Autore:	Luciano Viviani
Classe:	TERZA INFORMATAICA SERALE (3IS)
Anno scolastico:	2003/2004
Scuola:	Itis Euganeo

MANUALE ESSENZIALE PHP

Manuale

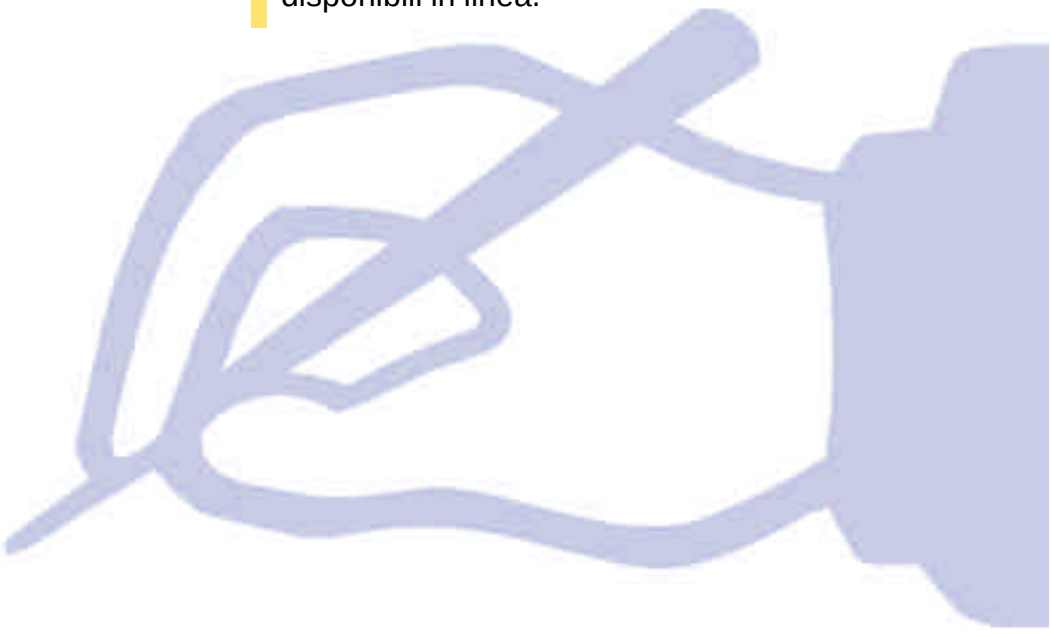
Questa dispensa riassume la sintassi essenziale del linguaggio PHP e le sue funzioni di interfaccia

con il database MySql. Lo scopo di questa dispensa, rivolta agli studenti del quinto anno della

specializzazione Informatica, è di fornire quei mezzi essenziali che permettano loro di realizzare

applicazioni client-server in una rete Internet-Intranet con accesso ai dati contenuti in un database.

Per ogni approfondimento il riferimento d'obbligo è ai relativi manuali disponibili in linea.



Introduzione

Questo testo ha lo scopo di descrivere l'uso del linguaggio PHP in particolare per il suo impiego con database MySql. Non contiene quindi una descrizione completa del linguaggio ma le sue parti ritenute essenziali. Come prerequisito è richiesta una conoscenza del linguaggio C++ cui si fa riferimento in svariati punti. Il manuale completo può essere scaricato dal sito www.php.it.
Gli esempi non sono stati verificati, possono quindi contenere degli errori: il manuale è stato realizzato per puro scopo didattico, chi usa PHP in applicazioni commerciali faccia riferimento alla documentazione ufficiale. Non sono in genere citate varianti del linguaggio utilizzate in versioni precedenti alla 4.1.0.

Sintassi del linguaggio

Uso del PHP

Normalmente i file PHP contengono parti scritte in HTML e altre parti scritte in PHP contenute tra speciali Tag. Quando l'interprete PHP inizia ad esaminare un file, visualizzerà il contenuto del file sino a quando non incontra uno dei tag speciali indicanti l'inizio del codice da interpretare come istruzioni PHP. A questo punto il parser eseguirà tutto il codice trovato sino a quando non incontrerà i tag di chiusura, che indicano al parser di tornare alla modalità di visualizzazione: tutto ciò che si trova all'esterno dei tag PHP sarà lasciato inalterato, mentre tutto ciò che si trova all'interno sarà eseguito come codice PHP.

Modi per uscire dalla modalità HTML

Esistono 4 set di tag che possono essere utilizzati per delimitare blocchi di codice PHP. Soltanto due di questi (`<?php. . .?>` e `<script language="php">. . .</script>`) sono sempre disponibili; gli altri possono essere attivati o disattivati tramite il file di configurazione php.ini. Sebbene i tag brevi o quelli in stile ASP possano essere pratici, il supporto di questi non è garantito in tutte le versioni. Quindi, se si intende inserire codice PHP all'interno di testi XML o XHTML, occorre utilizzare `<?php. . .?>` per essere conformi allo standard XML.
I tag supportati dal PHP sono:

```
1) <?php echo("se si vogliono produrre documenti XHTML o XML, si utilizzi
   questo modo\n"); ?>
2) <? echo ("questo è il più semplice, ovvero come istruzione SGML\n"); ?>
   <?= espressione ?>  Questa è un'abbreviazione per "<? echo espressione ?>"
3) <script language="php"> echo ("alcuni editor (tipo FrontPage) non amano le
   istruzioni di elaborazione");</script>
4) <% echo ("Opzionalmente puoi utilizzare tag nello stile ASP"); %>
   <%= $variable; # Questo &egrave; una abbreviazione per "<%echo .." %>
```

Il primo, `<?php. . .?>` , è il metodo preferenziale, dato che permette l'utilizzo del PHP all'interno di codice conforme a specifiche XML come XHTML.
Il secondo metodo è disponibile solo se sono stati abilitati i tags abbreviati. Ciò può essere impostato sia utilizzando la funzione `short_tags()` (solo PHP 3), sia abilitando nel file di configurazione del PHP l'opzione `short_open_tag`, oppure compilando il PHP utilizzando l'opzione `--enable-short-tags` nel comando configure. Sebbene siano abilitati nel php.ini-dist rilasciato, l'uso dei tag brevi è vivamente sconsigliato.
Il quarto modo è disponibile solo se sono stati attivati nel file di configurazione i tag in stile ASP tramite l'opzione asp_tags.

Nota: Il supporto per i tag nello stile ASP è stato aggiunto nella versione 3.0.4.
Nota: L'utilizzo dei tag brevi dovrebbe essere evitato nello sviluppo di applicazioni o librerie destinate alla distribuzione o destinati a server di produzione PHP di cui non si ha il controllo poiché questi tag potrebbero non essere attivi sul server di destinazione. Per avere maggiore portabilità, codice redistribuibile, occorre essere certi di non utilizzare i tag brevi.

Il tag di chiusura di un blocco include il carattere di 'a capo' immediatamente seguente, se presente. Inoltre, il tag di chiusura viene considerato automaticamente come punto e virgola; pertanto non occorre inserire il punto e virgola alla fine dell'ultima riga del blocco php.
Il PHP permette strutture tipo le seguenti:

```
<?php
if ($expression) {
    ?>
    <strong>Questa è vera.</strong>
    <?php
} else {
    ?>
    <strong>Questa è falsa.</strong>
    <?php
}
?>
```

Questo esempio agisce come atteso, poiché il PHP rileva il tag di chiusura ?>, e da questo punto, inizia a dare in output tutto ciò che trova fino a quando non rileva un'altro tag di apertura. Certamente l'esempio dato è macchinoso, ma per l'output di grossi blocchi di testo, l'uscire dalla modalità di parsing PHP, è generalmente più efficiente piuttosto che inviare il testo tramite ripetute funzioni echo() o print().

Separazione delle istruzioni

Le istruzioni sono separate come nel C o in Perl - ogni istruzione termina con un punto e virgola.
Il tag di chiusura (?>) implica anche la fine di un'istruzione, perciò le seguenti sono equivalenti:

```
<?php
    echo "Questo &grave; un test";
?>
<?php echo "Questo &grave; un test" ?>
```

Commenti

Il PHP supporta i commenti dei linguaggi 'C', 'C++' e della shell Unix. Per esempio:

```
<?php
    echo "Questo è un test"; // Questo è un commento su una linea C++
    /* Questo è un commento su più; linee
       ancora un'altra linea di commento */
    echo "Questo è un altro test";
    echo "Un ultimo test"; # Questo è un commento stile shell Unix
?>
```

Lo stile di commento su "una linea", attualmente commenta solo fino alla fine della linea o del blocco corrente di codice PHP.

```
<h1>Questo è un <?# echo "semplice";?> esempio.</h1>
<p>L'intestazione qui sopra dirà; 'Questo è un esempio'.
```

Occorre prestare attenzione nel non annidare i commenti di stile C, situazione che si presenta quando si commentano larghi blocchi di codice.

```
<?php
 /*
    echo "Questo è un test"; /* Questo commento causerà dei problemi */
 */
?>
```

Lo stile di commento su linea singola commenta il testo fino alla fine della riga oppure alla fine del blocco di codice PHP, dipende da cosa si incontra prima.

Tipi di dati

Introduzione

PHP supporta otto tipi primitivi:
Quattro scalari: `boolean, integer, float (in realtà double), string`
Due tipi composti: `array, object`
Due tipi speciali: `resource, NULL`
Il tipo di una variabile non è solitamente stabilito dal programmatore (Come in C o Java), ma è deciso da PHP in base al contesto in cui la variabile è usata.
Esempio:

```
<?php
$booleano = TRUE;   // un boolean
$stringa  = "Ciao";  // una string
$int  = 12;     // un integer
?>
```

Si può forzare la conversione di una variabile in un determinato tipo con l'operatore di cast (come in C o Java) o con la funzione `settype()`; da notare che la stessa variabile può assumere valori diversi in certe situazioni, in base al tipo che è in quel contesto.

Boolean

Può assumere i due valori TRUE o FALSE, è stato introdotto nella versione 4
Sintassi
Per introdurre una variabile booleana basta assegnarle il valore TRUE o FALSE (maiuscole o minuscole)

```
<?php
$test = True; // assegna il valore TRUE alla variabile $test
?>
```

Solitamente un valore booleano viene restituito da operatori di confronto in una struttura di controllo

```
<?php
if ($action == "show_version") {
    echo "La versione è 1.23";
}
?>
```

Conversione in boolean
Per convertire esplicitamente un valore in boolean si usano gli operatori di cast (bool) o (boolean), i seguenti valori sono considerati FALSE:

- il boolean FALSE
- l'integer 0 (zero)
- il float 0.0 (zero)
- la stringa vuota e la stringa "0"
- un array con zero elementi
- un oggetto con zero variabili membro
- NULL

Tutti gli altri valori sono considerati TRUE

```
<?php
echo gettype((bool) "");        // bool(false)
echo gettype((bool) 1);         // bool(true)
echo gettype((bool) -2);        // bool(true)
echo gettype((bool) "foo");     // bool(true)
echo gettype((bool) 2.3e5);     // bool(true)
echo gettype((bool) array(12)); // bool(true)
echo gettype((bool) array());   // bool(false)
?>
```

Integer

Un Integer è un numero intero di lunghezza dipendente dal sistema operativo, in genere 32 bit con segno, può essere specificato in base 10, 16, 8 eventualmente preceduto dal segno. I numeri che iniziano con una cifra diversa da 0 vengono considerati decimali, se iniziano con 0 ottali, se iniziano con 0x esadecimali

```
<?php
$a = 1234; # numero decimale
$a = -123; # numero negativo
$a = 0123; # numero ottale (equivalente a 83 decimale)
$a = 0x1A; # numero esadecimale (equivalente a 26 decimale)
?>
```

Overflow degli interi
Se specificate un numero oltre i limiti del tipo intero questo verrà interpretato come float, lo stesso accade per il risultato di una operazione tra interi

```
<?php
$large_number =  2147483647;
var_dump($large_number);
// output: int(2147483647), non supera il limite (32 bit)
$large_number = 2147483648;
var_dump($large_number);
// output: float(2147483648), supera il limite
// lo stesso vale per numeri in formato esadecimale
var_dump( 0x80000000 );
// output: float(2147483648)
$million = 1000000;
$large_number =  50000 * $million;
var_dump($large_number);
// output: float(50000000000, risultato dell'operazione
?>
```

Non esiste in PHP un operatore di divisione intera, il risultato è sempre float, l'operatore di cast `(int)` lo tronca, per arrotondare al valore più vicino si usa la funzione `round().`

```
<?php
var_dump(25/7);         // float(3.5714285714286)
var_dump((int) (25/7)); // int(3)
var_dump(round(25/7));  // float(4)
?>
```

Conversione ad intero
La conversione ad intero avviene automaticamente se un operatore, una funzione o una struttura di controllo richiede un argomento intero.
Da un boolean FALSE ritorna 0 e TRUE ritorna 1, le stringhe vengono interpretate come interi nei casi che vedremo successivamente, per altri tipi occorre cautela.

Float

Sono specificati usando i seguenti tipo di sintassi:

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

La dimensione di float dipende dal sistema operativo, normalmente viene usato il formato IEEE a 64 bit che fornisce all'incirca un valore massimo di 1.8e308 con una precisione di circa 14 cifre decimali. Dato che numeri decimali come 0.1 o 0.7 o frazioni come 1/3 non possono essere convertiti esattamente nella rappresentazione float binaria senza una piccola perdita di precisione *non confrontate mai l'uguaglianza di due numeri float.*

String

Una stringa è una serie di caratteri: in PHP la rappresentazione è a 8 bit, ci sono quindi esattamente 256 caratteri diversi. Questo implica che non c'è supporto nativo per i codici Unicode, qualche supporto è fornito dalle funzioni `utf8_encode()` e `utf8_decode()`. La lunghezza di una stringa è indefinita.

Sintassi

Una stringa costante può essere specificata in tre modi:

Tra apici (virgolette semplici)

Se volete inserire un apice nella stringa dovete farlo precedere da backslash (`\'`) , il carattere backslash può essere inserito raddoppiandolo(`\\`). In questo formato non possono essere inseriti caratteri speciali come `\n` o `\t`.

Tra virgolette

Se la stringa è racchiusa tra virgolette (**"**) al suo interno si possono usare i caratteri speciali del linguaggio C, in particolare:

```
\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)
\\                backslash
\$                dollar sign
\"                double-quote
\[0-7]            codice ottale del carattere  (da 1 a 3 cifre)
\x[0-9A-Fa-f]     codice esadecimale (1 o 2 cifre)
```

La caratteristica più importante delle stringhe racchiuse tra virgolette è che le variabili vengono espanse, cioè sostituite con il loro valore

Heredoc

In PHP 4 è stata aggiunta un'altra possibilità per scrivere stringhe costanti, utile quando la stringa è molto lunga e quindi si vuole scrivere su più linee, in questo caso la stringa va inserita tra

```
<<<identificatore
identificatore;
```

da notare che l'identificatore di chiusura deve necessariamente iniziare nella prima colonna di una riga ed essere identico a quello di apertura. All'interno si possono usare caratteri speciali e le variabili vengono espanse come con le virgolette.

```
<?php
$nome='Luciano';
$str = <<<STRINGA
Esempio di stringa
che si espande su più righe
usando la sintassi heredoc
Posso inserire la variabile
$nome che verrà sostituita
dalla stringa Luciano
STRINGA;
?>
```

Accesso ai singoli caratteri di una stringa

I caratteri di una stringa si comportano come gli elementi di un vettore, il primo carattere ha indice 0, l'indice in PHP 4 va racchiuso tra parentesi graffe `{}` anche se si possono usare quelle quadre degli array (sintassi deprecata);

```
<?php
$str = 'Stringa di prova';
$first = $str{0};// Prende il primo carattere di una stringa
$last = $str{strlen($str)-1}; // Prende l'ultimo carattere di una stringa
?>
```

Funzioni e operatori utili per le stringhe

Le stringhe possono essere concatenate con l'operatore punto '`.`', da notare che il + usato per esempio da Java non funziona. Ci sono inoltre molte funzioni utili per modificare le stringhe.

Conversione in stringa

Si può convertire un valore in stringa usando il cast `(string)` o la funzione `strval()`.

La conversione in stringa avviene automaticamente con le funzioni echo() o print() o quando un valore è confrontato con una stringa. TRUE viene convertito in '1' e FALSE nella stringa vuota (in questo modo è possibile la conversione inversa). Un numero intero o float è convertito nella sua rappresentazione.
Il contenuto di un Array non è convertito automaticamente in stringa, si ottiene la stringa 'Array' ma possono esserlo i suoi elementi, in modo simile si comportano gli Object.
NULL è sempre convertito nella stringa vuota.

Conversione di stringhe in numeri

Se una stringa contiene all'inizio un valore numerico in un formato valido viene convertita in numero.

```
<?php
$foo = 1 + "10.5";                // $foo è float (11.5)
$foo = 1 + "-1.3e3";              // $foo è float (-1299)
$foo = 1 + "bob-1.3e3";           // $foo è integer (1)
$foo = 1 + "bob3";                // $foo è integer (1)
$foo = 1 + "10 Small Pigs";       // $foo è integer (11)
$foo = 4 + "10.2 Little Piggies"; // $foo è float (14.2)
$foo = "10.0 pigs " + 1;          // $foo è float (11)
$foo = "10.0 pigs " + 1.0;        // $foo è float (11)
?>
```

Array

Un array in PHP è una mappa ordinata, cioè un tipo che associa una chiave a un valore. Viene creato con il seguente costrutto sintattico:

```
array( [key =>] value,...);
```

dove `key` può essere un intero o una stringa e `value` un qualsiasi valore, compreso un altro array.

```
<?php
$arr = array("foo" => "bar", 12 => true);
echo $arr["foo"]; // bar
echo $arr[12];    // 1
?>
```

Se non specificate alcuna chiave viene preso il massimo indice intero aumentato di 1, se specificate una chiave che esiste già il valore è sovrascritto.

```
<?php
// Questi due array sono gli stessi
array(5 => 43, 32, 56, "b" => 12);
array(5 => 43, 6 => 32, 7 => 56, "b" => 12);
?>
```

Attenzione!

Da PHP 4.3.0 il comportamento è cambiato per le chiavi costituite da numeri negativi, ora se il valore massimo è negativo il nuovo valore è 0 (prima era il numero negativo massimo più uno)
Un array può essere modificato scrivendo in esso valori esplicitamente con la sintassi:

`$arr[key]=valore;`

Se l'array non esiste viene creato
Se key non è specificata viene preso l'ultimo indice intero aumentato di 1, per rimuovere un elemento si usa la funzione `unset().`

```
<?php
$arr = array(5 => 1, 12 => 2);
$arr[] = 56;    // Come $arr[13] = 56;
$arr["x"] = 42; // aggiunge un nuovo elemento con chiave x
unset($arr[5]); // rimuove l'elemento
unset($arr);    // Cancella l'intero array
?>
```

Conversione in array

Per ciascuno dei tipi: integer, float, string, boolean e resource, se convertite il valore in array ottenete un array con un solo elemento con indice 0.
Gli array sono ordinati, l'ordine può essere cambiato con svariate funzioni di ordinamento, la funzione `count()` ne conta gli elementi.

Object

Il PHP consente di creare oggetti e di usarli in modo simile a C++ e Java, per esempio:

```
<?php
class oggetto
{
    function eseguiOggetto()
    {
        echo "Ho eseguito oggetto";
    }
}//definizione dalla classe
$bar = new oggetto;//creo il nuovo oggetto
$bar->eseguiOggetto();// e eseguo il suo metodo
?>
```

Se si converte un'altra variabile in un oggetto questo viene creato e contiene, se la variabile non è vuota, una variabile membro di nome scalar.

```
<?php
$obj = (object) 'ciao';
echo $obj->scalar;  // scrive 'ciao'
?>
```

Resource

È una variabile speciale che contiene un riferimento a una risorsa esterna creata e usata da funzioni speciali, per esempio un puntatore a un file aperto, una connessione a un database e così via. Nessun altro valore può essere convertito in una risorsa. PHP 4 libera automaticamente la risorse non usate, ad eccezione delle connessioni persistenti a un database.

NULL

Il valore speciale NULL rappresenta una variabile senza alcun valore ed è l'unico valore possibile per il tipo NULL introdotta in PHP 4
Una variabile è considerata NULL se:
gli è stata assegnata la costante NULL;
non gli è stato ancora assegnato un valore;
su di essa è stata applicata la funzione unset().

Variabili

Le variabili in PHP sono rappresentate dal simbolo di dollaro seguito dal nome della variabile, il nome è case-sensitive.
Fino alla versione 3 le variabili sono sempre assegnate per valore, cioè quando una espressione viene assegnata ad una variabile il risultato è copiato nella variabile stessa.
PHP 4 permette anche l'assegnazione per riferimento, ciò significa che la nuova variabile è un sinonimo (alias) di quella originale: modificandone una modifico anche l'altra. In questo modo si guadagna in velocità e in spazio in memoria, in particolare con variabili molto grandi. L'assegnazione per riferimento avviene ponendo prima della variabile il simbolo &.

```
<?php
$a = 'Luciano';
$b = &$a;              // a viene assegnata per riferimento ad a
$b = "Il mio nome è $b";
echo $a;     //viene scritto "Il mio nome è Luciano"
?>
```

Da notare che solo le variabili con nome possono essere assegnate per riferimento, $a=&(3+2); non è valida. Non è nemmeno concesso far riferimento al valore di ritorno di una funzione.

Variabili predefinite

PHP fornisce un gran numero di variabili predefinite, molte di queste tuttavia dipendono dal particolare server in esecuzione, dalla sua versione e dalla sua configurazione.

Dalla versione 4.1.0 PHP fornisce inoltre un insieme di array predefiniti contenenti variabili del server web, questi array sono sempre disponibili e sono noti come *autoglobals* o *superglobals*.

`$GLOBALS`
Contiene un riferimento a ciascuna variabile globale correntemente disponibile, le chiavi sono i nomi delle variabili.
`$_SERVER`
Variabili settate dal server web o direttamente correlate all'ambiente di esecuzione dello script corrente.
`$_GET`
Variabili fornite dal metodo GET HTTP.
`$_POST`
Variabili fornite dal metodo POST HTTP.
`$_COOKIE`
Variabili fornite dai cookie HTTP.
`$_FILES`
Variabili fornite all'upload di file.
`$_ENV`
Variabili fornite dall'ambiente.
`$_REQUEST`
Variabili fornite dai metodi di input GET, POST e COOKIE
`$_SESSION`
Variabili di sessione correntemente registrate

Variabili esterne a PHP

Vengono usate per leggere il contenuto di una form con il metodo GET e POST, per esempio una form HTML potrebbe essere la seguente:

```
<form action="ricevi.php" method="POST">
    Nome:  <input type="text" name="username"><br>
    Email: <input type="text" name="email"><br>
    <input type="submit" name="submit" value="Invia">
</form>
```

Ci sono vari modi con cui il file ricevi.php può ottenere i dati, eccone alcuni:

```
<?php
// Disponibile da PHP 4.1.0
   print $_POST['username'];
   print $_REQUEST['username'];
   import_request_variables('p', 'p_');
   print $p_username;
?>
```

L'accesso ai dati di una form che usa il metodo GET è simile, usando le variabili predefinite.
Nota: Se nel nome di una variabile esterna viene usato il punto PHP lo converte in un trattino basso (underscore)

Nomi di variabili con IMAGE SUBMIT

In una form è possibile usare un'immagine al posto del pulsante submit con la seguente sintassi:

```
<input type="image" src="image.gif" name="sub">
```

quando l'utente "clicca" con il mouse in un punto dell'immagine, la form viene trasmessa con la variabile di nome `sub` e due variabili addizionali: `sub_x` e `sub_y` che contengono le coordinate del punto all'interno dell'immagine.

HTTP Cookie

PHP supporta in modo trasparente i cookie come definiti dalle specifiche Netscape. I cookie sono creati con la funzione `setcookie()`, dato che i cookie fanno parte dell'header HTTP la funzione deve essere chiamata prima di inviare qualsiasi output al browser.
I dati dei cookie sono disponibili negli array di dati `$_COOKIE, $HTTP_COOKIE_VARS` e `$_REQUEST`.

Variabili static

Una variabile statica non perde il suo valore quando l'esecuzione di una funzione termina, possono essere usate per realizzare contatori, per esempio:

```
<?php
function Test()
{
    static $a = 0;
    echo $a;
    $a++;
}
?>
```

In questo caso il valore di `$a` viene incrementato e scritto ogni volta che la funzione viene chiamata, l'assegnazione `$a=0;` è eseguita solo la prima volta che la funzione è chiamata. Ad una variabile statica non può essere assegnata una espressione, per esempio `static $a=1+1;` non è un'espressione valida.

Variabili variabili

Talvolta è conveniente avere nomi di variabili che possono essi stessi cambiare, l'assegnazione viene fatta con un doppio simbolo di dollaro, per esempio:

```
<?php
$$a = "ciao";
?>
```

crea una nuova variabile con nome `$ciao` a cui ci si può riferire con `${$a}`.

Costanti

Una costante è un identificatore (nome) per un valore. Come si può intuire, tale valore non può cambiare durante l'esecuzione dello script (fanno eccezione le costanti magiche, che, in realtà, non sono costanti). Una costante è "case-sensitive" per default. È convenzione comune che i nomi di costante siano sempre maiuscoli.
Come le superglobals, costante è sempre globale. Si può accedere alle costanti da qualsiasi punto dello script senza tenere conto della visibilità.
È possibile definire una variabile utilizzando la funzione `define()`. Una volta definita, a una costante non è possibile cambiare il valore o eliminarla.
Le costanti possono contenere solo dati di tipo scalare (`boolean, integer, float` e `string`).
Per ottenere il valore di una costante è sufficiente specificarne il nome. A differenza delle variabili, non è necessario anteporre il simbolo $ al nome di una costante. Si può anche utilizzare la funzione `constant()`, per leggere il valore di una costante, nel caso in cui se ne ottenga dinamicamente il nome. Si utilizzi `get_defined_constants()` per ottenere una lista delle costanti definite.
Se si utilizza il nome di una costante che non è definita, PHP assume che detto valore sia il nome della costante stessa, come se si fosse inserito il testo nel nome . Quando ciò accade PHP segnala il problema con un `E_NOTICE`. Per sapere se una costante è definita, si può utilizzare la funzione `defined()`.
Di seguito sono riportate le principali differenze rispetto le variabili:

- Le costanti non iniziano con il segno del dollaro ($);
- Le costanti possono essere definite solo con la funzione `define()` e non tramite assegnazione;
- Le costanti possono essere definite e utilizzate ovunque senza seguire le regole di visibilità;
- Una volta impostate, le costanti non possono essere ridefinite né annullate;
- Le costanti possono essere solo valori scalari;

```
<?php
define("COSTANTE", "Ciao mondo.");
echo COSTANTE; // stampa "Ciao mondo."
echo Costante; // stampa "Costante" e genera una notice.
?>
```

Costanti predefinite
Il PHP mette a disposizione ad ogni script diverse costanti predefinite. Alcune di queste, tuttavia, sono create dai vari moduli, e, pertanto, saranno disponibili solo quando questi moduli sono caricati, sia dinamicamente sia staticamente.
Esistono quattro costanti magiche il cui valore cambia in base al contesto in cui sono utilizzate. Ad esempio, il valore di `_LINE_` dipende da quale linea si trova nel momento in cui è richiamata. Queste costanti speciali

sono 'case-insensitive' e sono:

Nome	Descrizione
LINE	Il numero di linea corrente.
FILE	Il nome e percorso assoluto del file.
FUNCTION	Nome della funzione. (Aggiunta nel PHP 4.3.0.)
CLASS	Nome della classe. (Aggiunta nel PHP 4.3.0.)
METHOD	Nome del metodo della classe (Aggiunta nel PHP 5.0.0.)

Espressioni

Il modo più semplice ma accurato per definire un'espressione è "qualsiasi cosa che ha un valore".
Una particolarità di PHP è di essere un *linguaggio orientato alle espressioni*, quasi tutto è un'espressione. Consideriamo per esempio una semplice assegnazione:

```
$a = 5;
```

Si vede facilmente che ci sono due valori coinvolti, la costante 5 e la variabile `$a`, ma in realtà c'è un valore in più coinvolto: l'assegnazione stessa, cioè la stringa '`$a=5`' che assume anch'essa il valore 5.Per esempio queste tre assegnazioni sono equivalenti, per quanto riguarda i valori assunti dalle variabili `$a` e `$b`:

```
$b=($a=5);
$a=5; $b=5;
$b=$a=5;
```

Si ricorda che le assegnazioni sono valutate da destra a sinistra.
In genere le espressioni contengono operatori, per esempio operatori di relazione (==, !=, >,...) che forniscono come risultato TRUE o FALSE, operatori aritmetici (+ , -, *) che possono essere combinati con l'assegnazione come nel linguaggio C (`$a *=5;`) e l'operatore condizionale ternario:

```
<?php
$primo ? $secondo : $terzo
?>
```

se il valore della prima espressione è TRUE (diverso da 0) viene valutata la seconda espressione altrimenti viene valutata la terza.

Operatori

Gli operatori sono simili a quelli del linguaggio C++, ci limiteremo a descrivere quelli che non esistono in C++ o quelli che funzionano in modo diverso.
Richiamiamo due importanti proprietà degli operatori: la precedenza e l'associatività:
La precedenza stabilisce l'ordine in cui l'espressione viene valutata, per esempio `3+5*5` viene valutata 5*5=25+3=28 perché l'operatore di moltiplicazione ha la precedenza su quello di somma. Le parentesi alterano la precedenza, in caso di dubbio usatele!
L'associatività invece si riferisce invece all'ordine in cui vengono eseguiti operatori con la medesima precedenza, per esempio gli operatori aritmetici di pari precedenza, che hanno associatività sinistra vengono eseguiti da sinistra verso destra, mentre gli operatori di cast o incremento vengono eseguiti da destra verso sinistra (per esempio `(float) 3+2+5;` esegue nell'ordine 3+2 poi al risultato viene aggiunto 5 e infine avviene la conversione in float.

Associatività	Operatori, quelli con precedenza più bassa sono elencati per primi
sinistra	,
sinistra	or
sinistra	xor
sinistra	and
destra	print
sinistra	= += -= *= /= .= %= &= |= ^= ~= <<= >>=
sinistra	? :
sinistra	||
sinistra	&&

sinistra	|
sinistra	^
sinistra	&
non associativi	== != === !==
non associativi	< <= > >=
sinistra	<< >>
sinistra	+ - .
sinistra	* / %
destra	! ~ ++ -- (int) (float) (string) (array) (object) @
destra	[
non associativi	new

Operatori di controllo errori

PHP supporta un operatore di controllo dell'errore: il carattere at (@). Quando prefisso ad una espressione in PHP, qualunque messaggio di errore che potesse essere generato da quella espressione sarà ignorato.

Se la caratteristica `track_errors` è abilitata, qualsiasi messaggio di errore generato dall'espressione sarà salvato nella variabile globale `$php_errormsg`. Questa variabile sarà sovrascritta ad ogni errore, così controllatela subito se volete usarla.

```
<?php
/* Errore di file intenzionale */
$my_file = @file ('file_inesistente') or
    die ("Apertura del file fallita: l'errore è '$php_errormsg'");
// questo funziona per qualsiasi espressione, non solo funzioni:
$value = @$cache[$key];
// non verrà generata una notifica se l'indice $key non esiste.
?>
```

Operatori di esecuzione

PHP supporta un operatore di esecuzione: backticks (``). Notare che quelli non sono apostrofi! PHP cercherà di eseguire il contenuto dei backticks come comando di shell; sarà restituito l'output (non sarà semplicemente esportato come output; può essere assegnato ad una variabile).

```
$output = `ls -al`;//lista il contenuto di una directory Unix
echo "<pre>$output</pre>";
```

Nota: L'operatore backtick è disabilitato quando è abilitata safe mode oppure quando è disabilitata `shell_exec()`.

Operatori di stringa

Ci sono due operatori di stringa. Il primo è l'operatore di concatenazione ('.'), che restituisce la concatenazione dei suoi argomenti a destra e a sinistra. Il secondo è l'operatore di assegnazione concatenata ('.='), che aggiunge alla fine dell'argomento sul lato destro l'argomento sul lato sinistro.

```
$a = "Ciao ";
$b = $a . "Mondo!"; // ora $b contiene "Ciao Mondo!"
$a = "Ciao ";
$a .= "Mondo!";     // ora $a contiene "Ciao Mondo!"
```

Strutture di controllo

Qualsiasi script PHP è costituito da una serie di istruzioni. Una istruzione può essere un'assegnazione, una chiamata di funzione, un loop, una istruzione condizionale che non fa nulla (istruzione vuota). Le istruzioni terminano con un punto e virgola. Inoltre, le istruzioni si possono raggruppare in blocchi di istruzioni racchiudendole tra parentesi graffe. Un gruppo di istruzioni è, a sua volta, un'istruzione. Anche in questo caso ci limiteremo a descrivere le differenze rispetto al linguaggio C++;

Sintassi alternativa per le strutture di controllo

PHP offre una sintassi alternativa per alcune delle sue strutture di controllo; vale a dire, `if`, `while`, `for`, `foreach` e `switch`. Fondamentalmente la sintassi alternativa consiste nel sostituire la prima parentesi graffa con il carattere "duepunti" (:) e la seconda parentesi graffa con `endif`, `endwhile`, `endfor`, `endforeach`, oppure `endswitch`, rispettivamente.

```
<?php if ($a == 5): ?>
a è uguale a 5
```

```
<?php endif; ?>
```

Nell'esempio precedente, il blocco HTML "a è uguale a 5" è incluso nel ramo if scritto utilizzando la sintassi alternativa. Il blocco HTML verrà visualizzato solamente se $a è uguale a 5. Questa sintassi è utile quando si vogliono produrre in modo condizionale due spezzoni di pagina HTML (o due intere pagine HTML).

foreach

PHP 4 (non PHP 3) permette l'uso della struttura di controllo `foreach` (per ogni elemento), alla stessa maniera del linguaggio Perl e altri. Ciò semplicemente fornisce una facile metodo per attraversare un array. Esistono due possibili notazioni sintattiche; la seconda è un'utile estensione della prima:

```
foreach(array_expression as $value) istruzione;
foreach(array_expression as $key => $value) istruzione;
```

La prima attraversa l'array dato da `array_expression`. Ad ogni ciclo, si assegna il valore dell'elemento corrente a `$value` e il puntatore interno avanza di una posizione (in modo tale che al ciclo successivo l'elemento corrente sarà il successivo elemento dell'array).
La seconda esegue lo stesso ciclo con la differenza che il valore dell'indice corrente viene assegnato ad ogni ciclo, alla variabile `$key`.
Nota: All'inizio dell'esecuzione di un ciclo `foreach` il puntatore interno viene automaticamente posizionato nella prima posizione. Questo significa che non è necessario utilizzare la funzione `reset()` prima di un ciclo `foreach`.
Nota: È importante notare che `foreach` opera su una copia dell'array, non sull'array stesso, pertanto il puntatore dell'array originale non viene modificato come accade utilizzando la funzione `each()` e le modifiche agli elementi dell'array non appaiono nell'array originale.

break

`break` termina l'esecuzione di una struttura `for`, `foreach`, `while`, `do..while` o `switch`.
break accetta un argomento opzionale che definisce, nel caso di cicli annidati, il livello del ciclo che è da interrompere.

```
/* Uso dell'argomento opzionale. */
$i = 0;
while (++$i) {
    switch ($i) {
    case 5:
        echo "At 5<br>\n";
        break 1;  /* Interrompe solo switch. */
    case 10:
        echo "At 10; quitting<br>\n";
        break 2;  /* Interrompe switch e while. */
    default:
        break;
    }
}
```

continue

Si utilizza per interrompere l'esecuzione del ciclo corrente e continuare con l'esecuzione all'inizio del ciclo successivo; `continue` accetta un argomento numerico opzionale che definisce, nel caso di cicli annidati, il numero di cicli da interrompere e da cui iniziare l'esecuzione dell'iterazione successiva.

declare

Il costrutto `declare` si usa per definire direttive di esecuzione per blocchi di istruzioni. La sintassi è simile alla sintassi di altre strutture di controllo:

```
declare (direttiva) istruzione
```

La sezione direttiva permette di impostare il comportamento del blocco `declare` . Attualmente è riconosciuta una sola direttiva: la direttiva `ticks`.

Tick

Un tick è un evento che si verifica per ogni N istruzioni di basso livello eseguite dal parser all'interno del blocco `declare`. Il valore per N viene specificato usando `ticks=N` all'interno della sezione direttiva del blocco `declare`.
L'evento (o gli eventi) che si verifica su ogni tick è specificato usando `register_tick_function()`. Vedere l'esempio più in basso per ulteriori dettagli. Notare che può verificarsi più di un evento per ogni tick.

```
<?php
// Una funzione che registra il tempo quando viene chiamata
function profile ($dump = FALSE)
{
    static $profile;
    // Restituisce il tempo in $profile, successivamente lo cancella
    if ($dump) {
        $temp = $profile;
        unset ($profile);
        return ($temp);
    }
    $profile[] = microtime ();
}
// Imposta un tick handler
register_tick_function("profile");
// Inizializza la funzione prima del blocco declare
profile ();
// Esegue un blocco di codice, attraverso un tick ogni seconda istruzione
declare (ticks = 2) {
    for ($x = 1; $x < 50; ++$x) {
        echo similar_text (md5($x), md5($x*$x)), "<br />";
    }
}
// Mostra i dati immagazzinati nel profilo
print_r (profile (TRUE));
?>
```

L'esempio descrive il codice PHP all'interno del blocco `declare`, registrando il tempo in cui è stata eseguita ogni seconda istruzione di basso livello. Questa informazione può poi essere usata per trovare le aree lente all'interno di particolari segmenti di codice. Questo processo può essere ottimizzato usando altri metodi: usando i tick è più conveniente e facile da implementare.
I tick sono ben adeguati per il debugging, l'implementazione di semplici multitasking, backgrounded I/O e molti altri compiti.

`require()`

L'istruzione `require()` include e valuta il file specifico. Informazioni dettagliate su come funziona quest'inclusione sono descritte nella documentazione di `include()`.
`require()` e `include()` sono identiche in ogni senso eccetto per come esse trattano gli errori. `include()` produce un Warning mentre `require()` restituisce un Fatal Error. In altre parole, non esitate ad usare `require()` se volete che un file mancante fermi l'esecuzione della pagina. `include()` non si comporta in questo modo, lo script continuerà nonostante tutto. Assicuratevi di avere un appropriato `include_path` impostato a dovere.

```
<?php
require 'prepend.php';
require $qualche_file;
require ('qualche_file.txt');
?>
```

`include()`

L'istruzione `include()` include e valuta il file specificato.
La documentazione seguente si applica anche a `require()`. I due costrutti sono identici in ogni aspetto eccetto per come essi trattano gli errori.
Quando un file viene incluso, il codice che esso contiene eredita lo scope delle variabili della riga in cui si verifica l'inclusione. Qualsiasi variabile disponibile in quella riga nella chiamata al file sarà disponibile all'interno del file chiamato, da quel punto in avanti.

```
vars.php
<?php
$colore = 'verde';
$frutto = 'mela';
?>
test.php
```

```
<?php
echo "Una $frutto $colore"; //Output: Una
include 'vars.php';
echo "Una $frutto $colore"; //Output: Una mela verde
?>
```

Quando un file viene incluso, il parsing esce dalla modalità PHP e entra in modalità HTML all'inizio del file incluso, e riprende alla fine. Per questa ragione, qualunque codice all'interno del file incluso che dovrebbe essere eseguito come codice PHP deve essere incluso all'interno dei tag PHP validi di apertura e chiusura.
Se "URL fopen wrappers" nel PHP sono abilitati (come nella configurazione di default), potete specificare il file da includere usando un URL (via HTTP) invece che un percorso locale.
Poichè `include()` e `require()` sono speciali costrutti di linguaggio, dovete includerli all'interno di blocchi di istruzioni se si trovano in un blocco condizionale.

```
<?php
// Questo NON VA BENE e non funzionerà come desiderato.
if ($condizione)
    include $file;
else
    include $un_altro;
// Questo è CORRETTO.
if ($condizione) {
    include $file;
} else {
    include $un_altro;
}
?>
```

`require_once()`

L'istruzione `require_once()` include e valuta il file specificato durante l'esecuzione dello script. È un comportamento simile all'istruzione `require()`, con la sola differenza che se il codice di un file è stato già incluso, esso non sarà incluso nuovamente.
`require_once()` dovrebbe essere usato nei casi dove lo stesso file potrebbe essere incluso e valutato più di una volta durante una particolare esecuzione di uno script, e volete essere sicuri che esso sia incluso esattamente una volta per evitare problemi con la ridefinizione di funzioni, riassegnazione di valori a variabili, etc.

`include_once()`

L'istruzione `include_once()` include e valuta il file specificato durante l'esecuzione dello script. È un comportamento simile all'istruzione `include()`, con la sola differenza che se il codice di un file è stato già incluso, esso non sarà incluso nuovamente. Come suggerisce il nome, esso sarà incluso solo una volta.

Funzioni

Funzioni definite dall'utente

Una funzione può essere definita usando la seguente sintassi:

```
function fun ($arg_1, $arg_2, ..., $arg_n)
{
    echo "Funzione di esempio.\n";
    return $retval;
}
```

All'interno di una funzione può apparire qualunque codice PHP valido, persino altre funzioni e definizioni di classe.

Argomenti delle funzioni

L'informazione può essere passata alle funzioni tramite la lista degli argomenti, che sono liste di variabili e/o costanti delimitati dalla virgola.
PHP supporta il passaggio di argomenti per valore (comportamento di default), il passaggio per riferimento, e i valori di default degli argomenti. Le liste di argomenti di lunghezza variabile sono supportate solo in PHP

4 e successivi: vedere Liste di argomenti a lunghezza variabile e i riferimenti alle funzioni `func_num_args()`, `func_get_arg()`, e `func_get_args()` per maggiori informazioni.

Costruire argomenti passati per riferimento

Di default, gli argomenti della funzione sono passati per valore (così se cambiate il valore dell'argomento all'interno della funzione, esso non cambierà fuori della funzione). Se volete permettere ad una funzione di modificare i suoi argomenti, dovete passarli per riferimento.

Se volete che una argomento sia passato sempre per riferimento ad una funzione, dovete anteporre una e commerciale (&) al nome dell'argomento nella definizione della funzione:

```
function aggiungi_qualcosa(&$string)
{
    $string .= 'e qualche altra cosa.';
}
$str = 'Questa è una stringa, ';
aggiungi_qualcosa($str);
echo $str;  // l'output sarà 'Questa è una stringa, e qualche altra cosa.'
```

Valori predefiniti degli argomenti

Una funzione può definire valori predefiniti in stile C++ per argomenti scalari come segue:

```
function fare_il_caffe ($tipo = "cappuccino")
{
    return "Sto facendo una tazza di $tipo.\n";
}
echo fare_il_caffe ();
echo fare_il_caffe ("espresso");
```

L'output dal frammento di sopra è:

```
Sto facendo una tazza di cappuccino.
Sto facendo una tazza di espresso.
```

Il valore predefinito deve essere un'espressione costante, non (per esempio) una variabile o un membro di classe.

Da notare che quando vengono usati argomenti predefiniti, qualunque argomento predefinito dovrebbe essere a destra degli argomenti non-predefiniti; diversamente, le cose non funzioneranno come ci si aspetti. Si consideri il seguente frammento di codice:

```
function fare_lo_yogurt ($tipo = "yogurt", $gusto)
{
    return "Fare una vaschetta di $tipo a $gusto.\n";
}
echo fare_lo_yogurt ("fragola");   // non funziona come si aspetta
```

L'output dell'esempio di sopra è:

```
Warning: Missing argument 2 in call to fare_lo_yogurt() in
/usr/local/etc/httpd/htdocs/php3test/functest.html on line 41
```

Ora, si confronti il codice di sopra con questo:

```
function fare_lo_yogurt ($gusto, $tipo = "yogurt")
{
    return "Fare una vaschetta di $tipo a $gusto.\n";
}
echo fare_lo_yogurt ("fragola");   // funziona come si aspetta
```

L'output di questo esempio è: `Fare una vaschetta di yogurt a fragola`.

Liste di argomenti a lunghezza variabile

PHP 4 ha il supporto per le liste di argomenti a lunghezza variabile nelle funzioni definite dall'utente. Ciò è abbastanza semplice, usando le funzioni `func_num_args()`, `func_get_arg()`, e `func_get_args()`.

Non è richiesta una speciale sintassi, e le liste di argomenti possono ancora essere provviste esplicitamente con le definizioni di funzioni e si comporteranno normalmente.

Valori restituiti

I valori vengono restituiti usando l'istruzione opzionale `return`. Può essere restituito qualsiasi tipo, incluse liste ed oggetti. Ciò provoca l'interruzione dell'esecuzione della funzione immediatamente e la restituzione del controllo alla linea successiva a quella da cui è stata chiamata.

```
function quadrato ($num)
{
    return $num * $num;
}
echo quadrato (4);   // L'output è '16'.
```

Non possono essere restituiti valori multipli da una funzione, ma risultati simili possono essere ottenuti restituendo un array.

```
function numeri_piccoli()
{
    return array (0, 1, 2);
}
list ($zero, $uno, $due) = numeri_piccoli();
```

Per restituire un riferimento da una funzione, è necessario usare l'operatore di passaggio per riferimento `&` nella dichiarazione di funzione e quando viene assegnato il valore restituito ad una variabile:

```
function &restituisce_riferimento()
{
    return $un_riferimento;
}
$nuovo_riferimento =& restituisce_riferimento();
```

Funzioni variabili

PHP supporta il concetto di funzioni variabili. Ciò significa che se un nome di variabile ha le parentesi accodate ad esso, PHP cercherà una funzione con lo stesso nome del valore della variabile, e cercherà di eseguirla. Tra le altre cose, ciò può essere usato per implementare delle callbacks, tabelle di funzioni e così via.

Le funzioni variabili non funzionano con costrutti di linguaggio come `echo()`, `unset()`, `isset()`, `empty()` e `include()`. Il costrutto `print()` è un'eccezione e funzionerà. Questa è una delle maggiori differenze tra le funzioni PHP e i costrutti di linguaggio.

```
<?php
function foo()
{
    echo "In foo()<br>\n";
}
function bar($arg = '')
{
    echo "In bar(); l'argomento era '$arg'.<br>\n";
}
$func = 'foo';
$func();
$func = 'bar';
$func('test');
?>
```

Classi e Oggetti

Una classe è una collezione di variabili e funzioni che utilizzano queste variabili. Una classe si definisce usando la seguente sintassi:

```
<?php
class Cart
{
    var $items;  // Articoli nel carrello

    // Aggiunge $num articoli di $artnr nel carrello
```

```
        function add_item ($artnr, $num)
        {
            $this->items[$artnr] += $num;
        }
        // Prende $num articoli di $artnr e li rimuove dal carrello

        function remove_item ($artnr, $num)
        {
            if ($this->items[$artnr] > $num) {
                $this->items[$artnr] -= $num;
                return true;
            } else {
                return false;
            }
        }
    }
    ?>
```

Il codice definisce una classe chiamata `Cart` composta da un array associativo che archivia gli articoli nel carrello e due funzioni per aggiungere e rimuovere gli articoli dal carrello stesso.

Cautela

Il nome `stdClass` è usato esclusivamente da Zend ed è riservato. Non è quindi possibile creare una classe chiamata `stdClass` in PHP.

Cautela

I nomi di funzione `__sleep` e `__wakeup` sono riservati e magici nelle classi PHP. Non è possibile creare funzioni con questi nomi nelle classi definite dall'utente, a meno che non sia desiderata la funzionalità magica connessa a questi nomi. Si veda sotto per avere più informazioni.

Cautela

PHP riserva tutti i nomi di funzione che iniziano con __ a funzioni magiche. Si suggerisce di non usare nomi di funzioni che utilizzano con i caratteri __in PHP a meno che non si desideri implementare una funzionalità magica.

In PHP 4, sono permesse inizializzazioni di variabili con valori costanti solamente grazie all'uso di `var`. Per inizializzare variabili con valori non-costanti, bisogna creare una funzione d'inizializzazione che è chiamata automaticamente all'istanziazione di un oggetto da una classe. Questo tipo di funzione si chiama costruttore .

```
<?php
class Cart
{
    var $todays_date;
    var $name;
    var $owner;
    var $items;
    function Cart()
    {
        $this->todays_date = date("Y-m-d");
        $this->name = $GLOBALS['firstname'];
        /* etc ... */
    }
}
?>
```

Le classi sono tipi del linguaggio, e sono modelli per variabili reali. Per creare una variabile oggetto si usa l'operatore new.

```
<?php
$cart = new Cart;
$cart->add_item("10", 1);
$another_cart = new Cart;
$another_cart->add_item("0815", 3);
```

```
?>
```

Il codice sopra, genera gli oggetti `$cart` e `$another_cart`, dalla classe `Cart`. La funzione `add_item()` dell'oggetto `$cart` è chiamata per aggiungere una ricorrenza dell'articolo numero 10 a `$cart`. A `$another_cart` sono aggiunte 3 ricorrenze dell'articolo numero 0815.

extends

Spesso si ha bisogno di avere classi con variabili e funzioni simili ad altre classi. É buona norma definire una classe in modo generico, sia per poterla riutilizzare spesso, sia per poterla adattare a scopi specifici. Per facilitare questa operazione, è possibile generare classi per estensione di altre classi. Una classe estesa o derivata ha tutte le variabili e le funzioni della classe di base più tutto ciò che viene aggiunto dall'estensione. Non è possibile che una sottoclasse, ridefinisca variabili e funzioni di una classe madre. Una classe estesa dipende sempre da una singola classe di base: l'eredità multipla non è supportata. Le classi si estendono usando la parola chiave 'extends'.

```
<?php
class Named_Cart extends Cart
{
    var $owner;

    function set_owner ($name)
    {
        $this->owner = $name;
    }
}
?>
```

Qui viene definita una classe `Named_Cart` che ha tutte le funzioni e variabili di `Cart` più la variabile `$owner` e la funzione `set_owner()`. Viene creato un carrello con nome con il metodo usato in precedenza, in più la classe estesa permette di settare o leggere il nome del carrello. Si possono usare variabili e funzioni sia di `Cart` che della sua estensione:

```
<?php
$ncart = new Named_Cart;    // Crea un carrello con nome
$ncart->set_owner("kris");  // Assegna il nome al carrello
print $ncart->owner;        // stampa il nome del proprietario
$ncart->add_item("10", 1);  // (funzionalità ereditata da Cart)
?>
```

La relazione mostrata è chiamata relazione "genitore-figlio". Si crea una classe di base, poi utilizzando extends si crea una nuova classe basata sulla classe genitore: la classe figlia. Successivamente si può usare la classe figlia come classe base per un'altra classe.

Nota: Una classe deve essere definita prima di essere utilizzata! Se si vuole la classe `Named_Cart` che estende la classe `Cart`, bisogna definire una classe `Cart` prima. Se si vuole creare un'altra classe chiamata `Yellow_named_cart` basata sulla classe `Named_Cart` bisogna definire la classe `Named_Cart` prima. Per farla breve: l'ordine di definizione delle classi è importante.

Costruttori

I costruttori sono funzioni che esistono in una classe e che sono chiamate automaticamente quando si crea una nuova istanza di una classe con new. In PHP 3, una funzione si trasforma in un costruttore quando ha lo stesso nome di una classe. In PHP 4, una funzione diventa un costruttore, quando ha lo stesso nome di una classe ed è definita all'interno della classe stessa - la differenza è sottile, ma cruciale.

```
<?php
// Funziona in PHP 3 e PHP 4.
class Auto_Cart extends Cart
{
    function Auto_Cart()
    {
        $this->add_item ("10", 1);
    }
}
?>
```

Questo codice definisce una classe `Auto_Cart`, che non è altro che `Cart` più un costruttore che inizializza il carrello con una occorrenza dell'articolo numero "10" ogni volta che un nuovo `Auto_Cart` è creato con `new`. I costruttori possono avere degli argomenti, e gli argomenti possono essere facoltativi, questo li rende molto versatili. Per poter usare una classe senza specificare parametri, tutti i parametri del costruttore devono essere resi facoltativi con valori di default.

```
<?php
// Funziona in PHP 3 and PHP 4.
class Constructor_Cart extends Cart
{
    function Constructor_Cart($item = "10", $num = 1)
    {
        $this->add_item ($item, $num);
    }
}

// Istanzia il vecchio e noioso carrello.

$default_cart = new Constructor_Cart;

// Un carrello nuovo ...

$different_cart = new Constructor_Cart("20", 17);
?>
```

`serialize()`

L'operazione di serializzazione di un oggetto restituisce una stringa che contiene una rappresentazione in caratteri di tutti i valori che possono essere memorizzati in PHP. `unserialize()` può usare questa stringa per ricreare dai valori variabili utilizzabili.

Usando `serialize()` per salvare un oggetto si salveranno tutte le variabili dell'oggetto. Le funzioni dell'oggetto non sono salvate, viene salvato solo il nome della classe.

Per potere usare `unserialize()` su un oggetto, la classe dell'oggetto deve essere definita. Cioè se avete un oggetto `$a` della classe `A` su una pagina di nome `page1.php` e usate `serialize()`, otterrete una stringa che si riferisce alla classe `A` e contiene tutti i valori delle variabili contenute in `$a`. Se desiderate poter deserializzare l'oggetto in un'altra pagina chiamata `page2.php`, dovete ricreare `$a` dalla classe `A`, la definizione della classe `A` perciò deve essere presente nella pagina `page2.php`. Questo può essere fatto per esempio memorizzando la definizione della classe A in un file che viene incluso sia in `page1.php` che in `page2.php`.

```
<?php
// classa.inc:
    class A
  {
      var $one = 1;
      function show_one()
      {
          echo $this->one;
      }
  }

// page1.php:
  include("classa.inc");
  $a = new A;
  $s = serialize($a);  // memorizzare $s in qualche posto della page2.
  $fp = fopen("store", "w");
  fputs($fp, $s);
  fclose($fp);
// page2.php:
// questo è necessario perché unserialize() funzioni correttamente.
  include("classa.inc");
```

```
  $s = implode("", @file("store"));
  $a = unserialize($s);
  // ora usiamo la function show_one() dell'oggetto $a.
  $a->show_one();
?>
```

Se state usando le sessioni ed usate `session_register()` per registrare oggetti, questi oggetti vengono serializzati automaticamente alla fine di ogni pagina PHP e sono deserializzate automaticamente su ogni pagina della sessione. Ciò significa che gli oggetti possono mostrarsi in ogni pagina e che sono parte integrante della sessione.
Si suggerisce vivamente di includere le definizioni delle classi degli oggetti registrati su tutte le pagine, anche se le classi non sono usate su tutte le pagine. Se un oggetto viene deserializzato senza la relativa definizione della classe, perderà l'associazione ad essa e si trasformerà in un oggetto della classe `stdClass` senza nessuna funzione disponibile, diventando inutile.
Così se nell'esempio qui sopra `$a` diventasse parte di una sessione e fosse registrato con `session_register("a")`, dovreste includere un file `classa.inc` su tutte le pagine in cui è valida la sessione, non soltanto nella `page1.php` e nella `page2.php`.

Le funzioni magiche __sleep e __wakeup

`serialize()` controlla se la vostra classe ha una funzione dal nome magico `__sleep`. In caso affermativo, quella funzione viene eseguita prima di qualsiasi serializzazione. La funzione può pulire l'oggetto e restituire un array con i nomi di tutte le variabili di quell' oggetto che dovrebbero essere serializzate.
Si intende usare `__sleep` quando chiudendo un collegamento ad un database l'oggetto può avere dati pendenti e l'oggetto ha bisogno di essere ripulito. Inoltre, la funzione è utile se avete oggetti molto grandi che non devono essere salvati completamente.
Per contro, `unserialize()` controlla per vedere se c'è nella classe una funzione dal nome magico `__wakeup`. Se è presente questa funzione può ricostruire qualunque risorsa che l'oggetto aveva.
L'intento di `__wakeup` è quello di ristabilire le connessioni ai database che possono esser state persi durante la serializzazione ed effettuare altre mansioni reinizializzazione.

Confronto di oggetti in PHP 4

In PHP 4, gli oggetti sono confrontati semplicemente, cioè: due istanze di oggetto sono uguali se hanno gli stessi attributi e valori, e sono istanze della stessa classe. Questa regola regola è applicata anche nel confronto di due oggetti utilizzando l'operatore di identità (===).

```
<?php
function bool2str($bool) {
    if ($bool === false) {
            return 'FALSE';
    } else {
            return 'TRUE';
    }
}
function compareObjects(&$o1, &$o2) {
    echo 'o1 == o2 : '.bool2str($o1 == $o2)."\n";
    echo 'o1 != o2 : '.bool2str($o1 != $o2)."\n";
    echo 'o1 === o2 : '.bool2str($o1 === $o2)."\n";
    echo 'o1 !== o2 : '.bool2str($o1 !== $o2)."\n";
}
class Flag {
    var $flag;
    function Flag($flag=true) {
            $this->flag = $flag;
    }
}
class SwitchableFlag extends Flag {
    function turnOn() {
        $this->flag = true;
    }
    function turnOff() {
```

```
            $this->flag = false;
        }
    }
    $o = new Flag();
    $p = new Flag(false);
    $q = new Flag();
    $r = new SwitchableFlag();
    echo "Confronto di istanze create con gli stessi parametri\n";
    compareObjects($o, $q);
    echo "\nConfronto di istanze create con parametri diversi\n";
    compareObjects($o, $p);
    echo "\nConfronto di un'istanza della classe genitore con una
    sottoclasse\n";
    compareObjects($o, $r);
    ?>
```

Si ha: Confronto di istanze create con gli stessi parametri

```
o1 == o2 : TRUE
o1 != o2 : FALSE
o1 === o2 : TRUE
o1 !== o2 : FALSE
```

Confronto di istanze create con parametri diversi

```
o1 == o2 : FALSE
o1 != o2 : TRUE
o1 === o2 : FALSE
o1 !== o2 : TRUE
```

Confronto di un'istanza della classe genitore con una sottoclasse

```
o1 == o2 : FALSE
o1 != o2 : TRUE
o1 === o2 : FALSE
o1 !== o2 : TRUE
```

Questo è l'output che si ottiene secondo le regole di confronto descritte sopra. Solo le istanze con gli stessi valori per gli attributi e derivanti dalla stessa classe sono considerate uguali ed identiche.

Funzioni MySQL

Con PHP 4, il supporto a MySQL è sempre abilitato; se non si specifica l'opzione di configurazione, vengono usate le librerie incluse.
La versione per Windows di PHP ha già compilato il supporto per questo modulo. Non occorre caricare alcun modulo addizionale per potere utilizzare queste funzioni.

Configurazione

Il comportamento di queste funzioni è influenzato dalle impostazioni di php.ini.

Nome	Predefinito	Modificabile in
mysql.allow_persistent	"On"	PHP_INI_SYSTEM
mysql.max_persistent	"-1"	PHP_INI_SYSTEM
mysql.max_links	"-1"	PHP_INI_SYSTEM
mysql.default_port	NULL	PHP_INI_ALL
mysql.default_socket	NULL	PHP_INI_ALL
mysql.default_host	NULL	PHP_INI_ALL
mysql.default_user	NULL	PHP_INI_ALL
mysql.default_password	NULL	PHP_INI_ALL
mysql.connect_timeout	"0" PHP	_INI_SYSTEM

`mysql.allow_persistent boolean`
Determina se consentire le connessioni persistenti a MySQL.
`mysql.max_persistent integer`
Il numero massimo di connessioni persistenti MySQL per processo.

mysql.max_links integer
Il numero massimo di connessioni MySQL per processo, incluse le connessioni persistenti.
mysql.default_port string
Il numero di porta TCP predefinito da usare per connettersi ad un server di database se nessuna altra porta viene specificata. Se nessun valore predefinito e specificato, la porta sarà ottenuta dalla variabile d'ambiente MYSQL_TCP_PORT, dalla voce mysql-tcp in /etc/services o dalla costante MYSQL_PORT in fase di compilazione, in questo ordine. Win32 userà solo la costante MYSQL_PORT.
mysql.default_socket string
Il nome del socket predefinito da usare per connettersi ad un server di database locale se nessun altro nome di socket viene specificato.
mysql.default_host string
L'host di default del server da usare per connettersi al server di database se nessun altro host viene specificato. Non si applica in safe mode.
mysql.default_user string
Il nome utente predefinito da usare per connettersi al server di database se nessun altro nome viene specificato. Non si applica in safe mode.
mysql.default_password string
La password predefinita da usare per connettrsi al server di database se nessuna altra password viene specificata. Non si appplica in safe mode.
mysql.connect_timeout integer
Timeout di connessione in secondi.

Tipi di risorse

Ci sono due tipi di risorsa usati nel modulo MySQL. Il primo è l'identificativo di connessione per una connessione ad un database, del secondo tipo sono le risorse che contengono i risultati di una query.

Costanti Predefinite

Costante	Descrizione
MYSQL_CLIENT_COMPRESS	Usa la compressione del protocollo
MYSQL_CLIENT_IGNORE_SPACE	Consente lo spazio dopo i nomi delle funzioni
MYSQL_CLIENT_INTERACTIVE	Lascia trascorrere interactive_timeout secondi (anziché wait_timeout) di inattività prima di chiudere la connessione

La funzione `mysql_fetch_array()` usa una costante per i diversi tipi di array risultato. Sono definite le seguenti costanti:

Costante	Descrizione
MYSQL_ASSOC	Le colonne sono restituite in un array avente il nome del campo come indice dell'array
MYSQL_BOTH	Le colonne sono restituite in un array avente sia un indice numerico sia un indice costituito dal nome del campo
MYSQL_NUM	Le colonne sono restituite in un array avente un indice numerico per i campi. Questo indice inizia da 0, il primo campo nel risultato

Questo esempio mostra come connettersi, eseguire una query, stampare le righe risultanti e disconnettersi dal database MySQL.

```
<?php
    /* Connessione e selezione del database */
    $connessione = mysql_connect("host_mysql", "utente_mysql",
"password_mysql")
        or die("Connessione non riuscita");
    print "Connesso con successo";
    mysql_select_db("mio_database") or die("Selezione del database non
riuscita");
    /* Esecuzione di una query SQL */
    $query = "SELECT * FROM mia_tabella";
    $risultato = mysql_query($query) or die("Query fallita");
    /* Stampa dei risultati in HTML */
```

```
        print "<table>\n";
        while ($linea = mysql_fetch_array($risultato, MYSQL_ASSOC)) {
            print "\t<tr>\n";
            foreach ($linea as $valore_colonna) {
                print "\t\t<td>$valore_colonna</td>\n";
            }
            print "\t</tr>\n";
        }
        print "</table>\n";
        /* Liberazione delle risorse del risultato */
        mysql_free_result($risultato);
        /* Chiusura della connessione */
        mysql_close($connessione);
    ?>
```

Gestione della sessione

Il supporto delle sessioni in PHP consiste nel mantenere certi dati attraverso accessi successivi. Per esempio, una volta autenticato un utente con Nome e Password validi per l'accesso ad alcune tabelle MySql, queste variabili possono essere conservate per tutta la sessione (altrimenti bisognerebbe chiederle ogni volta che accede al database)
Al visitatore che accede al vostro sito web viene assegnato un id unico, il cosiddetto id di sessione. Questo viene registrato in un cookie sul lato utente o è propagato tramite l'URL.
Il supporto delle sessioni vi permette di registrare numeri arbitrari di variabili che vengono preservate secondo richiesta. Quando un visitatore accede al vostro sito, PHP controllerà automaticamente (se session.auto_start è settato a 1) o su vostra richiesta (esplicitamente tramite session_start() o implicitamente tramite session_register()) se uno specifico id di sessione sia stato inviato con la richiesta. In questo caso , il precedente ambiente salvato viene ricreato.
Tutte le variabili registrate vengono serializzate dopo che la richiesta è finita. Le variabili registrate che non sono definite vengono marcate come indefinite. All'accesso successivo, queste non vengono definite dal modulo di sessione fino a quando l'utente non le definisce più tardi.
Da PHP 4.1.0, $_SESSION è disponibile come variabile globale proprio come $_POST, $_GET, $_REQUEST e così via.

```
<?php
if (isset($_SESSION['count'])) {
   $_SESSION['count']++;//incrementa la variabile count se esiste
}
else {
   $_SESSION['count'] = 0;//altrimenti la crea e la azzera
}
?>
```

Perché la variabile $_SESSION sia utilizzabile bisogna modificare il file php.ini nella directory WINDOWS (per chi usa questo sistema operativo) settando `session.auto_start =1` e creando, se non esiste, una directory temp dove vengono registrate le variabili di sessione.

Opzioni di configurazione nel file php.ini

Riportiamo alcune tra le opzioni di configurazione del file php.ini
`session.save_handler` definisce il nome dell'handler che è usato per archiviare e rilasciare i dati associati a una sessione. Di default è `files`.
`session.save_path` definisce l'argomento che è passato all'handler di sessione. Se scegliete handler files di default, questo è il percorso dove i files vengono creati. Di default è /tmp. Se la profondità del percorso session.save_path è più di 2, l'accumulo (gc) non sarà effettuato.
Attenzione
Se lasciate questo settato a directory leggibile da tutti /tmp (il default), altri utenti potrebbero essere in grado di dirottare le sessioni prendendo la lista dei files in quella directory.

session.name specifica il nome della sessione che è usata come nome del cookie. Dovrebbe contenere solo caratteri alfanumerici. Di default è PHPSESSID.
session.auto_start specifica se il modulo di sessione inizia una sessione automaticamente su richiesta iniziale. Di default è 0 (disattivata).
session.cookie_lifetime specifica il tempo di vita in secondi del cookie che viene mandato al browser. Il valore 0 significa "fino a che il browser viene chiuso". Di default è 0.
session.gc_maxlifetime specifica il numero di secondi dopo i quali i dati saranno considerati 'spazzatura' e cancellati.
session.use_cookies specifica se il modulo userà i cookies per archiviare l'id di sessione sul lato client. Di default è 1 (attivo).
session.cookie_path specifica il percorso da stabilire in session_cookie. Di default è /.

Funzioni per gestire la sessione

bool session_destroy (void)
session_destroy() distrugge tutti i dati associati alla sessione corrente. Non desetta nessuna delle variabili globali associate alla sessione o desetta il cookie di sessione.
Questa funzione ritorna TRUE in caso di successo e FALSE in caso di fallimento nel distruggere i dati di sessione.

```
<?php
// Inizializza la sessione.
// Se state usando session_name("qualcosa"), non dimenticatevelo adesso!
session_start();
// Desetta tutte le variabili di sessione.
session_unset();
// Infine , distrugge la sessione.
session_destroy();
?>
```

array session_get_cookie_params (void)
La funzione session_get_cookie_params() restituisce un array con le informazioni sul cookie di sessione corrente, l'array contiene i seguenti elementi:
"lifetime" - La durata del cookie.
"path" - Il percorso dove l'informazione è archiviata.
"domain" - Il dominio di validità del cookie.
"secure" - Il cookie dovrebbe essere spedito solo attraverso connessioni sicure. (Questo elemento è stato aggiunto in PHP 4.0.4.)
string session_id ([string id])
session_id() restituisce l'id di sessione per la sessione corrente. Se id è specificato, sostituirà l'id di sessione corrente.
string session_name ([string name])
session_name() ritorna il nome della sessione corrente. Se name è specificato, il nome della sessione corrente viene cambiato al suo valore.
void session_set_cookie_params (int lifetime [, string path [, string domain]])
Imposta i parametri del cookie definiti nel file php.ini. L'effetto di questa funzione dura solo per la durata dello script.
void session_unset (void)
La funzione session_unset() libera tutte le variabili di sessione correntemente registrate.

Funzioni MySql

int mysql_affected_rows ([resource identificativo_connessione])
restituisce il numero di righe coinvolte nell'ultima query INSERT, UPDATE o DELETE associata a identificativo_connessione. Se l'identificativo di connessione non è specificato, viene considerata l'ultima connessione aperta con mysql_connect().
Se l'ultima query era una query DELETE senza clausola WHERE, tutti i record saranno cancellati dalla tabella ma questa funzione restituirà zero.

Usando UPDATE, MySQL non aggiornerà le colonne nelle quali il nuovo valore è uguale al vecchio valore. Questo crea la possibilità che `mysql_affected_rows()` può non uguagliare realmente il numero di righe corrispondenti ma solo il numero di righe effettivamente coinvolte dalla query.

`mysql_affected_rows()` non funziona con l'istruzione SELECT ma solo con le istruzioni che modificano i record. Per ricavare il numero di righe restituite da SELECT, usare `mysql_num_rows()`.
Se l'ultima query fallisce, questa funzione restituisce -1.

Query di eliminazione

```
<?php
    /* connessione al database */
    mysql_pconnect("localhost", "utente_mysql", "password_mysql") or
        die("Connessione non riuscita: " . mysql_error());
    /* questo dovrebbe restituire il numero corretto di record eliminati */
    mysql_query("DELETE FROM mia_tabella WHERE id < 10");
    printf ("Records eliminati: %d\n", mysql_affected_rows());
    /* senza la clausola WHERE nell'istruzione DELETE,
    dovrebbe restituire 0 */
    mysql_query("DELETE FROM mia_tabella");
    printf ("Record eliminati: %d\n", mysql_affected_rows());
?>
```

L'esempio riportato sopra dovrebbe produrre il seguente output: Records eliminati: 10
Records eliminati: 0

Query di aggiornamento

```
<?php
    /* connessione al  database */
    mysql_pconnect("localhost", "utente_mysql", "password_mysql") or
        die("Connessione non riuscita: " . mysql_error());
    /* aggiornamento dei record */
    mysql_query("UPDATE mia_tabella SET used=1 WHERE id < 10");
    printf ("Record aggiornati: %d\n", mysql_affected_rows());
mysql_query("COMMIT");
?>
```

L'esempio riportato sopra dovrebbe produrre il seguente output: Record aggiornati: 10

`bool mysql_close ( [resource identificativo_connessione])`

Restituisce TRUE in caso di successo, FALSE in caso di fallimento.

`mysql_close()` chiude la connessione al server MySQL associata all'identificativo di connessione specificato. Se identificativo_connessione non è specificato, viene usata l'ultima connessione aperta.

L'uso di `mysql_close()` non è normalmente necessario, dal momento che le connessioni non persistenti sono chiuse automaticamente alla fine dell'esecuzione dello script.

Nota: `mysql_close()` non chiude le connessioni persistenti create da `mysql_pconnect()`.

Chiusura connessione MySQL

```
<?php
    $connessione = mysql_connect("localhost", "utente_mysql",
    "password_mysql")
        or die("Connessione non riuscita: " . mysql_error());
    print ("Connesso con successo");
    mysql_close($connessione);
?>
```

`resource mysql_connect ( [string server [, string nome_utente [, string password [, bool nuova_connessione [, int client_flags]]]]])`

Restituisce un identificativo di connessione MySQL in caso di successo oppure FALSE in caso di fallimento.

`mysql_connect()` stabilisce una connessione ad un server MySQL. I seguenti valore predefiniti sono assunti per i parametri opzionali mancanti: server = 'localhost:3306', nome_utente = nome dell'utente proprietario del processo server e password = password vuota.

Il parametro server può anche includere un numero di porta. Es. "hostname:porta" o un percorso ad un socket es. ":/percorso/al/socket" per il localhost.
Se si fa una seconda chiamata a `mysql_connect()` con gli stessi argomenti, nessuna nuova connessione sarà stabilita, ma sarà restituito l'identificativo della connessione già aperta. Il parametro `nuova_connessione` modifica questo comportamento e fa sì che `mysql_connect()` apra sempre una nuova connessione, anche se `mysql_connect()` era stata chiamata prima con gli stessi parametri. il parametro `client_flags` può essere combinato con le costanti MYSQL_CLIENT_COMPRESS, MYSQL_CLIENT_IGNORE_SPACE o MYSQL_CLIENT_INTERACTIVE.
Nota: Il parametro `nuova_connessione` è stato introdotto dal PHP 4.2.0
Il parametro `client_flags` è stato introdotto dal PHP 4.3.0
La connessione al server sarà chiusa prima della fine dell'esecuzione dello script, a meno che questa non sia precedentemente chiusa esplicitamente richiamando `mysql_close()`.

bool mysql_data_seek (resource identificativo_risultato, int numero_riga)

Restituisce TRUE in caso di successo, FALSE in caso di fallimento.
`mysql_data_seek()` muove il puntatore di riga interno del risultato MySQL associato all'identificativo specificato per puntare ad un determinato numero di riga. La successiva chiamata a `mysql_fetch_row()` dovrebbe restituire questa riga.
`numero_riga` inizia da 0. numero_riga dovrebbe essere un valore nell'intervallo che va da 0 a `mysql_num_rows - 1`.
Nota: La funzione `mysql_data_seek()` può essere usata solo insieme a `mysql_query()`, non con `mysql_unbuffered_query()`.

```
<?php
    $connessione = mysql_pconnect("localhost", "utente_mysql",
"password_mysql")
        or die("Connessione non riuscita: " . mysql_error());
    mysql_select_db("samp_db")
        or die("Selezione del database non riuscita: " . mysql_error());
    $query = "SELECT cognome, nome FROM amici";
    $risultato = mysql_query($query)
        or die("Query fallita: " . mysql_error());
    /* caricamento righe in ordine inverso */
    for ($i = mysql_num_rows($risultato) - 1; $i >= 0; $i--) {
        if (!mysql_data_seek($risultato, $i)) {
            echo "Non si può eseguire il seek alla riga $i: " .
mysql_error() . "\n";
            continue;
        }
        if(!($riga = mysql_fetch_object($risultato)))
            continue;
        echo "$riga->cognome $riga->nome<br />\n";
    }
    mysql_free_result($risultato);
?>
```

string mysql_db_name (resource risultato, int riga [, mixed campo])

`mysql_db_name()` accetta come primo paramentro il puntatore al risultato dalla chiamata a `mysql_list_dbs()`. Il parametro riga è un indice compreso nell'intervallo del risultato.
Se intercorre un errore, viene restituito FALSE. Usare `mysql_errno()` e `mysql_error()` per determinare la natura dell'errore.

```
<?php
    error_reporting(E_ALL);
    mysql_connect('dbhost', 'nome_utente', 'password');
    $db_list = mysql_list_dbs();
    $i = 0;
    $conteggio = mysql_num_rows($lista_db);
    while ($i < $conteggio) {
```

```
            echo mysql_db_name($lista_db, $i) . "\n";
            $i++;
        }
    ?>
```

int mysql_errno ([resource identificativo_connessione])

Restituisce il numero di errore dall'ultima funzione MySQL, oppure 0 (zero) se nessun errore è intercorso.
Gli errori ritornano dal database MySQL senza visualizzare i messaggi di avvertimento. Usando invece `mysql_errno()` si recupera il codice di errore. Notare che questa funzione restituisce solo il codice errore della più recente funzione MySQL eseguita, quindi se la si vuole usare, assicurarsi di controllare il valore prima di chiamare un'altra funzione MySQL.

```
<?php
    mysql_connect("localhost", "utente_mysql", "password_mysql");
    mysql_select_db("db_inesistente");
    echo mysql_errno() . ": " . mysql_error(). "\n";
    mysql_select_db("kossu");
    mysql_query("SELECT * FROM tabella_inesistente");
    echo mysql_errno() . ": " . mysql_error() . "\n";
?>
```

L'esempio riportato sopra dovrebbe produrre il seguente output:

```
1049: Unknown database 'db_inesistente'
1146: Table 'kossu.tabella_inesistente' doesn't exist
```

Nota: Se l'argomento opzionale è specificato la connessione indicata viene usata per recuperare il codice d'errore. Altrimenti viene usata l'ultima connessione aperta.

string errore_mysql ([resource identificativo_connessione])

Restituisce il testo dell'errore dall'ultima funzione MySQL, oppure " (la stringa vuota) se nessun errore intercorre.
Gli errori ritornano dal database MySQL senza visualizzare i messaggi di avvertimento. Si usa invece `mysql_error()` per recuperare il testo dell'errore. Notare che questa funzione restituisce solo il testo dell'errore della più recente funzione MySQL eseguita (ad esclusione di `mysql_error()` e `mysql_errno()`), quindi se la si vuole usare, assicurarsi di controllare il valore prima di richiamare un'altra funzione MySQL.

```
<?php
    mysql_connect("localhost", "utente_mysql", "password_mysql");
    mysql_select_db("db_inesistente");
    echo mysql_errno() . ": " . mysql_error(). "\n";
    mysql_select_db("kossu");
    mysql_query("SELECT * FROM tabella_inesistente");
    echo mysql_errno() . ": " . mysql_error() . "\n";
?>
```

L'esempio riportato sopra dovrebbe produrre il seguente output:

```
1049: Unknown database 'db_inesistente'
1146: Table 'kossu.tabella_inesistente' doesn't exist
```

Nota: Se l'argomento opzionale è specificato la connessione indicata viene usata per recuperare il codice d'errore. Altrimenti viene usata l'ultima connessione aperta.

array mysql_fetch_array (resource risultato [, int tipo_risultato])

Restituisce un array che corrisponde alla riga caricata o FALSE se non ci sono più righe.
`mysql_fetch_array()` è una versione estesa di `mysql_fetch_row()`. Oltre a memorizzare i dati del risultato in array con indice numerico, questa li memorizza anche con indici associativi usando i nomi dei campi come chiavi.

Se due o più colonne del risultato hanno gli stessi nomi di campo, l'ultima colonna avrà la precedenza. Per accedere alle altre colonne con lo stesso nome, si deve usare l'indice numerico della colonna o farne un alias. Per le colonne-alias, non si può accedere al contenuto con il nome della colonna originale (in questo esempio si usa 'campo').
Il secondo argomento opzionale tipo_risultato in `mysql_fetch_array()` è una costante e può assumere i seguenti valori: MYSQL_ASSOC, MYSQL_NUM e MYSQL_BOTH. Questa caratteristica è stata aggiunta nel PHP 3.0.7. MYSQL_BOTH è il valore predefinito per questo argomento.
Usando MYSQL_BOTH, si ottiene un array con entrambe gli indici (associativo e numerico). Usando MYSQL_ASSOC, si ottengono solo gli indici associativi (stesso funzionamento di `mysql_fetch_assoc()`), usando MYSQL_NUM, si ottengono solo gli indici numerici (stesso funzionamento di `mysql_fetch_row()`).

```
<?php
    mysql_connect("localhost", "utente_mysql", "password_mysql") or
        die("Connessione non riuscita: " . mysql_error());
    mysql_select_db("mio_db");
    $result = mysql_query("SELECT id, nome FROM mia_tabella");
    while ($riga = mysql_fetch_array($risultato, MYSQL_NUM)) {
        printf ("ID: %s  Nome: %s", $riga[0], $riga[1]);
    }
    mysql_free_result($risultato);
?>
```

`array mysql_fetch_assoc ( resource risultato)`

Restituisce un array associativo che corrisponde alla riga caricata o FALSE se non ci sono più righe.
`mysql_fetch_assoc()` è equivalente alla chiamata di `mysql_fetch_array()` con MYSQL_ASSOC come secondo parametro opzionale. Questa restituisce solo un array associativo. Questo è il modo in cui `mysql_fetch_array()` funzionava originalmente. Se è necessario l'indice numerico al posto di quello associativo, usare `mysql_fetch_array()`.
Se due o più colonne del risultato hanno gli stessi nomi di campo, l'ultima colonna avrà la precedenza. Per accedere alle altre colonne con lo stesso nome, si deve accedere al risultato con l'indice numerico usando `mysql_fetch_row()` oppure aggiungere degli alias. Vedere l'esempio nella descrizione di `mysql_fetch_array()` per quanto concerne gli alias.

```
<?php
    $conn = mysql_connect("localhost", "utente_mysql", "password_mysql");
    if (!$conn) {
        echo "Impossibile connettersi al DB: " . mysql_error();
        exit;
    }
    if (!mysql_select_db("mio_nome_db")) {
        echo "Impossibile selezionare mio_nome_db: " . mysql_error();
        exit;
    }
    $sql = "SELECT id as id_utente, nome_intero, stato_utente
            FROM   una_tabella
            WHERE  stato_utente = 1";
    $risultato = mysql_query($sql);
    if (!$risultato) {
        echo "Fallimento nell'esecuzione della query ($sql) dal DB: " .
mysql_error();
        exit;
    }
    if (mysql_num_rows($risultato) == 0) {
        echo "Nessuna riga trovata, niente da stampare quindi si esce";
        exit;
    }
    // Finché esiste una riga di dati, si pone questa riga in $riga come un
array associativo
    // Nota: Se ci si aspetta solo una riga, non è necessario usare un
ciclo
```

```
        // Nota: Se si mette extract($riga); all'interno del seguente ciclo,
        //       si creeranno $id_utente, $nome_intero e $stato_utente
        while ($riga = mysql_fetch_assoc($risultato)) {
            echo $riga["id_utente"];
            echo $riga["nome_intero"];
            echo $riga["stato_utente"];
        }
        mysql_free_result($risultato);
    ?>
```

object mysql_fetch_field (resource risultato [, int indice_campo])

Restituisce un oggetto contenente le informazioni di un campo.

`mysql_fetch_field()` può essere usata al fine di ottenere informazioni circa i campi di un determinato risultato di una query. Se l'indice del campo non è specificato, il successivo campo non ancora recuperato da `mysql_fetch_field()` viene considerato.

Le proprietà dell'oggetto sono:

`name` - nome della colonna

`table`	nome della tabella a cui appartiene la colonna
`max_length`	massima lunghezza della colonna
`not_null`	1 se la colonna non può essere NULL
`primary_key`	1 se la colonna è una chiave primaria
`unique_key`	1 se la colonna è una chiave unica
`multiple_key`	1 se la colonna è una chiave non-unica
`numeric`	1 se la colonna è numerica
`blob`	1 se la colonna è un BLOB
`type`	il tipo della colonna
`unsigned`	1 se la colonna è senza segno
`zerofill`	1 se la colonna è completata da zeri

```
    <?php
    mysql_connect('localhost:3306', $utente, $password)
        or die("Connessione non riuscita: " . mysql_error());
    mysql_select_db("database");
    $risultato = mysql_query("select * from tabella")
        or die("Query fallita: " . mysql_error());
    /* ottiene i metadata della colonna */
    $i = 0;
    while ($i < mysql_num_fields($risultato)) {
        echo "Informazioni della colonna $i:<br />\n";
        $meta = mysql_fetch_field($risultato);
        if (!$meta) {
            echo "Nessuna informazione disponibile<br />\n";
        }
        echo "<pre>
    blob:         $meta->blob
    max_length:   $meta->max_length
    multiple_key: $meta->multiple_key
    name:         $meta->name
    not_null:     $meta->not_null
    numeric:      $meta->numeric
    primary_key:  $meta->primary_key
    table:        $meta->table
    type:         $meta->type
    unique_key:   $meta->unique_key
    unsigned:     $meta->unsigned
    zerofill:     $meta->zerofill
    </pre>";
        $i++;
    }
```

```
mysql_free_result($risultato);
?>
```

array mysql_fetch_lengths (resource risultato)

Restituisce un array che corrisponde alle lunghezze di ogni campo nell'ultima riga caricata da `mysql_fetch_row()` oppure FALSE in caso di errore.

`mysql_fetch_lengths()` memorizza le lunghezze di ogni colonna dell'ultima riga restituita da `mysql_fetch_row()`, `mysql_fetch_array()` e `mysql_fetch_object()` in un array, iniziando con un indice pari a 0.

object mysql_fetch_object (resource risultato)

Restituisce un oggetto con proprietà che corrispondono alla riga caricata oppure FALSE se non ci sono più righe.

`mysql_fetch_object()` è simile a `mysql_fetch_array()`, con una differenza: viene restituito un oggetto invece che un array. Indirettamente, questo significa che si ha solo accesso ai dati attraverso i nomi dei campi e non attraverso il loro indice (i numeri sono nomi di proprietà illeciti).

```
<?php
/* questo è valido */
echo $riga->campo;
/* questo non è valido */
echo $riga->0;
?>

<?php
mysql_connect("hostname", "utente", "password");
mysql_select_db($db);
$risultato = mysql_query("select * from tabella");
while ($riga = mysql_fetch_object($risultato)) {
    echo $riga->id_utente;
    echo $riga->nome_intero;
}
mysql_free_result($risultato);
?>
```

array mysql_fetch_row (resource risultato)

Restituisce un array che corrisponde ad una riga caricata oppure FALSE se non ci sono più righe.

`mysql_fetch_row()` carica una riga di dati dal risultato associato all'identificativo specificato. La riga è restituita come un array. Ogni colonna del risultato è memorizzata in un indice dell'array, partendo dall'indice 0.

La susseguente chiamata a `mysql_fetch_row()` restituisce la successiva riga nell'intervallo del risultato oppure FALSE se non ci sono più righe.

string mysql_field_flags (resource risultato, int indice_campo)

`mysql_field_flags()` restituisce i flag del campo specificato. I flag sono restituiti come singole parole per flag separate da un singolo spazio, in modo che sia possibile suddividere il valore restituito usando `explode()`.

I seguenti flag sono restituiti, se la versione di MySQL è abbastanza aggiornata da supportarli: "not_null", "primary_key", "unique_key", "multiple_key", "blob", "unsigned", "zerofill", "binary", "enum", "auto_increment", "timestamp".

int mysql_field_len (resource risultato, int indice_campo)

`mysql_field_len()` restituisce la lunghezza del campo specificato.

string mysql_field_name (resource risultato, int indice_campo)

`mysql_field_name()` restituisce il nome del campo specificato dall'indice. risultato deve essere un identificativo di risultato valido e indice_campo è lo spiazzamento numerico del campo.

Nota: indice_campo inizia da 0.

```
<?php
/* La tabella utenti è costituita da tre campi:
 *   id_utente
 *   nome_utente
 *   password
 */
$connessione = mysql_connect('localhost', "utente_mysql",
"password_mysql");
mysql_select_db($nome_db, $connessione)
    or die("Errore nella selezione di $dbname: " . mysql_error());
$risultato = mysql_query("select * from utenti", $connessione);
echo mysql_field_name($risultato, 0) . "\n";
echo mysql_field_name($risultato, 2);
 ?>
```

L'esempio riportato sopra dovrebbe produrre il seguente output:

```
id_utente
password
```

int mysql_field_seek (resource risultato, int indice_campo)

Esegue il seek ad uno specifico indice di campo. Se la successiva chiamata a `mysql_fetch_field()` non include un indice di campo, quello specificato in `mysql_field_seek()` viene restituito.

string mysql_field_table (resource risultato, int indice_campo)

Ottiene il nome della tabella in cui si trova il campo specificato.

string mysql_field_type (resource risultato, int indice_campo)

`mysql_field_type()` è simile alla funzione `mysql_field_name()`. Gli argomenti sono identici, ma viene restituito il tipo del campo. Il tipo del campo sarà uno dei seguenti: "int", "real", "string", "blob" ed altri come dettagliati nella Documentazione di MySQL.

```
<?php
    mysql_connect("localhost", "utente_mysql", "password_mysql");
    mysql_select_db("mysql");
    $risultato = mysql_query("SELECT * FROM func");
    $campi = mysql_num_fields($risultato);
    $righe   = mysql_num_rows($risultato);
    $tabella = mysql_field_table($risultato, 0);
    echo "La tabella'".$table."' ha ".$fields." campi e ".$righe."
record\n";
    echo "La tabella ha i seguenti campi:\n";
    for ($i=0; $i < $campi; $i++) {
        $tipo = mysql_field_type($risultato, $i);
        $nome = mysql_field_name($risultato, $i);
        $lung = mysql_field_len($risultato, $i);
        $flag = mysql_field_flags($risultato, $i);
        echo $tipo." ".$nome." ".$lung." ".$flag."\n";
    }
    mysql_free_result($risultato);
    mysql_close();
?>
```

L'esempio riportato sopra dovrebbe produrre il seguente output:

```
La tabella 'func' ha 4 campi e 1 record
La tabella ha i seguenti campi:
string name 64 not_null primary_key binary
int ret 1 not_null
string dl 128 not_null
string type 9 not_null enum
```

bool mysql_free_result (resource risultato)

`mysql_free_result()` libera tutta la memoria associata all'identificativo del risultato.
`mysql_free_result()` deve essere richiamata solo se si è preoccupati sulla quantità di memoria usata dalle query che restituiscono dei grandi risultati. Tutta la memoria associata al risultato è automaticamente liberata al termine dell'esecuzione dello script.
Restituisce TRUE in caso di successo, FALSE in caso di fallimento.

string mysql_get_client_info (void)

`mysql_get_client_info()` restituisce una stringa che rappresenta la versione della libreria client.

```
<?php
    printf ("Informazioni sul client MySQL: %s\n",
mysql_get_client_info());
?>
```

L'esempio riportato sopra dovrebbe produrre il seguente output:
Informazioni sul client MySQL: 3.23.39

string mysql_get_host_info ([resource identificativo_connessione])

`mysql_get_host_info()`restituisce una stringa che descrive il tipo di connessione in uso per identificativo_connessione, includendo il nome dell'host del server. Se identificativo_connessione è omesso, sarà usata l'ultima connessione aperta.

```
<?php
    mysql_connect("localhost", "utente_mysql", "password_mysql") or
        die("Connessione non riuscita: " . mysql_error());
    printf ("Informazioni sull'host MySQL: %s\n", mysql_get_host_info());
?>
```

L'esempio riportato sopra dovrebbe produrre il seguente output:
Informazioni sull'host MySQL: Localhost via UNIX socket

string mysql_get_server_info ([resource identificativo_connessione])

`mysql_get_server_info()` restituisce la versione del server usato dalla connessione identificativo_connessione. Se identificativo_connessione è omesso, sarà usata l'ultima connessione aperta.

```
<?php
    mysql_connect("localhost", "utente_mysql", "password_mysql") or
        die("Connessione non riuscita: " . mysql_error());
    printf ("Versione server MySQL: %s\n", mysql_get_server_info());
?>
```

L'esempio riportato sopra dovrebbe produrre il seguente output:
Versione server MySQL: 4.0.1-alpha

string mysql_info ([resource identificativo_connessione])

`mysql_info()` restituisce informazioni dettagliate relative all'ultima query usando lo specifico identificativo_connessione. Se identificativo_connessione non è specificato, viene considerata l'ultima connessione aperta.
mysql_info() restituisce una stringa per tutte le istruzioni elencate di seguito. Per tutte le altre restituisce FALSE. Il formato della stringa dipende dall'istruzione data.
`INSERT INTO ... SELECT ...`
String format: Records: 23 Duplicates: 0 Warnings: 0
`INSERT INTO ... VALUES (...),(...),(...)...`
String format: Records: 37 Duplicates: 0 Warnings: 0
`LOAD DATA INFILE ...`
String format: Records: 42 Deleted: 0 Skipped: 0 Warnings: 0
`ALTER TABLE`
String format: Records: 60 Duplicates: 0 Warnings: 0
`UPDATE`
String format: Rows matched: 65 Changed: 65 Warnings: 0
I numeri sono indicati solo a titolo esemplificativo; i loro valori corrispondono alla query.

int mysql_insert_id ([resource identificativo_connessione])

`mysql_insert_id()` restituisce l'identificativo generato per una colonna AUTO_INCREMENT dal precedente query INSERT usando lo specifico identificativo_connessione. Se identificativo_connessione non è specificato, viene considerata l'ultima connessione aperta.

`mysql_insert_id()` restituisce 0 se la precedente query non ha generato un valore AUTO_INCREMENT. Se è necessario salvare il valore per usarlo in seguito, assicurarsi di richiamare `mysql_insert_id()` immediatamente dopo la query che ha generato il valore.

Nota: Il valore della funzione SQL LAST_INSERT_ID() di MySQL contiene sempre il più recente valore AUTO_INCREMENT generato e non è azzerato dalle query.

Attenzione

`mysql_insert_id()` converte il tipo restituito dalla funzione nativa dell'API C di MySQL `mysql_insert_id()` al tipo long (chiamata int nel PHP). Se la colonna AUTO_INCREMENT appartiene al tipo BIGINT, il valore restituito da `mysql_insert_id()` sarà inesatto. In questo caso si usi la funzione SQL di MySQL LAST_INSERT_ID() in una query SQL.

```
<?php
    mysql_connect("localhost", "utente_mysql", "password_mysql") or
        die("Connessione non riuscita: " . mysql_error());
    mysql_select_db("mio_db");
    mysql_query("INSERT INTO mia_tabella (prodotto) VALUES ('kossu')");
printf ("L'ultimo record inserito ha l'identificativo %d\n",
mysql_insert_id());
?>
```

resource mysql_list_dbs ([resource identificativo_connessione])

`mysql_list_dbs()` restituirà un risultato puntatore contenete i database resi disponibili dal demone mysql. Usare la funzione `mysql_tablename()` per esplorare questo risultato puntatore o qualsiasi funzione per i risultati delle tabelle, come `mysql_fetch_array()`.

```
<?php
$connessione = mysql_connect('localhost', 'utente_mysql',
'password_mysql');
$lista_db = mysql_list_dbs($connessione);
while ($riga = mysql_fetch_object($lista_db)) {
    echo $riga->Database . "\n";
}
?>
```

L'esempio riportato sopra dovrebbe produrre il seguente output:
database1
database2
database3
...

resource mysql_list_fields (string nome_database, string nome_tabella [, resource identificativo_connessione])

`mysql_list_fields()` recupera le informazioni relative ad una data tabella. Gli argomenti sono il nome del database ed il nome della tabella. Viene restituito un risultato puntatore che può essere usato con `mysql_field_flags()`, `mysql_field_len()`, `mysql_field_name()` e `mysql_field_type()`.

```
<?php
$connessione = mysql_connect('localhost', 'utente_mysql',
'password_mysql');
$campi = mysql_list_fields("database1", "tabella1", $connessione);
$colonne = mysql_num_fields($campi);
for ($i = 0; $i < $colonne; $i++) {
    echo mysql_field_name($campi, $i) . "\n";
}
```

L'esempio riportato sopra dovrebbe produrre il seguente output:

campo1
campo2
campo3
...

resource mysql_list_tables (string database [, resource identificativo_connessione])

`mysql_list_tables()` accetta un nome di database e restituisce un risultato puntatore in modo molto simile alla funzione `mysql_query()`. Usare la funzione `mysql_tablename()` per esplorare questo risultato puntatore o qualsiasi funzione per i risultati delle tabelle, come `mysql_fetch_array()`.
Il parametro `database` è il nome del database da cui recuperare la lista di tabelle. in caso di errore, `mysql_list_tables()` restituisce FALSE.

```
<?php
    $nome_db = 'nome_db_mysql';
    if (!mysql_connect('host_mysql', 'utente_mysql', 'password_mysql')) {
        print 'Connessione a mysql non riuscita';
        exit;
    }
    $risultato = mysql_list_tables($nome_db);
    if (!$risultato) {
print "Errore database, Impossibile elencare le tabelle\n";
        print 'Errore MySQL: ' . mysql_error();
        exit;
    }
    while ($riga = mysql_fetch_row($risultato)) {
        print "Tabella: $riga[0]\n";
    }
    mysql_free_result($risultato);
?>
```

int mysql_num_fields (resource risultato)

`mysql_num_fields()` restituisce il numero di campi in un risultato.
Vedere anche: `mysql_select_db()`, `mysql_query()`, `mysql_fetch_field()` e `mysql_num_rows()`.

int mysql_num_rows (resource risultato)

`mysql_num_rows()` restituisce il numero di righe in un risultato. Questo comando è valido solo per le istruzioni SELECT. Per recuperare il numero di righe coinvolte dalle query INSERT, UPDATE o DELETE, usare `mysql_affected_rows()`.

```
<?php
$connessione = mysql_connect("localhost", "utente_mysql",
"password_mysql");
mysql_select_db("database", $connessione);
$risultato = mysql_query("SELECT * FROM tabella1", $connessione);
$num_righe = mysql_num_rows($risultato);
echo "$num_righe Righe\n";
?>
```

resource mysql_pconnect ([string server [, string nome_utente [, string password [, int flag_client]]]])

Restituisce un identificativo di connessione MySQL valido in caso di successo oppure FALSE in caso di errore.
`mysql_pconnect()` stabilisce una connessione ad un server MySQL. I seguenti valori predefiniti sono usati per i parametri opzionali mancanti: server = 'localhost:3306', nome_utente = nome dell'utente prprietario del processo server e password = password vuota. Il parametro flag_client può essere una

combinazione delle costanti MYSQL_CLIENT_COMPRESS, MYSQL_CLIENT_IGNORE_SPACE o MYSQL_CLIENT_INTERACTIVE.
Il parametro server può includere una numero di porta. Es. "hostname:porta" o un percorso ad un socket es. ":/percorso/a/socket" per il localhost.
Nota: Il supporto per ":porta" è stato aggiunto nel PHP 3.0B4.
Il supporto per ":/percorso/a/socket" è stato aggiunto nel PHP 3.0.10.
`mysql_pconnect()` agisce in modo molto simile a `mysql_connect()` con due differenze principali.

- quando si connette, la funzione tenta innanzitutto di trovare una connessione (persistente) già aperta avente gli stessi host, username e password. Se viene trovata una connessione, viene restituito un identificativo a questa anziché aprirne una nuova.
- la connessione al server SQL non sarà chiusa quando l'esecuzione dello script termina. La connessione rimane invece aperta per usi futuri (`mysql_close()` non chiuderà le connessioni stabilite da `mysql_pconnect()`).

Il parametro opzionale flag_client è diventato disponibile nel PHP 4.3.0.
Questo tipo di link è perciò chiamato 'persistente'.
Attenzione!
L'uso di connessioni persistenti può richiedere un po' di messa a punto delle configurazioni di Apache e MySQL per assicurarsi di non eccedere il numero di connessioni consentite da MySQL.

`bool mysql_ping ( [resource identificativo_connessione])`

`mysql_ping()` controlla se una connessione al server funziona o meno. Se questa è caduta, viene tentata una riconnessione automatica. Questa funzione può essere usata dagli script che rimangono inattivi per un lungo periodo per controllare se il server ha chiuso la connessione o meno e riconnettersi se necessario. `mysql_ping()` restituisce TRUE se la connessione al server è funzionante, altrimenti FALSE.

`resource mysql_query ( string query [, resource identificativo_connessione [, int modo_risultato]])`

`mysql_query()` invia una query al database attualmente attivo sul server associato all'identificativo di connessione specificato. Se identificativo_connessione non è specificato, viene considerata l'ultima connessione aperta. Se nessuna connessione è aperta, la funzione prova a stabilire una connessione come se `mysql_connect()` fosse richiamata senza argomenti ed usa questa.
Il paramentro opzionale modo_risultato può essere MYSQL_USE_RESULT e MYSQL_STORE_RESULT. Il valore predefinito MYSQL_STORE_RESULT, così il risultato è bufferato. Vedere anche `mysql_unbuffered_query()` per la controparte di questo comportamento.
Nota: La stringa della query non dovrebbe terminare con un punto e virgola.
Solo per le istruzioni SELECT, SHOW, EXPLAIN o DESCRIBE `mysql_query()` restituisce un identificativo di risorsa o FALSE se la query non è stata eseguita correttamente. Per altri tipi di istruzioni SQL, `mysql_query()` restituisce TRUE in caso di successo e FALSE in caso di errore. Un valore restituito diverso da FALSE indica che la query era lecita ed è stata eseguita dal server. Questo non indica niente riguardo il numero di righe coinvolte o restituite. è assolutamente possibile che una query abbia successo ma che non coinvolga o restituisca nessuna riga.
`mysql_query()` fallisce e restituisce FALSE anche se non si hanno i permessi per accedere alle tabelle cui la query fa riferimento.
Assumendo che la query abbia successo, si può richiamare `mysql_num_rows()` per scoprire quante righe sono state restituite da un'istruzione SELECT o `mysql_affected_rows()` per scoprire quante righe sono state coinvolte da un'istruzione DELETE, INSERT, REPLACE o UPDATE.
Solo per le istruzioni SELECT, SHOW, DESCRIBE o EXPLAIN, `mysql_query()` restituisce un nuovo identificativo di risultato che si può passare a `mysql_fetch_array()` e ad altre funzioni che si occupano dei risultati delle tabelle. Quando si conclude il trattamento del risultato, si possono liberare le risorse associate ad esso richiamando `mysql_free_result()`. Comunque la memoria sarà liberata automaticamente Al termine dell'esecuzione dello script.

`mixed mysql_result ( resource risultato, int campo [, mixed campo])`

`mysql_result()` restituisce i contenuti di una cella da un risultato MySQL. L'argomento campo può essere l'indice o il nome del campo oppure il nome della tabella ed il nome del campo separati da un punto

(nome_tabella.nome_campo). Se il nome della colonna ha un alias ('select foo as bar from...'), usare l'alias al posto del nome della colonna.
Quando si lavora con un risultato di grandi dimensioni, si dovrebbero considerare l'uso delle funzioni che caricano l'intera riga. Poiché queste funzioni restituiscono i contenuti di celle multiple in una chiamata a funzione, sono MOLTO più veloci di `mysql_result()`. Notare anche che specificare un indice numerico per l'argomento campo è molto più veloce che specificare un argomento del tipo `nome_di_campo` o `nome_tabella.nome_campo`.
Le chiamate a `mysql_result()` non dovrebbero essere mescolate con chiamate ad altre funzioni che hanno a che fare con i risultati.
Alternative raccomandate per alte prestazioni: `mysql_fetch_row()`, `mysql_fetch_array()` e `mysql_fetch_object()`.

bool mysql_select_db (string nome_database [, resource identificativo_connessione])

Restituisce TRUE in caso di successo, FALSE in caso di fallimento.
`mysql_select_db()` imposta il database attualmente attivo sul server associato all'identificativo di connessione specificato. Se nessun identificativo di connessione è specificato, viene considerata l'ultima connessione aperta. Se nessuna connessione è aperta, la funzione proverà a stabilire una connessione come se `mysql_connect()` fosse richiamata senza argomenti ed userà questa.
Ogni chiamata successiva a `mysql_query()` sarà fatta sul database attivo.
Vedere anche: `mysql_connect()`, `mysql_pconnect()` e `mysql_query()`.

string mysql_stat ([resource identificativo_connessione])

`mysql_stat()` restituisce l'attuale stato del server.
Nota: `mysql_stat()` attualmente restituisce solo le seguenti informazioni: uptime, thread, query, tabelle aperte, tabelle svuotate e query al secondo. Per una lista completa delle altre variabili di stato si usi il comando SQL SHOW STATUS.

```
<?php
$connessione = mysql_connect('localhost', "utente_mysql",
"password_mysql");
$stato = explode('  ', mysql_stat($connessione));
print_r($stato);
?>
```

L'esempio riportato sopra dovrebbe produrre il seguente output: Array
(
[0] => Uptime: 5380
[1] => Threads: 2
[2] => Questions: 1321299
[3] => Slow queries: 0
[4] => Opens: 26
[5] => Flush tables: 1
[6] => Open tables: 17
[7] => Queries per second avg: 245.595
)

string mysql_tablename (resource risultato, int i)

`mysql_tablename()` prende il puntatore risultato dalla funzione `mysql_list_tables()` come un indice intero e restituisce il nome di una tabella. La funzione `mysql_num_rows()` può essere usata per determinare il numero di tabelle nel risultato puntatore. Usare la funzione `mysql_tablename()` per esplorare questo risultato puntatore o qualsiasi funzione per i risultati delle tabelle, come `mysql_fetch_array`

```
<?php
    mysql_connect("localhost", "utente_mysql", "password_mysql");
    $risultato = mysql_list_tables("mio_db");
    for ($i = 0; $i < mysql_num_rows($risultato); $i++)
```

```
            printf ("Tabella: %s\n", mysql_tablename($risultato, $i));
        mysql_free_result($risultato);
    ?>
```

int mysql_thread_id ([resource identificativo_connessione])

`mysql_thread_id()` restituisce l'attuale thread ID. Se La connessione è persa a ci si riconnette con `mysql_ping()`, il thread ID cambia. Questo significa che non si dovrebbe ottenere il thread ID e memorizzarlo per impieghi successivi. Si dovrebbe rilevare il thread ID quando è necessario.

```
<?php
$connessione = mysql_connect('localhost', 'utente_mysql',
'password_mysql');
$thread_id = mysql_thread_id($connessione);
if ($thread_id){
    printf ("L'attuale thread è %d\n", $thread_id);
}
?>
```

L'esempio riportato sopra dovrebbe produrre il seguente output: L'attuale thread è 73

resource mysql_unbuffered_query (string query [, resource identificativo_connessione [, int modo_risultato]])

`mysql_unbuffered_query()` invia query a MySQL senza caricare e bufferare le righe risultanti automaticamente come fa mysql_query(). Da una parte, questo risparmia un considerevole quantità di memoria con le query SQL che producono risultati di grandi dimensioni. Dall'altra parte, si può iniziare l'elaborazione dei risultati immediatamente dopo che la prima riga è stata recuperata: non si deve attendere finché la quei SQL sia completamente eseguita. Quando si usano diverse connessioni a database, si deve specificare il parametro opzionale identificativo_connessione.

Il parametro opzionale `modo_risultato` può essere MYSQL_USE_RESULT e MYSQL_STORE_RESULT. Il valore predefinito è MYSQL_USE_RESULT, quindi il risultato non è bufferato. Vedere anche `mysql_query()` per la controparte di questo comportamento.

Titolo:	Manuale essenziale PHP
Autore:	Luciano Viviani
Email:	luciano_viviani@libero.it
Classe:	TERZA INFORMATICA SERALE (3IS)
Anno scolastico:	2003/2004
Scuola:	Itis Euganeo Via Borgofuro, 6 Via Borgofuro 6 - 35042 Este (PD) - Italy Telefono 0429.21.16 - 0429.34.72 Fax 0429.41.86 http://www.itiseuganeo.it informazioni@itiseuganeo.it
Note legali:	Nessuna restrizione all'utilizzo