __construct(danh sách tham số)
{
    khởi tạo giá trị các member;
}

constructor được tự động thực hiện khi đối tượng được tạo

public function __destruct()
{
    dọn dẹp;
}

destructor được tự động thực hiện khi đối tượng bị hủy

Ví dụ

```php
<?php
class MyDestructableClass {
   function __construct() {
       print "In constructor\n";
       $this->name = "MyDestructableClass";
   }

   function __destruct() {
       print "Destroying " . $this->name . "\n";
   }
}

$obj = new MyDestructableClass();
?>
```


Lập trình hướng đối tượng: *constant*

- Khai báo

  const TÊN_HẰNG = giá trị;

- Truy xuất

  tên_lớp::TÊN_HẰNG // ngoài lớp
  self::TÊN_HẰNG // trong lớp

Ví dụ

```php
<?php
class MyClass
{
    const constant1 = 'constant value';

    function showConstant() {
        echo  self::constant1 . "<br/>";
    }
}

echo MyClass::constant1 . "<br/>";

$classname = "MyClass";
echo $classname::constant1; // As of PHP 5.3.0

$class = new MyClass();
$class->showConstant();

//echo $class::constant1; // As of PHP 5.3.0
?>
```


Lập trình hướng đối tượng:
static member và *static method*

- ***static member*** và ***static method*** cho phép truy cập các thuộc tính và các phương thức không cần phải tạo một bản sao của class.
- Biến *$this* không cung cấp cho các *static member* và *static method*
- Các thuộc tính không được truy cập qua toán tử ->

Lập trình hướng đối tượng: *static member*

- Khai báo

  … static $thuộc_tính

- Truy xuất

  tên_lớp::$thuộc_tính // ngoài lớp
  self::$thuộc_tính // trong lớp

Lập trình hướng đối tượng: *static method*

- Khai báo

  … static function phương_thức(…)

- Truy xuất

  tên_lớp::phương_thức(…) // ngoài lớp
  seft:: phương_thức(…) // trong lớp

Lập trình hướng đối tượng: *thừa kế (inherit)*

- Khai báo lớp con

```
class lớp_con extends lớp_cha
{
    các thuộc tính và phương thức
}
```

Tất cả các member và method được khai báo public hay protected trong lớp cha được thừa kế và có thể sử dụng trong lớp con

Lập trình hướng đối tượng: *phương thức nạp chồng*

- Gọi một method lớp cha

```
parent::phương_thức(…)
```

Bằng cách định nghĩa lại một phương thức đã có ở lớp cha, tất cả các lời gọi đến phương thức này mà không chỉ định rõ như trên sẽ được hiểu là gọi phương thức có cùng tên của lớp con

Lập trình hướng đối tượng: *sự đa hình*

- Khai báo lớp trừu tượng

```
abstract class lớp_trừu_tượng
{
    // các thuộc tính
    abstract public function phương_thức_trừu_tượng(…);
    …
    // các phương thức khác
}
```

Không thể tạo đối tượng trực tiếp từ lớp trừu tượng
Lớp con bắt buộc phải định nghĩa các phương thức trừu tượng của lớp cha

Lập trình hướng đối tượng: *ngăn kế thừa và nạp chồng*

- Lớp không thể kế thừa

```
final class không_thể_kế_thừa { … }
```

- Phương thức không thể nạp chồng

```
final public function không_thể_nạp_chồng(…) { … }
```

Lập trình hướng đối tượng: *interface*

- Khai báo Interface

```
interface giao_diện
{
    public function phương_thức();
    …
}
```

- Khai báo lớp theo mẫu Interface

```
abstract class tên_lớp implements giao_diện
{
    …
}
```

Các lớp sử dụng Interface hay kế thừa từ một lớp sử dụng Interface bắt buộc phải định nghĩa tất cả các phương thức trong Interface đó

Lập trình hướng đối tượng: *một số lưu ý*

- Phép gán đối tượng

```
$a = new lớp();
$b = $a;
// $a và $b cùng trỏ đến một thực thể của lớp
```

- Nhân bản đối tượng

```
$b = clone $a      // $b được tạo mới và sao chép giá trị từ $a
```

phương thức __clone():
sau khi sao chép toàn bộ các giá trị từ $a vào $b, phương thức này sẽ được từ động gọi nếu được định nghĩa trong lớp của $a và $b

Xây dựng chương trình Upload

- ***Bước 1: Xác định thuộc tính và phương thức***
- ***Bước 2: Xây dựng khung sườn cho lớp***
- ***Bước 3: Xây dựng chi tiết các phương thức xử lý***
- ***Bước 4: Sử dụng lớp***

Bước 1: Xác định thuộc tính và phương thức

- ***Thuộc tính***
  - Tên của tập tin ($_fileName)
  - Kích thước tối đa được phép upload ($_fileSize)
  - Phần mở rộng của các tập tin được phép upload ($_fileExtension)
  - Tập tin tạm trước khi upload lên server ($_fileTmp)
  - Thư mục chứa tập tin upload ($_uploadDir)

Bước 1: Xác định thuộc tính và phương thức

- ***Phương thức:***
  - Phương thức khởi tạo (__construct())
  - Phương thức thiết lập kích thước tập tin upload (setFileSize())
  - Phương thức thiết lập phần mở rộng của tập tin upload (setFileExtension())
  - Phương thức thiết lập thư mục upload tập tin (setUploadDir())
  - Phương thức kiểm tra tất cả các điều kiện của tập tin upload (isVail())
  - Phương thức upload tập tin (upload())

Bước 2: Xây dựng khung sườn cho lớp

```
class Upload{
// Khai báo các biến
var $_fileName;
…
// Khai báo các method
function upload($rename =  false, $prefix = 'file_')
{
}
…
}
```

Bước 3: Xây dựng chi tiết các phương thức xử lý

```
class Upload{
    //Ham upload tap tin
    function upload($rename =  false, $prefix = 'file_')
    {
    if($rename ==  false){
            $source = $this->_fileTmp;
            $dect = $this->_uploadDir . $this->_fileName;
            }
    else{
            $source = $this->_fileTmp;
            $dect =     $this->_uploadDir . $prefix . time() . '.' . $this->_fileExtension;

            }
             copy($source,$dect);
    }
}
```


Bước 4: Sử dụng lớp

```
<?php
include 'upload.class.php';
$upload  = new Upload('picture');
$upload->setUploadDir('images/');
if($upload->isVail() == false)
{       echo "<pre>";
        print_r($upload->_errors);
        echo "</pre>";
}else
{
        $upload->upload(true,'pic_');
}
?>
```

MYSQL

MySQL

- Là một hệ quản trị cơ sở dữ liệu mở được dùng phổ biến hiện nay
- Dữ liệu trong MySQL được lưu trữ trong các CSDL dưới dạng các bảng
- Bảng dữ liệu bao gồm mỗi quan hệ giữa các thực thể,các hàng và các cột.
- Một CSDL có thể chứa một hoặc nhiều bảng

Các kiểu dữ liệu cơ bản trong MySQL

Kiểu	Mô tả
char(length)	tối đa 255 ký tự, chiều dài cố định = *length*
varchar(length)	tối đa 255 ký tự, chiều dài động <= *length*
text	tối đa 65536 ký tự
int(length)	-2.147.483.648 đến +2.147.483.647
decimal(length,dec)	tối đa *length* chữ số trong đó *dec* chữ số thập phân

Các kiểu dữ liệu thông dụng trong MySQL

Kiểu	Mô tả
enum(“option1”, “option2”,…)	tập hợp tự định, nghĩa tối đa 65.535 giá trị
date	yyyy-mm-dd
time	hh:mm:ss
datetime	yyyy-mm-dd hh:mm:ss

Các lệnh thông dụng trong MySQL

Kiểu	Mô tả
CREATE	tạo CSDL hoặc bảng
ALTER	thay đổi bảng có sẵn
SELECT	chọn dữ liệu từ bảng
DELETE	xóa dữ liệu khỏi bảng
DESCRIBE	xem thông tin mô tả về cấu trúc bảng
INSERT INTO	ghi giá trị vào bảng
UPDATE	cập nhật dữ liệu đã có trong bảng
DROP	xóa bảng hay toàn bộ CSDL

Các lệnh thông dụng trong MySQL

CREATE INDEX *indexname* ON *tablename* (column [ASC|DESC], ...);

CREATE PROCEDURE *procedurename*([*parameters*]) BEGIN ... END;

CREATE TABLE *tablename*
(
    column datatype [NULL|NOT NULL] [CONSTRAINTS],
    column datatype [NULL|NOT NULL] [CONSTRAINTS],
    ...
);

CREATE USER *username*[*@hostname*] [IDENTIFIED BY [PASSWORD] '*password*'];

CREATE [OR REPLACE] VIEW *viewname* AS SELECT ...;

Các lệnh thông dụng trong MySQL

ALTER TABLE *tablename*
(
    ADD column datatype [NULL|NOT NULL] [CONSTRAINTS],
    CHANGE column columns datatype [NULL|NOT NULL] [CONSTRAINTS],
    DROP column,
    ...
);

Các lệnh thông dụng trong MySQL

SELECT *columnname*, ...
FROM *tablename*, ...
[WHERE ...]
[UNION ...]
[GROUP BY ...]
[HAVING ...]
[ORDER BY ...];

Các lệnh thông dụng trong MySQL

DELETE FROM *tablename*
[WHERE ...];

Các lệnh thông dụng trong MySQL

DESCRIBE *tablename* [*columnname* | *wild*]

Các lệnh thông dụng trong MySQL

INSERT INTO *tablename* [(columns, ...)]
VALUES(values, ...);

INSERT INTO *tablename* [(columns, ...)]
SELECT columns, ... FROM *tablename*, ...
[WHERE ...];

Các lệnh thông dụng trong MySQL

UPDATE *tablename*
SET *columname* = value, ...
[WHERE ...];

Các lệnh thông dụng trong MySQL

DROP DATABASE|INDEX|PROCEDURE|
TABLE|TRIGGER|USER|VIEW itemname;

Giao tiếp dòng lệnh

- Kết nối mysql server

mysql [-h *hostname*] [-P *portnumber*] -u *username* -p

mysql [-h hostname] [-P portnumber] --user=*user* --
password=*pass*

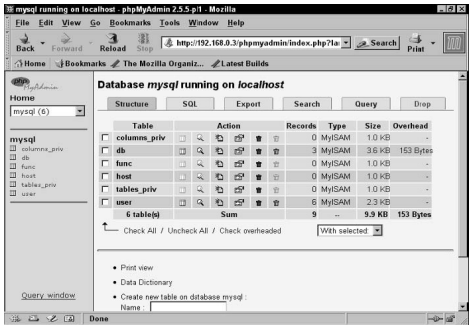
Nhập lệnh sau dấu nhắc lệnh *mysql>*

Mỗi lệnh SQL kết thúc bằng dấu ;

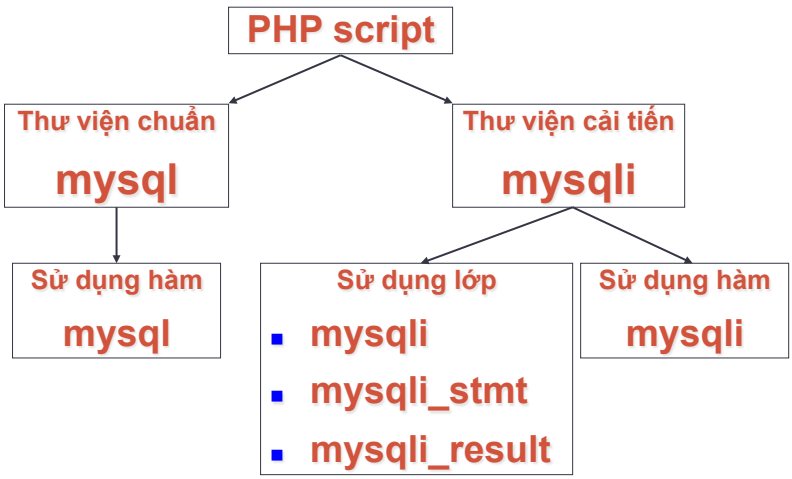
Giao tiếp đồ họa

Một số công cụ thông dụng

MySQL Query Browser
phpMyAdmin
MySQL Maestro
Navicat
MySQL Manager



Kết nối MySQL từ PHP



Thư viện mysql cải tiến trong PHP5

- Thiết lập trong php.ini

  extension=php_mysqli.dll

- Ưu điểm
  - Hỗ trợ lập trình hướng đối tượng
  - Hỗ trợ nhân bản và phân tán CSDL
  - Nén và mã hóa dữ liệu trên kết nối
  - Tối ưu hiệu năng và mã

Quy trình kết nối vào MySQL

1. Mở kết nối đến CSDL
2. Chọn CSDL
3. Chọn bảng mã (nếu cần)
4. Xử lý CSDL
5. Dọn dẹp
6. Đóng kết nối

Bước 1: Mở kết nối đến CSDL

```
// OOP mysqli
$mysqli = new mysqli('hostname', 'username',
                                        'password', 'dbname');

// mysqli
$link = mysqli_connect('hostname', 'username',
                                         'password', 'dbname');

// mysql
$link = mysql_connect('hostname', 'username', 'password');
```


Bước 2: Chọn CSDL

```
// OOP mysqli
$mysqli->select_db('dbname');

// mysqli
mysqli_select_db($link, 'dbname');

// mysql
mysql_select_db('dbname', $link);
```

Bước 3: Chọn bảng mã (nếu cần)

```
// OOP mysqli
$mysqli->query("SET NAMES 'character set' ")

// mysqli
mysqli_query($link, "SET NAMES 'character set' ")

// mysql
mysql_query("SET NAMES 'character set' ", $link)
```


Bước 4: Xử lý CSDL

- Truy vấn

```
// OOP mysqli
$result = $mysqli->query("query")

// mysqli
$result = mysqli_query($link, "query")

// mysql
$result = mysql_query("query", $link)
```

Bước 4: Xử lý CSDL (tt)

- Lấy dữ liệu từ truy vấn

```
// OOP mysqli
$row = $result->fetch_row()
$row = $result->fetch_assoc()
$row = $result->fetch_array(result_type)

// mysqli
$row = mysqli_fetch_row($result)
$row = mysqli_fetch_assoc($result)
$row = mysqli_fetch_array($result, result_type)

// mysql
$row = mysql_fetch_row($result)
$row = mysql_fetch_assoc($result)
$row = mysql_fetch_array($result, result_type)
```


Bước 5: Dọn dẹp

```
// OOP mysqli
$result->close()

// mysqli
mysqli_free_result($result)

// mysql
mysql_free_result($result)
```

Bước 6: Đóng kết nối

```
// OOP mysqli
$mysqli->close()

// mysqli
mysqli_close($link)

// mysql
mysql_close($link)
```


Ví dụ 1: chỉ đọc một bản ghi

```
<?
$db = mysql_connect("localhost","root");
mysql_select_db("test",$db);
$result = mysql_query("select * from tblNhanVien",$db);
printf("Id: %s <br>\n",mysql_result($result,0,"Id"));
printf("First Name: %s <br>\n",mysql_result($result,
    0,"FirstName"));
printf("Last Name: %s <br>\n",mysql_result($result,
    0,"LastName"));
printf("Address: %s <br>\n",mysql_result($result,0,"Address"));
printf("Positione: %s <br>\n",mysql_result($result,
    0,"Position"));
mysql_close($db);
?>
```

Ví dụ 2: Đọc toàn bộ dữ liệu sử dụng hàm *mysql_fetch_row*

```
<?php
$db = mysql_connect("localhost", "root");
mysql_select_db("test",$db);
$result = mysql_query("SELECT * FROM tblNhanVien",$db);

echo "<table border=1>\n";
echo "<tr><td>Name</td><td>Position</tr>\n";
while ($myrow = mysql_fetch_row($result)) {
  printf("<tr><td>%s %s</td><td>%s</td></tr>\n", $myrow[1],
     $myrow[2], $myrow[3]);
}
echo "</table>\n";
mysql_close($db);
?>
```


Ví dụ 3: Đọc toàn bộ dữ liệu sử dụng hàm *mysql_fetch_array*

```
<?php
$db = mysql_connect("localhost", "root") or die("Khong ket noi duong
    CSDL");
mysql_select_db("test",$db);
$result = mysql_query("SELECT * FROM tblNhanVien",$db);
if ($myrow = mysql_fetch_array($result))
{
 do
    {
      printf("<a href=\"%s?id=%s\">%s %s</a><br>\n", $PHP_SELF,
       $myrow["Id"], $myrow["FirstName"], $myrow["LastName"]);
    }
while ($myrow = mysql_fetch_array($result));
} else
{
 echo "Sorry, no records were found!";
 }
?>
```

Tương tác giữa web browser và web server

HTML Form

```
<form action=’đường dẫn đến file script’
   method=’cách gửi dữ liệu đến script’>
   <Các thành phần input>
</form>
```

- Đường dẫn đến file script phải là đường dẫn tương đối
- Trường hợp script nằm cùng file với FORM thì khai báo action=”<?php echo $_SERVER[PHP_SELF] ?>”
- Có hai cách gửi dữ liệu là GET và POST

HTML FORM: Các thành phần INPUT

- INPUT Text

  <input type="text" name="tên_biến" value="giá trị">

- INPUT Checkbox

  <input type= "checkbox" name="tên_biến" value="giá trị" [checked]>

- INPUT Radio

  <input type=”radio” name=”tên_biến” value=”giá trị” [checked]>

- INPUT Submit

  <input type="submit" name="tên_biến" value="giá trị">

HTML FORM: Các thành phần INPUT

- List Box

  <select name=”tên_biến”>
  <option value=”giá trị 1”>chuỗi hiển thị 1</option>
  …
  <option value=”giá trị n”>chuỗi hiển thị n</option>
  </select>

- Multiline List Box

  <select multiple name=”tên_biến_mảng[]”>
  <option value=”giá trị 1”>chuỗi hiển thị 1</option>
  …
  <option value=”giá trị n”>chuỗi hiển thị n</option>
  </select>

Chuyển dữ liệu với GET và POST

Nếu thiết lập register_globals = ON trong file cấu hình php.ini thì tất cả dữ liệu của các thành phần input trong form trở thành biến toàn cục có cùng tên và được truy xuất trực tiếp không cần thông qua các mảng siêu toàn cục

Kể từ phiên bản PHP 4.2.0, register_globals được mặc định thiết lập là OFF vì một số lý do về bảo mật

Trường hợp không quan tâm đến cách chuyển dữ liệu thì có thể dùng biến mảng siêu toàn cục $_REQUEST trong cả hai trường hợp dùng GET và POST

Chuyển dữ liệu với GET và POST

GET	POST
• Dữ liệu được gắn thêm vào URL khi gọi script • Các dữ liệu được đưa vào biến mảng siêu toàn cục $_GET với khóa tương ứng với tên các thành phần input trong form • Nên dùng trong trường hợp dữ liệu chỉ dùng để truy vấn, không đòi hỏi bảo mật • Không hỗ trợ uploading file • Chỉ hỗ trợ bảng mã ASCII chuẩn	• Dữ liệu được nhúng vào trong HTTP request khi gởi đến server • Các dữ liệu được đưa vào biến mảng siêu toàn cục $_POST với khóa tương ứng với tên các thành phần input trong form • An toàn hơn so với khi dùng GET nên được dùng phổ biến hơn • Hỗ trợ uploading file • Hỗ trợ nhiều bảng mã

Chuyển hướng người dùng

```
header("Location: http://www.example.com/");

header('Location: http://'. $_SERVER['HTTP_HOST']
                          .rtrim(dirname($_SERVER['PHP_SELF']), '/\\')
                          . "/" . $relative_url);
```

Bài tập

Viết một script thực hiện công việc sau:

- Yêu cầu người dùng nhập vào một số nguyên
- Đem so sánh số vừa nhập với một số nguyên cho trước
- Nếu bằng thì xuất ra câu chúc mừng người dùng đã đoán đúng con số bí mật
- Ngược lại thì thì thông báo kết quả là con số vừa nhập là lớn hay bé hơn con số bí mật và yêu cầu nhập lại cho đến khi nhập đúng