Guida PHP

per principianti

Guida PHP di base

di Gianluca Gillini

Cos'è PHP

A metà degli anni Novanta il Web era ancora formato in gran parte da **pagine statiche**, cioè da documenti HTML il cui contenuto non poteva cambiare fino a quando qualcuno non interveniva manualmente a modificarlo. Con l'evoluzione di Internet, però, si cominciò a sentire l'esigenza di rendere dinamici i contenuti, cioè di far sì che la stessa pagina fosse in grado di proporre contenuti diversi, personalizzati in base alle preferenze degli utenti, oppure estratti da una base di dati (database) in continua evoluzione.

PHP nasce nel 1994, ad opera di Rasmus Lerdorf, come una serie di macro la cui funzione era quella di facilitare ai programmatori l'amministrazione delle homepage personali: da qui trae origine il suo nome, che allora significava appunto Personal Home Page. In seguito, queste macro furono riscritte ed ampliate fino a comprendere un pacchetto chiamato Form Interpreter (PHP/FI).

Essendo un progetto di tipo open source (cioè "codice aperto", quindi disponibile e modificabile da tutti), ben presto si formò una ricca comunità di sviluppatori che portò alla creazione di **PHP 3**: la versione del linguaggio che diede il via alla crescita esponenziale della sua popolarità. Tale popolarità era dovuta anche alla forte integrazione di PHP con il Web server Apache (il più diffuso in rete), e con il database MySQL. Tale combinazione di prodotti, integralmente ispirata alla filosofia del free software, diventò ben presto vincente in un mondo in continua evoluzione come quello di Internet.

Alla fine del 1998 erano circa 250.000 i server Web che supportavano PHP: un anno dopo superavano il milione. I 2 milioni furono toccati in aprile del 2000, e alla fine dello stesso anno erano addirittura 4.800.000. Il 2000 è stato sicuramente l'anno di maggiore crescita del PHP, coincisa anche con il rilascio della **versione 4**, con un nuovo motore (Zend) molto più veloce del precedente ed una lunga serie di nuove funzioni, fra cui quelle importantissime per la gestione delle sessioni. La crescita di PHP, nonostante sia rimasta bloccata fra luglio e ottobre del 2001, è poi proseguita toccando quota 7.300.000 server alla fine del 2001, per superare i 10 milioni alla fine del 2002, quando è stata rilasciata la **versione 4.3.0**. La continua evoluzione dei linguaggi di programmazione concorrenti e l'incremento notevole dell'utilizzo del linguaggio anche in applicazioni enterprise ha portato la Zend a sviluppare una nuova versione del motore per supportare una struttura ad oggetti molto più rigida e potente.

Nasce così **PHP 5**, che si propone come innovazione nell'ambito dello sviluppo web open source soprattutto grazie agli strumenti di supporto professionali forniti con la distribuzione standard ed al grande sforzo di Zend che, grazie alla partnership con IBM, sta cercando di spingere sul mercato soluzioni di supporto enterprise a questo ottimo linguaggio. Lo sviluppo di PHP procede comunque con due progetti paralleli che supportano ed evolvono sia la versione 4 che la versione 5. Questa scelta è stata fatta poichè tuttora sono pochi i fornitori di hosting che hanno deciso di fare il porting dei propri server alla nuova versione del linguaggio.

Oggi PHP è conosciuto come PHP: Hypertext Preprocessor, ed è un linguaggio completo di scripting, sofisticato e flessibile, che può girare praticamente su qualsiasi server Web, su qualsiasi sistema operativo (Windows o Unix/Linux, ma anche Mac, AS/400, Novell, OS/2 e altri), e consente di interagire praticamente con qualsiasi tipo di database (SQLite, MySQL, PostgreSQL, SQL Server, Oracle, SyBase, Access e altri). Si può utilizzare per i più svariati tipi di progetti, dalla semplice home page dinamica fino al grande portale o al sito di e-commerce.

La programmazione web: lato client e lato server

Parlando di PHP e di altri linguaggi di scripting può capitare di sentir citare le espressioni "lato client" e "lato server": per chi non è esperto della materia, tali definizioni possono suonare un po' misteriose. Proviamo a chiarire questi concetti: vediamo come funziona, in maniera estremamente semplificata, la richiesta di una pagina Web. L'utente apre il suo browser e digita un indirizzo Internet, ad esempio `www.nostrosito.it/pagina1.html`: a questo punto il browser si collega al server `www.nostrosito.it` e gli chiede la pagina `pagina1.html`. Tale pagina contiene esclusivamente codice HTML: il server la prende e la spedisce al browser, così com'è (insieme ad eventuali file allegati, ad esempio immagini). Il nostro utente quindi avrà la possibilità di visualizzare questa pagina.

Supponiamo ora che l'utente richieda invece la pagina `pagina2.php`: questa, contrariamente a quella di prima, non contiene solo codice HTML, **ma anche PHP**. In questo caso il server, prima di spedire la pagina, esegue il codice PHP, che in genere produce altro codice HTML: ad esempio, PHP potrebbe controllare che ore sono e generare un messaggio di questo tipo: "Buon pomeriggio, sono le 17.10!" oppure: "Ehi, che ci fai alzato alle 4 del mattino?". Dopo l'esecuzione, la pagina non conterrà più codice PHP, ma solo HTML. A questo punto è pronta per essere spedita al browser. (Ovviamente, il file che non contiene più codice PHP non è quello "originale", ma la "copia" che viene spedita al browser. L'originale rimane disponibile per le prossime richieste.) Quindi l'utente vede solo il codice HTML, e non ha accesso al codice PHP che ha generato la pagina.

Per comprendere ancora meglio questo concetto, confrontiamo PHP con un altro linguaggio di scripting molto diffuso sul Web, cioè **JavaScript**, che di solito viene usato come linguaggio "lato client": JavaScript infatti viene eseguito non dal server, ma dal browser dell'utente (il client, appunto). JavaScript ci consente di eseguire operazioni che riguardano il sistema dell'utente, come ad esempio aprire una nuova finestra del browser, o controllare la compilazione di un modulo segnalando eventuali errori prima che i dati vengano spediti al server. Ci permette anche di avere un'interazione con l'utente: ad esempio, possiamo far sì che quando il mouse passa su una determinata immagine, tale immagine si modifichi.

Per svolgere tutti questi compiti, JavaScript **deve essere eseguito sul sistema dell'utente**: per questo il codice JavaScript viene spedito al browser insieme al codice HTML. Quindi l'utente ha la possibilità di visualizzarlo, contrariamente a ciò che accade con PHP. Abbiamo citato alcune utili funzioni svolte da JavaScript sul browser dell'utente: PHP, essendo eseguito sul server, non è in grado di svolgere direttamente queste funzioni. Ma attenzione: questo non significa che non sia in grado ugualmente di controllarle! Infatti PHP svolge principalmente la funzione di 'creare' il codice della pagina che viene spedita all'utente: di conseguenza, così come può creare codice HTML, allo stesso modo può creare codice JavaScript. Questo significa che PHP ci può permettere, ad esempio, di decidere se ad un utente dobbiamo spedire il codice JavaScript che apre una nuova finestra, oppure no. In pratica, quindi, lavorando sul lato server abbiamo il controllo anche del lato client. Rimane un ultimo dettaglio da svelare: come fa il server a sapere quando una pagina contiene codice PHP che deve essere eseguito prima dell'invio al browser? Semplice: si basa sull'estensione

delle pagine richieste.

Nell'esempio che abbiamo visto prima, pagina1 aveva l'estensione `.html`, mentre pagina2 aveva l'estensione `.php`: sulla base di questo, il server sa che nel secondo caso deve eseguire PHP, mentre nel primo può spedire il file così com'è. In realtà il server deve essere istruito per poter fare ciò: generalmente gli si dice di eseguire PHP per le pagine che hanno estensione `.php`. È possibile comunque assegnargli qualsiasi altra estensione (fino a qualche anno fa veniva utilizzata `phtml`, anche se ormai la pratica è caduta in disuso). Si possono utilizzare anche le estensioni standard `.htm` e `.html`, ma ciò significherebbe chiamare PHP per tutte le pagine richieste, anche se non contengono codice PHP: questo rallenterebbe inutilmente il lavoro del server e dunque è meglio evitarlo.

Caratteristiche e vantaggi di PHP

A fronte di quello detto precedentemente va precisato che PHP non è l'unico linguaggio lato server disponibile per chi si appresta a sviluppare pagine web. Sono disponibili varie alternative, sia proprietarie che open source, ed ognuna di queste ha i suoi pregi ed i suoi difetti. Dato che il paragone tra due linguaggi di programmazione di buon livello è spesso soggettivo, in questa sede preferisco descrivere in modo semplice le caratteristiche di PHP ed i vantaggi che queste possono portare allo sviluppatore, lasciando a voi l'eventuale compito di confrontarlo con altri strumenti.

La versione di PHP su cui si basa questa guida (la **5.1.2**) implementa soluzioni avanzate che permettono un controllo completo sulle operazioni che possono essere svolte dal nostro server web. L'accesso ai cookie ed alle sessioni è molto semplice ed intuitivo, avvenendo attraverso semplici variabili (vedi paragrafo relativo) che possono essere accedute da qualunque posizione all'interno del codice.

PHP ha una lunga storia legata esclusivamente al web, e per questo motivo esistono moltissime librerie testate e complete per svolgere i compiti più diversi: abbiamo strumenti per la gestione delle template, librerie che permettono la gestione completa di un mail server, sia in invio che in ricezione e molto altro ancora. A supporto di tutto questo bisogna dire che il modulo per eseguire script PHP è ormai installato di default sui server di hosting, e che la comunità di sviluppatori risolve molto velocemente i bug che si presentano agli utenti.

A supporto di PHP, sia Zend che la comunità php.net, hanno associato una serie di strumenti molto utili:

- Il repository PEAR (http://pear.php.net) che contiene decine di classi ben organizzate e documentate per svolgere la maggior parte delle operazioni ad alto e basso livello richieste durante lo sviluppo di applicazioni web. Tra queste ricordiamo il layer di astrazione per l'accesso ai database, le classi per il debugging ed il logging, quelle per la generazione di grafici avanzati e quelle per la gestione delle template. Ne abbiamo parlato diffusamente in un nostro articolo (http://php.html.it/articoli/leggi/877/pear-una-montagna-di-codice-php/).
- Il repository PECL (http://pecl.php.net) che contiene molte estensioni native che estendono le potenzialità del linguaggio con funzionalità di basso livello ad alte prestazioni. Abbiamo sistemi di cache ed ottimizzazione del codice intermedio generato durante l'esecuzione di script PHP, sistemi per il debugging avanzato ed il profiling del codice e molto altro.
- Il template engine Smarty (http://smarty.php.net), uno dei più robusti ed utilizzati template engine per PHP in circolazione. Ne abbiamo parlato diffusamente in un nostro articolo (http://php.html.it/articoli/leggi/909/template-engine-separare-programmazione-e-design/).
- A tutto questo va aggiunto che la funzione di Zend con IBM sta portando allo sviluppo di

strumenti di supporto professionali per gli sviluppatori, quali Zend Studio 5.0, Zend Safe Guard ed altri strumenti che coprono perfettamente tutto il processo di sviluppo e mantenimento del software.

Non è tutto rose e fiori purtroppo: per esempio il fatto che PHP venga distribuito in due versioni differenti (tuttora la 4 e la 5) **limita gli sviluppatori** nell'utilizzo delle caratteristiche della nuova versione, dato che la maggior parte dei servizi di hosting continuano ad aggiornare quella precedente. Oltretutto il fatto che PHP funzioni ad estensioni non è sempre un vantaggio, dato che spesso e volentieri i vari servizi di hosting hanno configurazioni differenti. Un'altra lacuna, che obbliga gli sviluppatori a sviluppare codice specifico per raggirarla, è la mancanza del supporto per caratteri Unicode che rende più complicato lo sviluppo di applicazioni multilingua per paesi che accettano caratteri speciali all'interno delle loro parole. Fortunatamente la versione 6 di PHP (che è tutt'ora in sviluppo) includerà nativamente questo supporto.

Insomma: PHP è un ottimo linguaggio, leader tra quelli open source per lo sviluppo web, molto semplice da imparare e subito produttivo. Oltretutto ha una serie di strumenti di appoggio molto completi e con la versione 5 un robusto supporto per la programmazione ad oggetti. Anche se con qualche difetto, penso sia la scelta più adeguata per un gran numero di situazioni.

PHP e l'HTML

PHP è un linguaggio la cui funzione fondamentale è quella di produrre codice HTML, che è quello dal quale sono formate le pagine Web. Ma, poichè PHP è un linguaggio di programmazione, abbiamo la possibilità di analizzare diverse situazioni (l'input degli utenti, i dati contenuti in un database) e di decidere, di conseguenza, di **produrre codice HTML condizionato** ai risultati dell'elaborazione. Questo è, in parole povere, il Web dinamico. Come abbiamo visto precedentemente, quando il server riceve una richiesta per una pagina PHP, la fa analizzare dall'interprete del linguaggio, il quale restituisce un file contenente solo il codice che deve essere inviato al browser (in linea di massima HTML, ma può esserci anche codice JavaScript, fogli di stile CSS o qualunque altro contenuto fruibile da un browser, come immagini e documenti Pdf).

Detto questo, come avviene la produzione di codice HTML? La prima cosa da sapere è come fa l'interprete PHP a discernere quale porzione di un file contiene codice da elaborare e quale codice da restituire solamente all'utente. Questa fase di riconoscimento è molto importante, dato che permette a PHP di essere incluso all'interno di normale codice HTML in modo da renderne dinamica la creazione. Il codice PHP deve essere compreso fra appositi tag di apertura e di chiusura, che sono i seguenti:

```
<?php //tag di apertura
?>    //tag di chiusura
```

Tutto ciò che è contenuto fra questi tag deve corrispondere alle regole sintattiche del PHP, ed è codice che sarà eseguito dall'interprete e non sarà inviato direttamente al browser al browser. Per generare l'output da inviare al browser attraverso codice PHP viene normalmente utilizzato il costrutto `echo`. Vediamo un semplice esempio, composto da codice HTML e codice PHP (il codice PHP è evidenziato in rosso):

```
<html>
<head>
  <title>
    <?php
      echo "Pagina di prova PHP";
    ?>
```

```
  </title>
</head>
<body>
  <?php
    echo "Buona giornata!";
  ?>
</body>
</html>
```

Questo banalissimo codice produrrà un file HTML il cui contenuto sarà semplicemente:

```
<html>
<head>
  <title>
    Pagina di prova PHP
  </title>
</head>
<body>
  Buona giornata!
</body>
</html>
```

Quindi l'utente vedrà sul suo browser la riga "Buona giornata!". È opportuno ricordare che il dato da inviare al browser che segue il comando `echo` può essere racchiuso tra parentesi e che al comando possono essere date in input più stringhe (questo è il nome che viene dato ad una ripetizione di qualunque carattere compreso tra due apici singoli (`' '`) o doppi (`" "`)), separate da virgole, così:

```
echo "Buongiorno a tutti!", "<br />\n", "È una bellissima giornata";
```

Se si decide di utilizzare il separatore virgola, non possono essere utilizzate le parentesi. Nel prosieguo del corso useremo spesso il verbo 'stampare' riferito alle azioni prodotte dal comando `echo` o da istruzioni con funzionalità analoghe (quali `print`, `sprintf` e altro): ricordiamoci però che si tratta di una convenzione, perchè in questo caso la 'stampa' non avviene su carta, ma sull'input che verrà inviato al browser!

Facciamo caso ad un dettaglio: nelle istruzioni in cui stampavamo "Buongiorno a tutti", abbiamo inserito, dopo il `<br />`, il simbolo `\n`. Questo simbolo ha una funzione abbastanza importante nella programmazione e nello scripting che serve più che altro per dare leggibilità al codice HTML che stiamo producendo. Infatti PHP, quando trova questa combinazione di caratteri fra virgolette, li trasforma in un carattere di **ritorno a capo**: questo ci permette di controllare l'impaginazione del nostro codice HTML. Bisogna però stare molto attenti a non confondere il codice HTML con il layout della pagina che l'utente visualizzerà sul browser: infatti, sul browser è solo il tag `<br />` che forza il testo ad andare a capo.

Quando questo tag non c'è, il browser allinea tutto il testo proseguendo sulla stessa linea (almeno fino a quando gli altri elementi della pagina e le dimensioni della finestra non gli "consigliano" di fare diversamente), anche se il codice HTML ha un ritorno a capo.

Vediamo di chiarire questo concetto con un paio di esempi:

```
<?php
echo "prima riga\n";
echo "seconda riga<br />";
echo "terza riga";
?>
```

Questo codice php produrrà il seguente codice HTML:

```
prima riga
seconda riga<br />terza riga
```

mentre l'utente, sul browser, leggerà:

```
prima riga seconda riga
terza riga
```

Questo perchè il codice PHP, mettendo il codice 'newline' dopo il testo 'prima riga', fa sì che il codice HTML venga formato con un ritorno a capo dopo tale testo. Il file ricevuto dal browser quindi andrà a capo proprio lì. Il browser, però, non trovando un tag che gli indichi di andare a capo, affiancherà la frase 'prima riga' alla frase 'seconda riga', limitandosi a mettere uno spazio fra le due. Successivamente accade l'esatto contrario: PHP produce un codice HTML nel quale il testo 'seconda riga' è seguito dal tag `<br />`, ma non dal codice 'newline'. Per questo, nel file HTML, '`seconda riga<br />`' e '`terza riga`' vengono attaccati. Il browser, però, quando trova il tag `<br />` porta il testo a capo.

Avrete forse notato che in fondo ad ogni istruzione PHP abbiamo messo un punto e virgola; infatti la sintassi del PHP prevede che il punto e virgola debba obbligatoriamente chiudere ogni istruzione. Ricordiamoci quindi di metterlo sempre, con qualche eccezione che vedremo più avanti. Da quanto abbiamo detto finora emerge una realtà molto importante: chi vuole avvicinarsi al PHP deve già avere una conoscenza approfondita di HTML e di tutto quanto può far parte di una pagina web (si consiglia per questo di leggere le nostre Guide all'HTML (http://xhtml.html.it/guide/leggi/51/guida-html/), al JavaScript (http://javascript.html.it/guide/leggi/26/guida-javascript-per-esempi/), e ai CSS (http://css.html.it/guide/leggi/2/guida-css-di-base/)).

Questo perchè lo scopo principale di PHP è proprio la produzione di questi codici (anche se va ricordato che può essere utilizzato per scopi differenti, come linguaggio di shell o per la creazione di applicazioni desktop grazie all'estensione PHP-GTK (http://php.html.it/articoli/leggi/884/introduzione-a-phpgtk/)).

La struttura sintattica del linguaggio

Nel paragrafo precedente abbiamo visto che PHP necessita di una coppia di tag per l'apertura e la chiusura del codice contenuto in un file richiesto da un Web Server. Si tratta dei tag

```
<?php
.....
?>
```

Abbiamo però la possibilità di usare anche alcune sintassi alternative, che sono comunque sconsigliate per permettere una corretta distribuzione e portabilità dei propri progetti. In primo luogo, in script scritti in versioni precedenti di PHP, potremmo trovare il classico tag `<script>` con specificato esplicitamente il linguaggio PHP:

```
<script language="php">
.......
</script>
```

Un'altra possibilità è quella di usare i `tag brevi`, che devono essere abilitati manualmente

modificando le impostazioni del file di configurazione `php.ini`:

```
<?
....
?>
```

Ricordiamoci tuttavia che in PHP 5 la **configurazione standard** disabilita i tag brevi. Un'ultima possibilità è quella di usare i tag in "stile ASP":

```
<%
....
%>
```

Anche questi però devono essere abilitati in `php.ini` e sono praticamente inutilizzati dalla maggior parte dei programmatori. Tutta questa varietà di sintassi è causata dalla volontà degli sviluppatori di mantenere **la retrocompatibilità** con le vecchie versioni che, a causa di scelte differenti di design, permettevano l'utilizzo di tag alternativi. È bene ricordare che rimane buona pratica utilizzare i tag completi evitando se possibile i tag brevi ed escludendo le altre possibilità.

Per chi non ci avesse fatto caso, puntualizzo un concetto molto importante: **i tag delimitano il codice PHP**, ed il codice contenuto al loro interno non sarà inviato al browser, ma compilato e eseguito. Da questo potremmo dedurre che tutto ciò che sta fuori da questi tag non verrà toccato da PHP, che si limiterà a passarlo al browser così com'è, eventualmente ripetendolo in base a situazioni particolari che vedremo in seguito. In linea di massima è bene ricordare che scrivere:

```
<?php
echo "<strong>";
?>
prova</strong>
```

e:

```
<strong>prova</strong>
```

(o qualsiasi altra combinazione) restituisce lo stesso risultato.

I commenti

Un altro argomento molto importante legato alla sintassi di PHP sono i **commenti**. Chi ha esperienza, anche minima, di programmazione, sa bene che molte volte i commenti si rivelano di importanza decisiva quando si tratta di mettere le mani su un programma realizzato da qualcun altro, e anche quando il programma è stato scritto da noi stessi, soprattutto se è passato qualche tempo dalla realizzazione. I commenti svolgono un ruolo fondamentale in questa fase di "rivisitazione" del codice, in quanto possono facilitare di molto la comprensione di passaggi apparentemente oscuri.

È bene quindi non risparmiare mai un commento quando possibile (senza comunque esagerare altrimenti, al posto di chiarire il codice lo renderete ancora più illeggibile), anche perchè il loro utilizzo **non appesantisce l'esecuzione dello script** (l'interprete PHP salta tutte le parti che riconosce come commenti), nè il trasferimento della pagina al browser (infatti i commenti, essendo contenuti all'interno del codice PHP, fanno parte di ciò che non viene inviato al browser).

Abbiamo tre diverse possibilità per posizionare i commenti all'interno del nostro codice: la prima è

l'uso dei commenti in stile C++, caratterizzati da due barre:

```
<?php
// Commento in stile C++
?>
```

La seconda sono i commenti in stile Perl e Python, contraddistinti dall'uso del cancelletto (anche se ormai obsoleti e poco utilizzati):

```
<?php
# Commento in stile Perl
# e python
?>
```

Entrambi questi tipi di commenti sono limitati ad una sola riga: l'interprete PHP, quando trova le barre o il cancelletto, salta tutto ciò che si trova da quel punto fino al termine della riga. Questo ci permette di porre il commento anche sulla stessa riga del codice commentato, così:

```
<?php
echo 'Buongiorno a tutti <br />';
//stampo un messaggio di saluto
print 'Esclusi quelli antipatici';
# faccio una precisazione
// Questa riga contiene solo commento
?>
```

L'ultimo tipo di commento che abbiamo a disposizione permette di specificare commenti multilinea senza dover ripetere i caratteri ogni nuova riga grazie ad una coppia di caratteri utilizzati per l'apertura e la chiusura. Tutto il codice che segue /* viene considerato commento da PHP finchè non incontra la serie di caratteri */. Un semplice esempio:

```
<?php

/*
Questo è un commento
multiriga specificando
utilizzando la stessa sintassi
usata in Java e C
*/
echo /* commento */ "Ciao a tutti" /* i commenti vengono saltati*/;

?>
```

Riguardo i commenti multilinea, è importante ricordare che non possono essere innestati e che comunque rappresentano un separatore per l'interprete PHP. Per questo motivo le seguenti sintassi sono errate:

```
<?php
/*
Commento /*
multilinea
*/
Qui verrà generato un errore ...
*/
ec/* questa sintassi è errata */ho "prova";
?>
```

La scelta di quale tipo di commento utilizzare è solitamente soggettiva, anche se spesso vengono utilizzati i commenti multiriga per documentare il codice e quelli a riga singola per aggiungergli dei semplici appunti sul funzionamento logico.

Le variabili

Le variabili sono componenti fondamentali di qualsiasi linguaggio di programmazione, in quanto ci consentono di **trattare i dati del nostro programma** senza sapere a priori quale sarà il loro valore. Possiamo immaginare una variabile come una specie di contenitore all'interno del quale viene conservato il valore che ci interessa, e che può cambiare di volta in volta.

In PHP possiamo scegliere il nome delle variabili usando lettere, numeri ed il trattino di sottolineatura, o underscore (_). Il primo carattere del nome deve essere però una lettera o un underscore (non un numero).

Dobbiamo inoltre ricordare che il nome delle variabili è sensibile all'**uso delle maiuscole e delle minuscole**: di conseguenza, se scriviamo due volte un nome di variabile usando le maiuscole in maniera differente, per PHP si tratterà di due variabili distinte!

In PHP il nome delle variabili è preceduto dal **simbolo del dollaro** ($). PHP ha una caratteristica che lo rende molto più flessibile rispetto ad altri linguaggi di programmazione: non richiede, infatti, che le variabili vengano dichiarate prima del loro uso. Possiamo quindi permetterci di riferirci ad una variabile direttamente con la sua valorizzazione:

```
<?php
  $a = 5;
?>
```

Con questa riga di codice definiamo la variabile a, assegnandole il valore 5. In fondo all'istruzione abbiamo il punto e virgola, che, come già accennato in precedenza, deve chiudere tutte le istruzioni PHP. L'utilità di una variabile diventa fondamentale nel momento in cui è possibile utilizzarla all'interno di espressioni matematiche o logiche. Vediamo un semplice esempio:

```
<?php
  $a = 9;
  $b = 4;
  $c = $a * $b;

  echo "Il risultato dell'operazione (9 * 4) è :";
  echo $c;
?>
```

In questo brano di codice abbiamo valorizzato tre variabili: a, alla quale stavolta abbiamo dato il valore 9; b, a cui abbiamo assegnato il valore 4 e c, che dovrà assumere il valore del prodotto di a e b. Infine abbiamo stampato il risultato ottenuto. Evidentemente, dopo l'esecuzione del codice c varrà 36.

Negli esempi abbiamo visto l'**inizializzazione delle variabili**, termine che sta ad indicare la prima volta in cui assegniamo un valore ad una variabile. In realtà, possiamo riferirci ad una variabile anche senza che sia stata inizializzata, anche se questa risulta un'operazione sconsigliata e potrebbe generare errori durante l'esecuzione in base ad alcune direttive di configurazione di PHP che vedremo più avanti. Ad esempio, supponendo che nel nostro script non sia stata valorizzata nessuna variabile `z`, potremmo avere un'istruzione di questo genere:

```
<?php
  echo $z;
?>
```

Questo codice non produrrà alcun output, in quanto la variabile `z` non esiste. Gli errori generati quando si cerca di utilizzare in lettura una variabile non inizializzata sono di tipo `E_NOTICE`: sono gli errori di livello più basso, cioè meno gravi, che normalmente non vengono mostrati da PHP, ma che possiamo abilitare attraverso il file di configurazione `php.ini`. Un errore di questo genere in effetti non compromette il buon funzionamento dello script, che infatti viene eseguito regolarmente; però potrebbe essere ugualmente indice di un qualche errore commesso da chi ha scritto il codice. Facciamo un esempio per chiarire meglio quanto esposto:

```
<?php
  $a = 74;
  $b = 29;
  $risultato = $a + $b;
  echo $risulato;
?>
```

Questo codice vorrebbe assegnare i valori `74` e `29` a due variabili, poi sommarli e infine stampare il risultato. Però contiene un errore: nell'istruzione di stampa abbiamo indicato la variabile `risulato` invece che `risultato`. Chi ha scritto il codice si aspetterebbe di vedere comparire sul browser il risultato `103`, invece non troverà proprio nulla, perchè la variabile `risulato` non è definita, e quindi non ha nessun valore.

Si può perdere anche molto tempo alla ricerca del problema (in realtà questo sembra un problema banalissimo, ed in effetti lo è; però ricordiamoci che un caso del genere potrebbe presentarsi in un contesto molto più complesso, ed inoltre molto spesso un errore semplice come questo si dimostra molto più difficile da scovare per chi lo ha commesso). Qui però la segnalazione di errore di PHP può venirci in aiuto: infatti, se siamo abituati ad utilizzare le variabili nella maniera più corretta, cioè dopo averle inizializzate, un errore come questo ci indica chiaramente che abbiamo sbagliato a scrivere il nome. Per questo il nostro consiglio è quello di **tenere abilitata la visualizzazione degli errori** anche di tipo `E_NOTICE`, e di utilizzare le variabili solo se inizializzate o dopo aver controllato la loro esistenza.

In questo modo impareremo da subito a programmare in maniera più corretta e, anche se impiegheremo qualche minuto in più per cominciare, risparmieremo tante ore (e tanto fegato) per il futuro...

Concludiamo questa lezione sulle variabili con un accenno alle variabili dinamiche, pratica tuttora sconsigliata ma che potrebbe capitarvi di incontrare in contesti relativi a script di vecchia fattura; in qualche situazione può presentarsi la necessità di utilizzare delle variabili senza sapere a priori il loro nome. In questi casi, il nome di queste variabili sarà contenuto in ulteriori variabili.

Facciamo un esempio: col codice seguente stamperemo a video il contenuto delle variabili `pippo`, `pluto` e `paperino`:

```
<?php
 $pippo = 'gawrsh!';
 $pluto = 'bau!';
 $paperino = 'quack!';

 $nome = 'pippo';
 echo $$nome.'<br>';
 $nome = 'pluto';
 echo $$nome.'<br>';
 $nome = 'paperino';
 echo $$nome.'<br>';
?>
```

Il risultato sul browser sarà `gawrsh!`, `bau!` e `quack!`, ciascuno sulla propria riga (infatti ogni istruzione print crea il tag HTML
 che indica al browser di andare a capo; vedremo più avanti che il punto serve a concatenare i valori che vengono stampati). Il **doppio segno del dollaro** ci permette infatti di usare la variabile `nome` come contenitore del nome della variabile di cui vogliamo stampare il valore. In pratica, è come se avessimo detto a PHP: «stampa il valore della variabile che si chiama come il valore della variabile `nome`». Questo era un esempio banale, e l'uso delle variabili dinamiche era in realtà perfettamente inutile, in quanto sapevamo benissimo come si chiamavano le variabili che ci interessavano. Però in situazioni reali può capitare di trovarsi in un ambito nel quale non sappiamo come si chiamano le variabili, e dobbiamo usare altre variabili per ricavarne il nome, oltrechè il valore.

I tipi di dato

Una variabile può contenere diversi tipi di valori, ognuno dei quali ha un comportamento ed un'utilità differente. Analizzeremo brevemente i tipi di dato che PHP permette di utilizzare all'interno del proprio codice premettendo che PHP, a differenza di altri linguaggi, associa il tipo di dato al valore e non alla variabile (ad esempio possiamo assegnare alla stessa variabile una stringa e poi un numero senza incorrere in alcun errore) ed effettua conversioni automatiche dei valori nel momento in cui siano richiesti tipi di dato differenti (ad esempio in un'espressione).

Valore booleano

I tipi di dato boolean servono per indicare i valori vero o falso all'interno di espressioni logiche. Il tipo booleano è associato alle variabili che contengono il risultato di un'espressione booleana oppure i valori true e false. Vediamo un rapido esempio:

```
<?php
$vero = true;
$falso = false;
?>
```

Intero

Un numero intero, positivo o negativo, il cui valore massimo (assoluto) può variare in base al sistema operativo su cui gira PHP, ma che generalmente si può considerare, per ricordarlo facilmente, di circa 2 miliardi (2 elevato alla 31esima potenza).

```
<?php
$int1 = 129;
$int2 = -715;
$int3 = 5 * 8; //$int3 vale 40
?>
```

Virgola mobile

Un numero decimale (a volte citato come "double" o "real"). Attenzione: per indicare i decimali non si usa la virgola, ma il punto. Anche in questo caso la dimensione massima dipende dalla piattaforma. Normalmente comunque si considera un massimo di circa 1.8e308 con una precisione di 14 cifre decimali. Si possono utilizzare le seguenti sintassi:

```
<?php
$vm1 = 4.153; // 4,153
$vm2 = 3.2e5; // 3,2 * 10^5, cioè 320.000
$vm3 = 4E-8; // 4 * 10^-8, cioè 4/100.000.000 = 0,00000004
?>
```

Stringa

Una stringa è un qualsiasi insieme di caratteri, senza limitazione normalmente contenuto all'interno di una coppia di apici doppi o apici singoli. Le stringhe delimitate da apici sono la forma più semplice, consigliata quando all'interno della stringa non vi sono variabili di cui vogliamo ricavare il valore:

```
<?php

$frase = 'Anna disse: "Ciao a tutti!" ma nessuno rispose';
echo $frase;

?>
```

Questo codice stamperà la frase: 'Anna disse: "Ciao a tutti!" ma nessuno rispose'. Gli apici doppi ci consentono di usare le stringhe in una maniera più sofisticata, in quanto, se all'interno della stringa delimitata da virgolette PHP riconosce un nome di variabile, lo sostituisce con il valore della variabile stessa.

```
<?php

$nome = 'Anna';
echo "$nome è simpatica... a pochi"; // stampa: Anna è simpatica... a pochi
echo '$nome è simpatica... a pochi'; // stampa: $nome è simpatica... a pochi
echo "{$nome} è simpatica a pochi";  // è una sintassi alternativa, con lo
stesso effetto della prima

?>
```

Ci sono un paio di regole molto importanti da ricordare quando si usano le stringhe delimitate da apici o virgolette: siccome può capitare che una stringa debba contenere a sua volta un apice o un

paio di virgolette, abbiamo bisogno di un sistema per far capire a PHP che quel carattere fa parte della stringa e non è il suo delimitatore. In questo caso si usa il cosiddetto 'carattere di escape', cioè la barra rovesciata (backslash: \). Vediamo alcuni esempi:

```
<?php

echo 'Torniamo un\'altra volta'; // stampa: Torniamo un'altra volta
echo "Torniamo un'altra volta"; // stampa: Torniamo un'altra volta
echo "Torniamo un\'altra volta"; // stampa: Torniamo un\'altra volta
echo 'Torniamo un'altra volta'; // causa un errore, perchè l'apostrofo viene
scambiato per l'apice di chiusura
echo 'Anna disse "Ciao" e se ne andò'; // stampa: Anna disse "Ciao" e se ne andò
echo "Anna disse \"Ciao\" e se ne andò"; // stampa: Anna disse "Ciao" e se ne
andò
echo 'Anna disse \"Ciao\" e se ne andò'; // stampa: Anna disse \"Ciao\" e se ne
andò
echo "Anna disse "Ciao" e se ne andò"; // errore

?>
```

Da questi esempi si può capire che il **backslash** deve essere utilizzato come carattere di escape quando vogliamo includere nella stringa lo stesso tipo di carattere che la delimita; se mettiamo un backslash davanti ad un apice doppio in una stringa delimitata da apici singoli (o viceversa), anche il backslash entrerà a far parte della stringa stessa, come si vede nel terzo e nel settimo esempio. Il backslash viene usato anche come 'escape di sè stesso', nei casi in cui vogliamo esplicitamente includerlo nella stringa:

```
<?php
echo "Questo: \"\\\" è un backslash";  // stampa: Questo: "\" è un backslash
echo 'Questo: \'\\\' è un backslash'); // stampa: Questo: '\' è un backslash
echo "Questo: '\' è un backslash");    // stampa: Questo: '\' è un backslash
echo "Questo: '\\' è un backslash");   // stampa: Questo: '\' è un backslash
?>
```

Analizziamo il primo esempio: il primo backslash fa l'escape del primo paio di virgolette; il secondo backslash fa l'escape del terzo, che quindi viene incluso nella stringa; il quarto fa l'escape del secondo paio di virgolette. Il secondo esempio equivale al primo, con l'uso degli apici al posto delle virgolette. Negli ultimi due casi non è necessario fare l'escape del backslash, in quanto il backslash che vogliamo stampare non può essere scambiato per un carattere di escape (infatti vicino ad esso ci sono degli apici, che in una stringa delimitata da virgolette non hanno bisogno di escape). Di conseguenza, fare o non fare l'escape del backslash in questa situazione è la stessa cosa, e difatti i due esempi forniscono lo stesso risultato.

Passiamo ad esaminare l'ultimo modo di rappresentare le stringhe: la sintassi heredoc, poco utilizzata se non in situazioni nelle quali è necessario specificare stringhe molto lunghe. Questa ci consente di delimitare una stringa con i caratteri **<<<** seguiti da un identificatore (in genere si usa `EOD`, ma è solo una convenzione: è possibile utilizzare qualsiasi stringa composta di caratteri alfanumerici e underscore, di cui il primo carattere deve essere non numerico: la stessa regola dei nomi di variabile). Tutto ciò che segue questo delimitatore viene considerato parte della stringa, fino a quando non viene ripetuto l'identificatore seguito da un punto e virgola. Attenzione: l'identificatore di chiusura deve occupare una riga a sè stante, deve iniziare a colonna 1 e non deve contenere nessun altro carattere (nemmeno spazi vuoti) dopo il punto e virgola.

```
<?php

$nome = "Paolo";
```

```
$stringa = <<<EOD
Il mio nome è $nome
EOD;
echo $stringa;

?>
```

Questo codice stamperà 'Il mio nome è Paolo'. Infatti la sintassi `heredoc` risolve i nomi di variabile così come le virgolette. Rispetto a queste ultime, con questa sintassi abbiamo il vantaggio di poter includere delle virgolette nella stringa senza farne l'escape:

```
<?php

$frase = "ciao a tutti";
$stringa = <<<EOT
Il mio saluto è "$frase"
EOT;
echo $stringa;

?>
```

In questo caso stamperemo 'Il mio saluto è "ciao a tutti"'.

Array

Possiamo considerare un array come una **variabile complessa**, che contiene una serie di valori, ciascuno dei quali caratterizzato da una chiave, o indice che lo identifica univocamente. Facciamo un primo esempio, definendo un array composto di cinque valori:

```
$colori = array('bianco', 'nero', 'giallo', 'verde', 'rosso');
```

A questo punto ciascuno dei nostri cinque colori è caratterizzato da un indice numerico, che PHP assegna automaticamente a partire da 0. Per recuperare un determinato valore dalla variabile che contiene l'array, è sufficiente specificare il suo indice all'interno di parentesi quadre dietro al nome della variabile:

```
echo $colori[1]; // stampa 'nero'
echo $colori[4]; // stampa 'rosso'
```

Gli array verranno trattati in modo più approfondito nella prossima lezione.

Oggetto

Le classi e gli oggetti sono due degli argomenti sui quali gli sviluppatori di PHP hanno voluto puntare maggiormente nella nuova versione. L'argomento verrà trattato in modo approfondito nella Guida Teorica a PHP.

Espressioni e operatori aritmetici di PHP

Gli operatori sono un altro degli elementi di base di qualsiasi linguaggio di programmazione, in quanto ci consentono non solo di effettuare le tradizionali operazioni aritmetiche, ma più in generale di **manipolare il contenuto delle nostre variabili**. Il più classico e conosciuto degli operatori è quello di assegnazione:

Operatore di assegnazione

```
$nome = 'Giorgio';
```

Il simbolo '=' serve infatti ad assegnare alla variabile `$nome` il valore `'Giorgio'`. In generale, possiamo dire che con l'operatore di assegnazione prendiamo ciò che sta alla destra del segno di ugualianza ('=') ed assegnamo lo stesso valore a ciò che sta a sinistra. Potremmo ad esempio assegnare ad una variabile il valore di un'altra variabile:

```
$a = 5;
$b = $a;
```

Con la prima istruzione assegnamo ad `$a` il valore `5`, con la seconda assegnamo a `$b` lo stesso valore di `$a`.

Altri operatori molto facili da comprendere sono quelli che permettono di effettuare operazioni aritmetiche sui dati: addizione, sottrazione, divisione, moltiplicazione, modulo.

Operazioni aritmetiche

```
$a = 3 + 7; // addizione
$b = 5 - 2; // sottrazione
$c = 9 * 6; // moltiplicazione
$d = 8 / 2; // divisione
$e = 7 % 4; // modulo
            // (il modulo è il resto della divisione intera, in questo caso 3)
```

Uno degli operatori più utilizzati è quello che serve per **concatenare le stringhe: il punto**.

Concatenare le stringhe

```
$nome = 'pippo';

$stringa1 = 'ciao ' . $nome; // $stringa1 vale 'ciao pippo'
```

Con l'operatore di assegnazione si può anche usare una variabile per effettuare un calcolo il cui risultato deve essere assegnato alla variabile stessa. Ad esempio, supponiamo di avere una variabile di cui vogliamo aumentare il valore:

```
$a = $a + 10; // il valore di $a aumenta di 10
```

Con questa istruzione, viene eseguito il calcolo che sta alla destra del segno '=' ed il risultato viene memorizzato nella variabile indicata a sinistra. Risulta chiaro che il valore della variabile `$a` prima dell'istruzione viene utilizzato per il calcolo, ma dopo che l'istruzione è stata eseguita viene utilizzata per memorizzare il risultato, quindi il suo valore cambia.

Un risultato di questo tipo si può ottenere anche con gli operatori di assegnazione combinati, che ci permettono di rendere il codice più compatto:

Operazioni di calcolo

```
$x += 4;  // incrementa $x di 4 (equivale a $x = $x + 4)
$x -= 3;  // decrementa $x di 3 (equivale a $x = $x - 3)
$x .= $a; // il valore della stringa $a viene concatenato a $x (equivale a $x =
$x . $a)
$x /= 5;  // equivale a $x = $x / 5
$x *= 4;  // equivale a $x = $x * 4
$x %= 2;  // equivale a $x = $x % 2
```

In questo modo diciamo a PHP che vogliamo assegnare alla variabile specificata a sinistra il risultato dell'operazione che si trova prima del simbolo uguale applicandola alla variabile stessa ed al valore specificato a destra. Più facile a farsi che a dirsi.

Nel caso fosse necessario incrementare e decrementare una variabile di una sola unità, ci vengono incontro gli operatori di incremento e decremento:

Incremento e decremento

```
$a++; ++$a; // incrementa di 1
$a--; --$a; // decrementa di 1
```

La differenza tra anteporre e posporre l'operatore di incremento o decremento è fondamentale nel momento in cui si utilizzando questi operatori all'interno di espressioni. Per ora vi basti sapere che anteporre l'operatore alla variabile dice al compilatore di incrementare la variabile e successivamente utilizzare il suo valore all'interno dell'espressione, mentre posporre l'operatore informa il compilatore che dovrà utilizzare nell'espressione il valore attuale e successivamente applicarvi l'incremento o il decremento.

Gli operatori logici e le espressioni booleane in PHP

Gli operatori di confronto

Gli operatori di confronto sono fondamentali perchè ci permettono, effettuando dei confronti fra valori, di prendere delle decisioni, cioè di far svolgere al nostro script determinate operazioni invece di altre. Quando utilizziamo gli operatori di confronto, confrontiamo i due valori posti a sinistra e a destra dell'operatore stesso.

Dopo aver valutato un'espressione questo tipo, PHP arriva a valutare se essa è vera o falsa. Quindi il risultato sarà di tipo booleano (`true` o `false`).

Operazioni di confronto

Operatore	Descrizione
==	uguale
!=	diverso
===	identico (cioè uguale e dello stesso tipo: ad esempio per due variabili di tipo intero)
>	maggiore
>=	maggiore o uguale
<	minore
<=	minore o uguale

Vediamo alcuni esempi:

```
$a = 7; $b = 7.0; $c = 4; //assegnamo valori a tre variabili
$a == $b;  // vero
$a == $c;  // falso
$a === $b; // falso, perchè $a è intero mentre $b è float
$a > $c;   // vero
$c >= $a;  // falso, $c è minore di $a
$a < $b;   // falso, hanno lo stesso valore
$c <= $b;  // vero
```

Una piccola osservazione sul terzo confronto: siccome abbiamo assegnato il valore di `$b` usando la notazione col punto decimale, per PHP `$b` è una variabile del tipo in virgola mobile, anche se in realtà il suo valore è intero. Per questo il confronto di identità restituisce falso.

Fino a qui abbiamo visto comunque casi molto semplici perchè tutte le variabili avevano valori numerici. Gli stessi confronti però si possono fare anche con altri tipi di variabili, ed in particolare con le **stringhe**. In questo caso il confronto viene fatto basandosi sull'ordine alfabetico dei caratteri: vale a dire che vengono considerati 'minori' i caratteri che 'vengono prima' nell'ordine alfabetico. Quindi 'a' è minore di 'b', 'b' è minore di 'c', eccetera. Inoltre tutte le lettere minuscole sono 'maggiori' delle lettere maiuscole, e tutte, maiuscole e minuscole, sono 'maggiori' delle cifre da 0 a 9:

```
$a = 'Mario'; $b = 'Giorgio'; $c = 'Giovanni'; $d = 'antonio'; $e = '4 gatti';

$a < $b; // falso, la 'G' precede la 'M'
$b < $c; // vero, la 'r' ('Gior') precede la 'v' ('Giov')
$d > $a; // vero, la 'a' minuscola è 'maggiore' di qualsiasi lettera maiuscola
$c > $e; // vero, ogni lettera è 'maggiore' di qualsiasi cifra
```

Dato che PHP è un linguaggio con una tipizzazione debole permette di **confrontare tra loro variabili contenenti tipi di dato differenti** cercando di trasformare le variabili in valori confrontabili. Se per esempio effettuassimo un confronto (`==`) tra una variabile contenente l'intero uno (`1`) ed una contenente la stringa uno (`'1'`) otterremmo un valore di verità dato che PHP trasformerebbe entrambi i valori in numeri in modo che siano confrontabili. Per assegnare questo valore numerico, PHP controlla se all'inizio della stringa ci sono dei numeri: se ne trova, considererà tutti i numeri che trova inizialmente come il valore numerico di quella stringa. Se non ne trova, il valore della stringa sarà `0`:

```
$a = 7; $b = 5; $c='molte persone'; $d='7 persone'; $e='5';

$a == $d;  // vero, $d vale 7
$a === $d; // falso, valgono entrambi 7 ma $a è un intero mentre $d è una
stringa
$b > $c;   // vero, $b vale 5 mentre $c vale 0
$e > $c;   // falso: questo è un confronto fra due stringhe, quindi valgono le
regole viste prima
```

Prestiamo attenzione all'ultimo esempio: il valore di `$e` era stato assegnato usando gli apici, e questo fa sì che PHP lo consideri una stringa anche se il contenuto è un numero.

Il confronto fra un numero e una stringa può avvenire in maniera voluta, ma è più probabile che avvenga per caso, quando cioè una variabile che pensavamo contenesse un numero contiene in realtà una stringa. È evidente che in questo caso potremo facilmente ottenere un risultato diverso da quello che ci aspettavamo, o, viceversa, potremmo ottenere casualmente il risultato atteso: in quest'ultima situazione è possibile che risultati inaspettati arrivino più avanti nello script, se utilizzeremo di nuovo la stessa variabile.

In tutte queste situazioni, tener presente il modo in cui PHP tratta questi confronti può essere di aiuto per spiegarci comportamenti apparentemente bizzarri del nostro script. È comunque buona norma assicurarsi che due espressioni restituiscano risultati dello stesso tipo quando si effettuano confronti, oppure utilizzare gli operatori `===` e `!==` che tengono conto anche del tipo di dato utilizzato.

Gli operatori logici

Con gli operatori logici possiamo combinare più valori booleani, oppure negarne uno (nel caso di NOT). Questi valori sono:

Operatore	Descrizione
`Or` o `||`	valuta se almeno uno dei due operatori è vero; si può indicare con '`Or` oppure col "doppio pipe" (`||`)
`And` o `&&`	valuta se entrambi gli operatori sono veri; si indica con `And` o con la doppia "e commerciale" (&&)
`Xor`	viene chiamato anche 'or esclusivo', e valuta se uno solo dei due operatori è vero: l'altro deve essere falso; si indica con `Xor`
`!`	è l'operatore 'not' e vale come negazione. Si usa con un solo operatore: è vero quando l'operatore è falso, e viceversa; si indica con il punto esclamativo (`!`)

Anche in questa occasione vediamo qualche esempio:

```
10 > 8 And 7 < 6; // falso, perchè la prima condizione è vera ma la seconda è
falsa
10 > 8 Or 7 < 6;  // vero
9 > 5 And 5 == 5; // vero, entrambe le condizioni sono vere
9 > 5 Xor 5 == 5; // falso, solo una delle due deve essere vera perchè si
verifichi lo 'Xor'
4 < 3 || 7 > 9;   // falso, nessuna delle due condizioni è vera
6 == 6 && 1 > 4;  // falso, solo la prima condizione è vera
```

Per quanto riguarda gli operatori 'and' e 'or', le due diverse notazioni differiscono per il livello di precedenza in caso di espressioni complesse. Infatti, siccome è possibile combinare molti operatori in espressioni anche assai complicate, è necessario sapere con quale ordine PHP valuta i diversi operatori. Queste regole ricalcano le regole algebriche in base alle quali moltiplicazioni e divisioni hanno la precedenza su addizioni e sottrazioni, ma sono più complesse perchè devono considerare anche gli altri operatori. Vediamo quindi qual è l'ordine di priorità dei diversi operatori, iniziando da quelli che hanno la priorità maggiore:

1. Operatori di incremento e decremento (`++ --`)
2. Moltiplicazione, divisione, modulo (`* / %`)
3. Addizione e sottrazione (`+ -`)
4. Operatori di confronto per minore e maggiore (`< <= => >`)
5. Operatori di confronto per uguaglianza e disuguaglianza (`== === !=`)
6. Operatore logico 'and', nella notazione col simbolo (&&)
7. Operatore logico 'or', nella notazione col simbolo (`||`)
8. Operatori di assegnazione, compresi quelli 'sintetici' (`= += -= /= *= %= .=`)
9. Operatore logico 'and', nella notazione letterale (`And`)
10. Operatore logico 'xor' (`Xor`)
11. Operatore logico 'or', nella notazione letterale (`Or`)

Abbiamo già visto prima, in occasione degli esempi sugli operatori logici, l'applicazione di questi principi di precedenza: infatti in tutte quelle espressioni venivano valutati prima gli operatori di confronto e, solo dopo, quelli logici. Un'altra classica rappresentazione di esempio è quella dell'espressione algebrica:

```
5 + 4 * 2;   // questa espressione vale 13 e non 18,
             // perché la moltiplicazione viene eseguita prima

(5 + 4) * 2; // questa invece vale 18, perché le parentesi modificano
```

```
            // l'ordine di esecuzione
```

Come abbiamo visto, così come avviene in algebra, usando le parentesi possiamo determinare a nostro piacere quali operatori devono essere valutati per primi. Per questo motivo, sebbene sia possibile imparare a memoria l'ordine di precedenza che abbiamo visto poco fa, il nostro consiglio è quello di non tenerne conto, e di **utilizzare sempre le parentesi** quando abbiamo bisogno di costruire un'espressione un po' complessa: in questo modo ridurremo il rischio di errori, e soprattutto renderemo il nostro codice molto più leggibile. Infatti leggere un'espressione regolata dalle parentesi è molto più immediato che non doversi ricordare quali degli operatori hanno la precedenza sugli altri.

Le espressioni

Dopo questa breve introduzione sugli operatori, possiamo definire un'espressione come una qualsiasi combinazione di funzioni (v. lezioni successive), valori e operatori che si risolvono in un valore. Nel caso visto prima, l'espressione 7+3 ha come valore 10.

In generale, in PHP, qualsiasi cosa utilizzabile come un valore può essere considerata un'espressione. Vediamo alcuni rapidi esempi:

```
15 * 3;               // espressione il cui valore è 45

'Giacomo' . ' Verdi'; // espressione il cui valore è 'Giacomo Verdi'

$a + $b;              // espressione il cui valore è dato dalla somma
                      // dei valori delle variabili $a e $b
```

Come possiamo vedere, quindi, la presenza di operatori fa sì che il valore dell'espressione risulti diverso da quello dei singoli valori che fanno parte dell'espressione stessa. Vediamo un caso particolare, quello dell'operatore di assegnazione:

```
$a = 6; // il valore di questa espressione è 6
```

Quando usiamo una espressione per assegnare un valore ad una variabile, il valore che tale espressione assume è uguale a quello che si trova a destra dell'operatore di assegnazione (che è anche quello che viene assegnato all'operatore di sinistra). Questo significa che noi possiamo scrivere

```
echo 'Paolo';           // stampa 'Paolo'
echo ($nome = 'Paolo'); // stampa sempre 'Paolo'
```

Le due espressioni hanno infatti lo stesso valore, cioè 'Paolo'. Quindi con le due istruzioni viste sopra otteniamo sul browser lo stesso risultato. La differenza, ovviamente, è che con la seconda, oltre a stampare il valore a video, abbiamo anche assegnato lo stesso valore alla variabile $nome.

Vediamo qualche altro esempio:

```
7 > 4;      //valore dell'espressione: true (vero)

$a = 7 > 4; //valore dell'espressione: lo stesso di prima;
            //la variabile $a assume quindi il valore true

$b = 5 * 4; //valore dell'espressione: 20; viene assegnato a $b
```

Precedentemente avevamo accennato ad una differenza nella valutazione dell'espressione fra i diversi modi di utilizzare gli operatori di incremento e di decremento. Vediamo ora di approfondire questo concetto:

```
$a = 10;
$b = 10;
++$a;  // incrementiamo $a, che diventa 11; l'espressione vale 11
$b++;  // anche $b diventa 11; qui però l'espressione vale 10
```

La differenza è questa: se usiamo l'operatore di incremento (o di decremento) prima della variabile, l'espressione assume il nuovo valore della variabile stessa. Se invece lo mettiamo dopo, l'espressione prenderà il valore che la variabile aveva prima dell'operazione. Di conseguenza:

```
$a = 5;
$b = 5;
echo ++$a; // $a diventa 6, e viene stampato '6'
echo $b++; // anche $b diventa 6, ma viene stampato '5'
echo ++$b; // a questo punto $b è diventato 7, e viene stampato '7'
```

Istruzione If

Con le strutture di controllo andiamo ad analizzare un altro degli aspetti fondamentali della programmazione: la **possibilità cioè di eseguire operazioni diverse**, ed eventualmente di eseguirle più volte, in base alla valutazione di determinate condizioni. In questa lezione esamineremo le istruzioni che ci permettono di eseguire o non eseguire certe porzioni di codice.

If

La principale di queste istruzioni è la `if`, la cui sintassi più elementare è la seguente:

```
if( <condizione> ) {
<codice>
}
```

Dopo l'`if`, deve essere indicata fra parentesi un'espressione da valutare (condizione). Questa espressione verrà valutata in senso booleano, cioè il suo valore sarà considerato vero o falso. Nel caso in cui l'espressione non abbia un valore booleano, PHP convertirà comunque questo valore in booleano utilizzando precise regole che vedremo tra qualche riga. Dunque, se la condizione è vera, l'istruzione successiva viene eseguita. In caso contrario, viene ignorata. Potremmo anche avere la necessità di eseguire, nel caso in cui la condizione sia vera, non una sola ma più istruzioni. Questo è perfettamente possibile, ma dobbiamo ricordarci di comprendere questo blocco di istruzioni fra due parentesi graffe, ad esempio così:

```
// ....
if ($nome == 'Luca') {
echo "ciao Luca!<br>";
echo "dove sono i tuoi amici?<br>";
}
echo "ciao a tutti voi";
```

In questo modo, se la variabile `$nome` ha effettivamente il valore 'Luca' vengono eseguite le due istruzioni successive, comprese fra le parentesi graffe. Dopo avere valutato la condizione ed eventualmente eseguito le due istruzioni previste, lo script proseguirà con ciò che sta fuori dalle parentesi graffe. Quindi nel nostro esempio la frase "ciao a tutti voi" viene prodotta in ogni caso. È buona norma usare comunque le parentesi graffe per delimitare il codice condizionato, anche quando è costituito da una sola istruzione: infatti questo rende il codice più leggibile, ed inoltre potrebbe evitarci degli errori se ci dovesse capitare di voler aggiungere delle istruzioni al blocco condizionato dimenticando di aggiungere le graffe.

Dobbiamo notare un particolare, riguardo alla sintassi di questa istruzione: per la prima volta, vediamo che in questo costrutto mancano i punto e virgola. Infatti la `if` e la condizione espressa fra parentesi non devono averli; continuiamo invece a metterli nel blocco di codice condizionato.

Abbiamo detto poco fa che la condizione espressa fra parentesi potrebbe non avere un valore booleano. PHP però è in grado di considerare booleano qualsiasi valore, in base ad alcune regole molto semplici. Facciamo un altro esempio banale:

```
if (5) {
print "ciao Luca!";
}
```

Questo codice, da un punto di vista logico, non ha nessun senso, ma ci permette di capire come PHP interpreta le nostre espressioni. Infatti in questo caso la stringa "ciao Luca!" verrà sempre stampata. Questo perchè, per PHP, il valore 5, così come qualsiasi numero diverso da 0, è considerato 'vero'. In sostanza, PHP considera come falsi:

- il valore numerico 0, nonchè una stringa che contiene '0'
- una stringa vuota
- un array con zero elementi
- un valore NULL, cioè una variabile che non è stata definita o che è stata eliminata con unset(), oppure a cui è stato assegnato il valore NULL esplicitamente

Qualsiasi altro valore, per PHP è un valore vero. Quindi qualsiasi numero, intero o decimale purchè diverso da 0, qualsiasi stringa non vuota, se usati come espressione condizionale saranno considerati veri. Quindi anche la banale espressione che abbiamo utilizzato nell'esempio di prima.

Analizziamo ora un caso che spesso crea problemi se non viene compreso fin dalle prime battute. Vediamo questo codice

```
$a = 7;
if ($a = 4)
echo '$a è uguale a 4!';
```

A prima vista, qualcuno potrebbe essere ingannato da queste istruzioni, soprattutto chi ha precedenti esperienze di programmazione in linguaggi strutturati diversamente, nei quali un'espressione come `$a = 4` potrebbe essere sia un'assegnazione che un test. Costoro infatti si aspetteranno che questo blocco di codice non stampi niente, e rimarrebbero molto sorpresi, qualora lo provassero, di vedere invece sul browser la frase '$a è uguale a 4'.

Questo succederebbe perchè l'espressione che abbiamo messo fra parentesi è un'assegnazione: cioè essa assegna alla variabile $a il valore 4. L'istruzione seguente viene quindi eseguita, per il motivo che abbiamo visto prima: il valore della nostra espressione è 4, che per PHP è un valore vero. Se invece che `if ($a = 4)` avessimo scritto `if ($a = 0)` l'istruzione seguente sarebbe stata saltata, perchè la nostra condizione avrebbe preso valore 0, cioè falso. Per ottenere il risultato corretto logicamente, avremmo dovuto utilizzare l'operatore di confronto:

```
$a = 7;
if ($a == 4)
echo '$a è uguale a 4!';
```

Ricordiamoci quindi, quando vogliamo verificare se il valore di una variabile è uguale a qualcosa, di utilizzare l'operatore condizionale di uguaglianza, cioè il doppio uguale ("=="). In caso contrario, non solo otterremo molto spesso risultati diversi da quelli che ci saremmo aspettati, ma avremo anche modificato il valore di una variabile che volevamo soltanto verificare.

Istruzioni Else e Elseif

Else

Andiamo ora un po' più a fondo nell'analisi dell'istruzione `if`: essa infatti ci permette non solo di indicare quali istruzioni vogliamo eseguire se la condizione è vera, ma anche di esprimere un blocco di codice da eseguire quando la condizione è falsa. Ecco come:

```
If (<condizione>) {
 <codice>
} else {
 <codice>
}
```

La parola chiave `else`, che significa 'altrimenti', deve essere posizionata subito dopo la parentesi graffa di chiusura del codice previsto per il caso 'vero' (o dopo l'unica istruzione prevista, se non abbiamo usato le graffe). Anche per 'else' valgono le stesse regole: niente punto e virgola, parentesi graffe obbligatorie se dobbiamo esprimere più di un'istruzione, altrimenti facoltative. Ovviamente il blocco di codice specificato per 'else' viene ignorato quando la condizione è vera, mentre viene eseguito se la condizione è falsa.

Le istruzioni `if` possono essere nidificate una dentro l'altra, consentendoci così di creare codice di una certa complessità. Esempio:

```
if ($nome == 'Luca') {
if ($cognome == 'Rossi') {
print "Luca Rossi è di nuovo fra noi";
} else {
print "Abbiamo un nuovo Luca!";
}
} else {
print "ciao $nome!";
}
```

In questo caso, abbiamo nidificato un ulteriore test all'interno del primo caso, quello in cui `$nome` ha il valore 'Luca'. Abbiamo infatti previsto un messaggio diverso, a seconda del valore della variabile `$cognome`.

Elseif

Un'ulteriore possibilità che ci fornisce l'istruzione `if` in PHP è quella di utilizzare la parola chiave `elseif`. Attraverso questa possiamo indicare una seconda condizione, da valutare solo nel caso in cui quella precedente risulti falsa. Indicheremo quindi, di seguito, il codice da eseguire nel

caso in cui questa condizione sia vera, ed eventualmente, con `else`, il codice previsto per il caso in cui anche la seconda condizione è falsa.

```
if ($nome == 'Luca') {
print "bentornato Luca!";
} elseif ($cognome == 'Verdi') {
print "Buongiorno, signor Verdi";
} else {
print "ciao $nome!";
}
```

In questo caso, abbiamo un'istruzione da eseguire quando `$nome` vale 'Luca'; nel caso in cui ciò non sia vero, è prevista una seconda istruzione se `$cognome` è 'Verdi'; se nemmeno questo è vero, allora verrà eseguita la terza istruzione. Da notare che, se `$nome` è 'Luca' e $cognome è 'Verdi', viene comunque eseguita solo la prima istruzione, perchè dopo avere verificato la condizione, tutti gli altri casi vengono saltati.

Istruzione Switch e operatore ternario

Switch

Passiamo ora a verificare una seconda istruzione che ci permette di prevedere diversi valori possibili per un'espressione ed eseguire codice specifico in base al valore:

```
switch (<condizione>) {
 case <valore 1>:
<codice>
break;
case <valore 1>:
<codice>
break;
....
default:
<codice>;
break;
}
```

L'istruzione `switch` prevede che indichiamo, fra parentesi, un'espressione che verrà valutata per il suo valore (questa volta non si tratta necessariamente di un valore booleano). Di seguito, tra parentesi graffe, esprimeremo una serie di espressioni da confrontare con quella indicata prima: dal momento in cui ne trova una il cui valore è uguale, PHP esegue il codice indicato di seguito, fino a quando non incontra un'istruzione `break`. Come possiamo vedere dall'esempio, le espressioni da confrontare con la prima vengono precedute dalla parola chiave `case` e seguite dai due punti. L'istruzione `default` può essere indicata come 'ultimo caso', che si considera verificato quando nessuno dei casi precedenti è risultato vero. L'indicazione `default` può anche essere assente.

È molto importante comprendere la funzione dell'istruzione `break` in questa situazione: infatti, quando PHP verifica uno dei casi, esegue non solo il codice che trova subito dopo, ma anche tutto quello che trova di seguito, compreso quello relativo ai casi seguenti. Questo fino a quando non trova, appunto, un'istruzione `break`. Se non mettessimo un'istruzione `break` alla fine di un blocco di codice relativo ad un caso particolare, l'esecuzione continuerebbe eseguendo anche il case successivo. Questo comportamento ci permette però di prevedere un unico comportamento per più

valori dell'espressione sotto esame:

```
switch ($nome) {
case 'Luca':
case 'Giorgio':
case 'Franco':
print "Ciao, vecchio amico!";
break;
case 'Mario':
print "Ciao, Mario!";
break;
case 'Paolo':
print "Finalmente, Paolo!";
break;
default:
print "Benvenuto, chiunque tu sia";
}
```

In questo caso, abbiamo previsto un unico messaggio per il caso in cui la variabile `$nome` valga 'Luca', 'Giorgio' o 'Franco'.

L'operatore ternario

L'operatore ternario è così chiamato perchè è formato da tre espressioni: il valore restituito è quello della seconda o della terza di queste espressioni, a seconda che la prima sia vera o falsa. In pratica, si può considerare, in certi casi, una maniera molto sintetica di effettuare una `if`.

($altezza >= 180) ? 'alto' : 'normale' ;

Questa espressione prenderà il valore 'alto' se la variabile `$altezza` è maggiore o uguale a 180, e 'normale' nel caso opposto. Come vediamo, l'espressione condizionale è contenuta fra parentesi e seguita da un punto interrogativo, mentre due punti separano la seconda espressione dalla terza. Questo costrutto può essere utilizzato, ad esempio, per valorizzare velocemente una variabile senza ricorrere all'`if`:

$tipologia = ($altezza >= 180) ? 'alto' : 'normale';

equivale a scrivere

```
if ($altezza >= 180) $tipologia = 'alto' ;
else
$tipologia = 'normale';
```

Come potete vedere, in termini di spazio il risparmio è abbastanza significativo. Bisogna però stare attenti ad abusare di questa forma, perchè a volte può rendere più difficoltosa la leggibilità del nostro script, anche se risulta di indubbia utilità nel caso in cui si necessiti rendere più compatto il codice.

I cicli: for, while e do

I cicli sono un altro degli elementi fondamentali di qualsiasi linguaggio di programmazione, in quanto ci permettono di **eseguire determinate operazioni in maniera ripetitiva**. È una necessità che si presenta molto spesso: infatti non è raro che un programma o uno script debbano elaborare quantità anche molto grosse di dati; in questa situazione, si prevederà il trattamento da utilizzare per

ogni singolo dato (o gruppo di dati correlati), che sarà poi ripetuto per tutte le ricorrenze di tali dati.

Ciclo For

Iniziamo subito con un esempio (come al solito abbastanza banale): supponiamo di voler mostrare i multipli da 1 a 10 di un numero, ad esempio 5. La prima soluzione è quella di usare il ciclo `for`:

```
for ($mul = 1; $mul <= 10; ++$mul) {
$ris = 5 * $mul;
echo "5 * $mul = $ris <br/>";
}
```

Questo costrutto, simile a quello usato in altri linguaggi, utilizza la parola chiave `for`, seguita, fra parentesi, dalle istruzioni per definire il ciclo; di seguito, si racchiudono fra parentesi graffe tutte le istruzioni che devono essere eseguite ripetutamente. Le tre istruzioni inserite fra le parentesi tonde e separate da punto e virgola vengono trattate in questo modo: la **prima** viene eseguita una sola volta, all'inizio del ciclo; la **terza** viene eseguita alla fine di ogni iterazione del ciclo; la **seconda** deve essere una condizione, e viene valutata prima di ogni iterazione del ciclo: quando risulta falsa, l'esecuzione del ciclo viene interrotta, ed il controllo passa alle istruzioni dopo le parentesi graffe. Quando invece è vera, le istruzioni fra parentesi graffe vengono eseguite. Ovviamente è possibile che tale condizione risulti falsa fin dal primo test: in questo caso, le istruzioni contenute fra le parentesi graffe non saranno eseguite nemmeno una volta.

Il formato standard è quindi quello che abbiamo visto sopra, che utilizza le parentesi tonde per definire un 'contatore': con la prima istruzione lo si inizializza, con la seconda lo si confronta con un valore limite oltre il quale il ciclo deve terminare, con la terza lo si incrementa dopo ogni esecuzione.

Con il ciclo `for`, così come con tutti gli altri cicli, è molto importante stare attenti a non creare una **situazione in cui il ciclo non raggiunge mai una via d'uscita** (il cosiddetto 'loop'): in questo caso, infatti, lo script rieseguirebbe il nostro ciclo all'infinito. Nel nostro esempio di prima potrebbe succedere, ad esempio, se sbagliassimo a scrivere il nome della variabile `$mul` nell'istruzione di incremento: se avessimo scritto `$mus++`, avremmo incrementato questa variabile, che non viene utilizzata nel ciclo, dopo ciascuna delle sue esecuzioni; in questo modo `$mul` rimarrebbe sempre uguale ad 1, ed il nostro ciclo stamperebbe all'infinito "5 * 1 = 5", perchè `$mul` non diventerebbe mai maggiore di 10.

In molte situazioni classiche di programmazione, un errore di questo genere potrebbe costringerci a forzare la chiusura del programma o addirittura a spegnere la macchina: nel caso di PHP, questo di solito non succede, in quanto gli script PHP hanno un limite di tempo per la loro esecuzione, oltre il quale si arrestano. Tale limite è normalmente di 30 secondi, ed è comunque impostabile attraverso il file `php.ini`.

Il ciclo While

Vediamo ora un altro tipo di ciclo, più semplice nella sua costruzione: il ciclo `while`. Questo si può considerare come una specie di `if` ripetuta più volte: infatti la sua sintassi prevede che alla parola chiave `while` segua, fra parentesi, la condizione da valutare, e fra parentesi graffe, il codice da rieseguire fino a quando tale condizione rimane vera. Vediamo con un esempio come ottenere lo stesso risultato dell'esempio precedente:

```
$mul = 1;
while ($mul <= 10) {
$ris = 5 * $mul;
```

```
print("5 * $mul = $ris<br>");
$mul++;
}
```

Il ciclo `while`, rispetto al `for`, non ci mette a disposizione le istruzioni per inizializzare e per incrementare il contatore: quindi dobbiamo inserire queste istruzioni nel flusso generale del codice, per cui mettiamo l'inizializzazione prima del ciclo, e l'incremento all'interno del ciclo stesso, in fondo. Anche in questa situazione, comunque, il concetto fondamentale è che l'esecuzione del ciclo termina quando la condizione fra parentesi non è più verificata: ancora una volta, quindi, è possibile che il ciclo non sia eseguito mai, nel caso in cui la condizione risulti falsa fin da subito.

Il ciclo do...while

Esiste invece un'altra forma, simile al `while`, con la quale possiamo assicurarci che il codice indicato tra le parentesi graffe venga eseguito almeno una volta: si tratta del `do...while`

```
$mul = 11;
do {
$ris = 5 * $mul;
print("5 * $mul = $ris<br>");
$mul++;
} while ($mul <= 10)
```

Con questa sintassi, il `while` viene spostato dopo il codice da ripetere, ad indicare che la valutazione della condizione viene eseguita solo dopo l'esecuzione del codice fra parentesi graffe. Nel caso del nostro esempio, abbiamo inizializzato la variabile $mul col valore 11, per creare una situazione nella quale, con i cicli visti prima, non avremmo ottenuto alcun output, mentre con l'uso del `do...while` il codice viene eseguito una volta nonostante la condizione indicata fra parentesi sia falsa fin dall'inizio. L'esempio stamperà quindi "5 * 11 = 55".

Uscire da un ciclo

Abbiamo visto che PHP termina l'esecuzione di un ciclo quando la condizione a cui è sottoposto non è più verificata. Abbiamo però a disposizione altri strumenti per modificare il comportamento del nostro script dall'interno del ciclo: infatti possiamo dire a PHP di non completare la presente iterazione e passare alla successiva (con `continue`) oppure di interrompere definitivamente l'esecuzione del ciclo (con `break`). Vediamo un esempio:

```
for ($ind = 1; $ind < 500; $ind++) {
if ($ind % 100 == 0) {
break;
}elseif ($ind % 25 == 0) {
continue;
}
echo "valore: $ind <br/>";
}
```

Questo codice imposta un ciclo per essere eseguito 500 volte, con valori di $ind che vanno da 1 a 500. In realtà, le istruzioni che abbiamo posto al suo interno fanno sì che la stampa del valore di $ind non venga eseguita ogni volta che `$ind` corrisponde ad un multiplo di 25 (infatti l'istruzione 'continue' fa sì che PHP salti la 'print' e passi direttamente all'iterazione successiva, incrementando la variabile), e che il ciclo si interrompa del tutto quando $ind raggiunge il valore 100.

Ciclo Foreach

Esiste un ultimo tipo di ciclo, ed è un ciclo particolare perchè è costruito appositamente per il trattamento degli array e di particolari oggetti chiamati Iteratori: si tratta del `foreach`. Questo ci permette di costruire un ciclo che viene ripetuto per ogni elemento di una collezione passata come argomento. La sintassi è la seguente:

```
foreach ($arr as $chiave => $valore) {
 <codice>
}
```

All'interno delle parentesi graffe avremo a disposizione, nelle due variabili che abbiamo definito `$chiave` e `$valore`, l'indice e il contenuto dell'elemento che stiamo trattando. È da notare che, nel caso ci interessi soltanto il valore e non la chiave (ad esempio perchè le chiavi sono numeriche), possiamo anche evitare di estrarre la chiave stessa:

```
foreach ($arr as $valore) {
<codice>
}
```

Gli array

Precedentemente abbiamo accennato il tipo di dato array con il seguente esempio:

$colori = array('bianco', 'nero', 'giallo', 'verde', 'rosso');

Usando questo tipo di definizione, PHP associa a ciascuno dei valori che abbiamo elencato un indice numerico, a partire da 0. Quindi, in questo caso, 'bianco' assumerà l'indice 0, 'nero' l'indice 1, e così via fino a 'rosso' che avrà indice 4. Per riferirsi ad un singolo elemento dell'array si indica il nome dell'array seguito dall'indice contenuto fra parentesi quadre:

echo $colori[2]; //stampa 'giallo'

Esiste poi un metodo per aggiungere un valore all'array; questo metodo può essere usato anche, come alternativa al precedente, per definire l'array:

$colori[] = 'blu';

Con questo codice verrà creato un nuovo elemento nell'array `$colori`, che avrà l'indice 5. Questa sintassi infatti può essere "tradotta" come "aggiungi un elemento in fondo all'array $colori". Come abbiamo detto, questa sintassi è valida anche per definire un array, in alternativa a quella usata prima: infatti, se ipotizziamo che l'array `$colori` non fosse ancora definito, questa istruzione lo avrebbe definito creando l'elemento con indice 0. È naturalmente possibile anche indicare direttamente l'indice, anche in modo non consecutivo:

```
$colori[3] = 'arancio';
$colori[7] = 'viola';
```

Dopo questa istruzione, l'elemento con indice 3, che prima valeva 'verde', avrà il valore cambiato in 'arancio'. Inoltre avremo un nuovo elemento, con indice 7, con il valore 'viola'. È da notare che, dopo queste istruzioni, il nostro array ha un "buco", perchè dal codice 5 si salta direttamente al codice 7: successivamente, se useremo di nuovo l'istruzione di "incremento" con le parentesi quadre vuote, il nuovo elemento prenderà l'indice 8. Infatti PHP, quando gli proponiamo un'istruzione di quel tipo, va a cercare l'elemento con l'indice più alto, e lo aumenta di 1 per creare quello nuovo.

Ma l'argomento array non si limita a questo: infatti gli indici degli elementi non sono necessariamente numerici. Possono essere anche delle stringhe:

$persona['nome'] = 'Mario';

Con questa istruzione abbiamo definito un array di nome `$persona`, creando un elemento la cui chiave è 'nome' ed il cui valore è 'Mario'. È da ricordare che le chiavi numeriche ed associative possono coesistere nello stesso array. Vediamo un esempio banale, ipotizzando la formazione di una squadra di calcio:

```
$formazione[1] ='Buffon';
$formazione[2] ='Panucci';
$formazione[3] ='Nesta';
$formazione[4] ='Cannavaro';
$formazione[5] ='Coco';
$formazione[6] ='Ambrosini';
$formazione[7] ='Tacchinardi';
$formazione[8] ='Perrotta';
$formazione[9] ='Totti';
$formazione[10] ='Inzaghi';
$formazione[11] ='Vieri';
$formazione['ct'] = 'Trapattoni';
```

In questo caso abbiamo creato un array con dodici elementi, di cui undici con chiavi numeriche, ed uno con chiave associativa. Se in seguito volessimo aggiungere un elemento usando le parentesi quadre vuote, il nuovo elemento prenderà l'indice 12. Avremmo potuto creare lo stesso array usando l'istruzione di dichiarazione dell'array, così:

$formazione = array(1 => 'Buffon', 'Panucci', 'Nesta', 'Cannavaro', 'Coco', 'Ambrosini', 'Tacchinardi', 'Perrotta', 'Totti', 'Inzaghi', 'Vieri', 'ct' => 'Trapattoni');

Analizziamo il formato di questa istruzione: per prima cosa abbiamo creato il primo elemento, assegnandogli esplicitamente la chiave 1. Come possiamo vedere, il sistema per fare ciò è di indicare la chiave, seguita dal simbolo **=>** e dal valore dell'elemento. Se non avessimo indicato 1 come indice, PHP avrebbe assegnato al primo elemento l'indice 0. Per gli elementi successivi, ci siamo limitati ad elencare i valori, in quanto PHP, per ciascuno di essi, crea la chiave numerica aumentando di 1 la più alta già esistente. Quindi 'Panucci' prende l'indice 2, 'Nesta' il 3 e così via. Arrivati all'ultimo elemento, siccome vogliamo assegnargli una chiave associativa, siamo obbligati ad indicarla esplicitamente.

È da notare che quando abbiamo usato le chiavi associative le abbiamo **indicate fra apici**: ciò è necessario per mantenere la 'pulizia' del codice, in quanto, se non usassimo gli apici (come spesso si vede fare), PHP genererebbe un errore di tipo 'notice', anche se lo script funzionerebbe ugualmente (dato che il valore verrebbe convertito automaticamente in una stringa). Vediamo ora qualche esempio di creazione e stampa dei valori di un array:

$persona['nome'] = 'Mario'; //corretto
$persona[cognome] = 'Rossi'; /*funziona, ma genera un errore 'notice'*/
echo $persona['cognome']; //stampa 'Rossi': corretto
echo "ciao $persona[nome]"; /*stampa 'ciao Mario': corretto (niente apici fra virgolette)*/
echo "ciao $persona['nome']"; //NON FUNZIONA, GENERA ERRORE
echo "ciao {$persona['nome']}"; /*corretto: per usare gli apici fra virgolette dobbiamo comprendere il tutto fra parentesi graffe*/
echo "ciao " . $persona['nome']; /*corretto: come alternativa, usiamo il punto per concatenare (v. lez.10 sugli operatori)*/

Abbiamo così visto in quale maniera possiamo creare ed assegnare valori agli array, usando indici

numerici o associativi, impostando esplicitamente le chiavi o lasciando che sia PHP ad occuparsene. Vediamo ora in che modo possiamo creare strutture complesse di dati, attraverso gli array a più dimensioni.

Un array a più dimensioni è un array nel quale uno o più elementi sono degli array a loro volta. Supponiamo di voler raccogliere in un array i dati anagrafici di più persone: per ogni persona registreremo nome, cognome, data di nascita e città di residenza

```
$persone = array( array('nome' => 'Mario', 'cognome' => 'Rossi', 'data_nascita' => '1973/06/15', 'residenza' => 'Roma'), array('nome' => 'Paolo', 'cognome' => 'Bianchi', 'data_nascita' => '1968/04/05', 'residenza' => 'Torino'), array('nome' => 'Luca', 'cognome' => 'Verdi', 'data_nascita' => '1964/11/26', 'residenza' => 'Napoli'));
print $persone[0]['cognome']; // stampa 'Rossi'
print $persone[1]['residenza']; // stampa 'Torino'
print $persone[2]['nome']; // stampa 'Luca'
```

Con questo codice abbiamo definito un array **formato a sua volta da tre array**, che sono stati elencati separati da virgole, per cui ciascuno di essi ha ricevuto l'indice numerico a partire da 0. All'interno dei singoli array, invece, tutte le chiavi sono state indicate come associative. Da notare che, sebbene in questo caso ciascuno dei tre array 'interni' abbia la stessa struttura, in realtà è possibile dare a ciascun array una struttura autonoma. Vediamo un altro esempio:

```
$persone = array( 1 => array('nome' => 'Mario Rossi', 'residenza' => 'Roma', 'ruolo' => 'impiegato'), 2 => array('nome' => 'Paolo Bianchi', 'data_nascita' => '1968/04/05', 'residenza' => 'Torino'), 'totale_elementi' => 2);
print $persone[1]['residenza']; // stampa 'Roma'
print $persone['totale_elementi']; // stampa '2'
```

In questo caso il nostro array è formato da due array, ai quali abbiamo assegnato gli indici 1 e 2, e da un terzo elemento, che non è un array ma una variabile intera, con chiave associativa 'totale_elementi'. I due array che costituiscono i primi due elementi hanno una struttura diversa: mentre il primo è formato dagli elementi 'nome', 'residenza' e 'ruolo', il secondo è formato da 'nome', 'data_nascita' e 'residenza'.

Le funzioni in PHP: gestire le variabili

Anche se la versione 5 ha portato un grosso supporto per la programmazione ad oggetti (http://php.html.it/guide/leggi/167/guida-programmazione-ad-oggetti-con-php-5/) (che tratteremo brevemente anche nei paragrafi successivi) le funzioni rimangono comunque la parte fondamentale del linguaggio dato che PHP è nato come **linguaggio procedurale**, e tutt'oggi molti sviluppatori continuano a seguire questa filosofia di programmazione.

Una funzione è un insieme di istruzioni che hanno lo scopo (la funzione, appunto) di eseguire determinate operazioni. La praticità delle funzioni sta nel fatto che ci consentono di non dover riscrivere tutto il codice ogni volta che abbiamo la necessità di eseguire quelle operazioni comuni: ci basta infatti richiamare l'apposita funzione, fornendole i parametri, cioè i dati/informazioni, di cui ha bisogno per la sua esecuzione.

Le funzioni possono essere **incorporate nel linguaggio** oppure essere **definite dall'utente**. In entrambi i casi, il modo di richiamarle è lo stesso.

La sintassi fondamentale con la quale si richiama una funzione è molto semplice:

```
nome_funzione();
```

Si tratta semplicemente di indicare il nome della funzione, seguito da parentesi tonde. Queste parentesi devono contenere i parametri da passare alla funzione, ma vanno obbligatoriamente indicate anche se non ci sono parametri, nel qual caso rimangono vuote come nell'esempio che abbiamo visto sopra. Nel caso poi in cui la funzione restituisca un valore, possiamo indicare la variabile in cui immagazzinarlo:

```
$valore = nome_funzione();
```

In questo modo, la variabile `$valore` riceverà il risultato della funzione. Le funzioni possono essere utilizzate anche all'interno di espressioni: in tal caso il valore restituito verrà utilizzato durante la valutazione dell'espressione:

```
$prova = (10 * numero_anni()) - numero_casuale();
```

Abbiamo detto che le funzioni possono essere incorporate nel linguaggio oppure definite dall'utente: in questa lezione passeremo in rassegna alcune fra le funzioni incorporate maggiormente utilizzate in PHP, tenendo però presente che tali funzioni sono numerosissime, rivolte agli scopi più disparati, e che quindi non ne avremo che una visione molto parziale. Consiglio la consultazione del manuale online per una trattazione più completa delle funzionalità e dei casi particolari.

Cominciamo dalle principali funzioni che operano sulle variabili in generale:

`empty(valore)`
: verifica se la variabile che le passiamo è vuota oppure no. Per 'vuota' si intende che la variabile può contenere una stringa vuota o un valore numerico pari a 0, ma può anche essere non definita o essere impostata al valore NULL (l'eventuale indicazione di una variabile non definita, in questo caso, non genera errore notice). Restituisce un valore booleano (vero o falso).

`isset(valore)`
: verifica se la variabile è definita. Una variabile risulta non definita quando non è stata inizializzata o è stata impostata col valore `NULL`. Restituisce un valore booleano.

`is_null(valore)`
: verifica se la variabile equivale a `NULL`, ma genera un errore 'notice' se viene eseguito su una variabile non definita. Restituisce un valore booleano.

`is_int(valore), is_integer(valore), is_long(valore)`
: verifica se la variabile è di tipo intero. Le tre funzioni sono equivalenti. Restituiscono un valore booleano.

`is_float(valore), is_double(valore), is_real(valore)`
: verifica se la variabile è di tipo numerico double (o float). Le tre funzioni sono equivalenti. Restituiscono un valore booleano.

`is_string(valore)`
: verifica se la variabile è una stringa. Restituisce un valore booleano.

`is_array(valore)`
: verifica se la variabile è un array. Restituisce un valore booleano.

`is_numeric(valore)`
: verifica se la variabile contiene un valore numerico. È molto importante la distinzione fra questa funzione e `is_int()` o `is_float()`, perchè queste ultime, nel caso di una stringa che contiene valori numerici, restituiscono falso, mentre `is_numeric()` restituisce vero. Restituisce un valore booleano.

`gettype(valore)`
: verifica quale tipo di dato le abbiamo passato. Restituisce una stringa che rappresenta il tipo di dato, ad esempio: `boolean`, `integer`, `double`, `string`, `array`. È bene però, in uno

script, non fare affidamento su questi valori per dedurre il tipo di dato, perchè in versioni future di PHP alcune di queste stringhe potrebbero essere modificate. Meglio usare le funzioni viste prima.

`print_r(valore)`
: stampa (direttamente sul browser) informazioni relative al contenuto della variabile che le abbiamo passato. È utile in fase di debug, quando a seguito di comportamenti 'strani' del nostro script vogliamo verificare il contenuto di certi dati. Se il valore passato è un array, la funzione ne evidenzia le chiavi ed i valori relativi. Restituisce un valore booleano.

`unset(valore)`
: distrugge la variabile specificata. In realtà non si tratta di una funzione, ma di un costrutto del linguaggio, e ne abbiamo già parlato nella lez. 8. Dopo l'`unset()`, l'esecuzione di `empty()` o `is_null()` sulla stessa variabile restituirà vero, mentre `isset()` restituirà falso. Non restituisce valori.

Vediamo ora qualche esempio per chiarire l'utilizzo di queste funzioni:

```
$b = empty($a); // $a non è ancora definita, quindi $b sarà vero
$a = 5;
$b = isset($a);     // vero
$b = is_float($a);  // falso: $a è un intero
$b = is_string($a); // falso

$a = '5';
$b = is_int($a);     // falso: $a ora è una stringa
$b = is_string($a);  // vero
$b = is_numeric($a); // vero: la stringa ha un contenuto numerico
$c = gettype($b);    // $c prende il valore 'boolean'

unset($a);        // eliminiamo la variabile $a;
$b = is_null($a); // vero, ma genera errore
```

In questi esempi abbiamo sempre assegnato alla variabile `$b` i valori booleani restituiti dalle funzioni. Nella pratica, è più frequente che tali valori non siano memorizzati in variabili, ma che vengano usati direttamente come condizioni, ad esempio per delle istruzioni di tipo `if`. Ancora un paio di esempi:

```
if (empty($a)) {
  print ('$a è vuota o non definita!');
} else {
  print ('$a contiene un valore');
}

if (is_numeric($a)){
  print ('$a contiene un valore numerico');
} else {
  print ('$a non contiene un numero');
}
```

Le funzioni in PHP: gestire le stringhe

Passiamo ora ad esaminare alcune funzioni che operano sulle stringhe (l'eventuale indicazione di un parametro tra parentesi quadre indica che quel parametro è facoltativo, quindi può non essere indicato nel momento in cui si chiama la funzione):

`strlen(stringa)`
: verifica la lunghezza della stringa, cioè il numero di caratteri che la compongono. Restituisce un numero intero.

`trim(stringa)`
: elimina gli spazi all'inizio e alla fine della stringa. Restituisce la stringa modificata.

`ltrim(stringa)`
: elimina gli spazi all'inizio della stringa. Restituisce la stringa modificata.

`rtrim(stringa)`
: elimina gli spazi alla fine della stringa. Restituisce la stringa modificata.

`substr(stringa, intero [, intero])`
: restituisce una porzione della stringa, in base al secondo parametro (che indica l'inizio della porzione da estrarre), e all'eventuale terzo parametro, che indica quanti caratteri devono essere estratti. Se il terzo parametro non viene indicato, viene restituita tutta la parte finale della stringa a partire dal carattere indicato.

  I caratteri vanno contati a partire da zero, per cui se si chiama la funzione con `substr(stringa, 4)` verranno restituiti tutti i caratteri a partire dal quinto. Si può anche indicare un numero negativo come carattere iniziale: in questo caso, il carattere iniziale della porzione di stringa restituita verrà contato a partire dal fondo.

  Ad esempio, con `substr(stringa, -5, 3)` si otterranno tre caratteri a partire dal quintultimo (da notare che in questo caso il conteggio non inizia da zero, ma da 1: cioè `-1` indica l'ultimo carattere, `-2` il penultimo e così via).

  Se infine si indica un numero negativo come terzo parametro, tale parametro non verrà più utilizzato come numero di caratteri restituiti, ma come numero di caratteri non restituiti a partire dal fondo. Esempio: `substr(stringa, 3, -2)` restituisce i caratteri dal quarto al terzultimo. La funzione restituisce la porzione di stringa richiesta.

`str_replace(stringa, stringa, stringa)`
: effettua una sostituzione della prima stringa con la seconda all'interno della terza. Ad esempio: `str_replace('p', 't', 'pippo')` sostituisce le 'p' con le 't' all'interno di 'pippo', e quindi restituisce 'titto'. Restituisce la terza stringa modificata. Esiste anche la funzione `str_ireplace()`, che è equivalente ma che cerca la prima stringa nella terza senza tener conto della differenza fra maiuscole e minuscole.

`strpos(stringa, stringa)`
: cerca la posizione della seconda stringa all'interno della prima. Ad esempio: `strpos('Lorenzo', 're')` restituisce 2, ad indicare la terza posizione. Restituisce un intero che rappresenta la posizione a partire da 0 della stringa cercata. Se la seconda stringa non è presente nella prima, restituisce il valore booleano FALSE. La funzione `stripos()` fa la stessa ricerca senza tenere conto della differenza fra maiuscole e minuscole.

`strstr(stringa, stringa)`
: cerca la seconda stringa all'interno della prima, e restituisce la prima stringa a partire dal punto in cui ha trovato la seconda. `strstr('Lorenzo', 're')` restituisce 'renzo'. Restituisce una stringa se la ricerca va a buon fine, altrimenti il valore booleano FALSE. La funzione stristr() funziona allo stesso modo ma non tiene conto della differenza fra maiuscole e minuscole.

`strtolower(stringa)`
: converte tutti i caratteri alfabetici nelle corrispondenti lettere minuscole. Restituisce la stringa modificata.

`strtoupper(stringa)`

converte tutti i caratteri alfabetici nelle corrispondenti lettere maiuscole. Restituisce la stringa modificata.

`ucfirst(stringa)`
: trasforma in maiuscolo il primo carattere della stringa. Restituisce la stringa modificata.

`ucwords(stringa)`
: trasforma in maiuscolo il primo carattere di ogni parola della stringa, intendendo come parola una serie di caratteri che segue uno spazio. Restituisce la stringa modificata.

`explode(stringa, stringa [, intero])`
: trasforma la seconda stringa in un array, usando la prima per separare gli elementi. Il terzo parametro può servire ad indicare il numero massimo di elementi che l'array può contenere (se la suddivisione della stringa portasse ad un numero maggiore, la parte finale della stringa sarà interamente contenuta nell'ultimo elemento).

  Ad esempio: `explode(' ', 'ciao Mario')` restituisce un array di due elementi in cui il primo è 'ciao' e il secondo 'Mario'. Restituisce un array.

Dobbiamo fare un'annotazione relativa a tutte queste funzioni, in particolare quelle che hanno lo scopo di modificare una stringa: la stringa modificata è il risultato della funzione, che dovremo assegnare ad una variabile apposita. Le variabili originariamente passate alla funzione rimangono invariate. Vediamo alcuni esempi sull'uso di queste funzioni:

```
$a = 'IERI ERA DOMENICA';

/* $b diventa 'ieri era domenica',
 * ma $a rimane 'IERI ERA DOMENICA'
 */
$b = strtolower($a);

strlen('abcd');                     // restituisce 4
trim(' Buongiorno a tutti ');       // restituisce 'Buongiorno a tutti'
substr('Buongiorno a tutti', 4);    // 'giorno a tutti' (inizia dal quinto)
substr('Buongiorno a tutti', 4, 6); // 'giorno'(6 caratteri a partire dal
quinto)
substr('Buongiorno a tutti', -4);   // 'utti' (ultimi quattro)
substr('Buongiorno a tutti', -4, 2); // 'ut' (2 caratteri a partire dal
quartultimo)
substr('Buongiorno a tutti, 4, -2); // 'giorno a tut' (dal quinto al
terzultimo)


str_replace('Buongiorno', 'Ciao', 'Buongiorno a tutti'); // 'Ciao a tutti'
str_replace('dom', 'x', 'Domani è domenica');           // 'Domani è xenica'
str_ireplace('dom', 'x', 'Domani è domenica');          // 'xani è xenica'

strpos('Domani è domenica', 'm'); // 2 (prima 'm' trovata)
strstr('Domani è domenica', 'm'); // 'mani è domenica' (a partire dalla prima
'm')

strtoupper('Buongiorno a tutti'); // 'BUONGIORNO A TUTTI'
ucfirst('buongiorno a tutti');    // 'Buongiorno a tutti';
ucwords('buongiorno a tutti');    // 'Buongiorno A Tutti';

/* suddivide la stringa in un array, separando un elemento
 * ogni volta che trova una virgola; avremo quindi un array
 * di tre elementi: ('Alberto','Mario','Giovanni')
 */
```

```
explode(',','Alberto,Mario,Giovanni');

/* in questo caso l'array può contenere al massimo due elementi,
 * per cui nel primo elemento andrà 'Alberto' e nel secondo il
 * resto della stringa: 'Mario,Giovanni'
 */
explode(',','Alberto,Mario,Giovanni',2);
```

Le funzioni in PHP: gestire gli array

Vediamo ora alcune funzioni che operano sugli array:

`count(array)`
: conta il numero di elementi dell'array. Restituisce un intero.

`array_reverse(array [, booleano])`
: inverte l'ordine degli elementi dell'array. Se vogliamo mantenere le chiavi dell'array di input, dobbiamo passare il secondo parametro con valore TRUE. Restituisce l'array di input con gli elementi invertiti.

`sort(array)`
: ordina gli elementi dell'array. Bisogna fare attenzione, perchè questa funzione, contrariamente a molte altre, modifica direttamente l'array che le viene passato in input, che quindi andrà perso nella sua composizione originale. I valori vengono disposti in ordine crescente secondo i criteri che abbiamo visto nella lezione 10. Le chiavi vanno perse: dopo il sort, l'array avrà chiavi numeriche a partire da 0 secondo il nuovo ordinamento. Non restituisce nulla.

`rsort(array)`
: ordina gli elementi dell'array in ordine decrescente. Anche questa funzione modifica direttamente l'array passato in input e riassegna le chiavi numeriche a partire da 0. Non restituisce nulla.

`asort(array)`
: funziona come `sort()`, con la differenza che vengono mantenute le chiavi originarie degli elementi. Non restituisce nulla.

`arsort(array)`
: come `rsort()`, ordina in modo decrescente; mantiene però le chiavi originarie. Non restituisce nulla.

`in_array(valore, array)`
: cerca il valore all'interno dell'array. Restituisce un valore booleano: vero o falso a seconda che il valore cercato sia presente o meno nell'array.

`array_key_exists(valore, array)`
: cerca il valore fra le chiavi (e non fra i valori) dell'array. Restituisce un valore booleano.

`array_search(valore, array)`
: cerca il valore nell'array e ne indica la chiave. Restituisce la chiave del valore trovato o, se la ricerca non va a buon fine, il valore FALSE.

`array_merge(array, array [, array...])`
: fonde gli elementi di due o più array. Gli elementi con chiavi numeriche vengono accodati l'uno all'altro e le chiavi rinumerate. Le chiavi associative invece vengono mantenute, e nel caso vi siano più elementi nei diversi array con le stesse chiavi associative, l'ultimo sovrascrive i precedenti. Restituisce l'array risultante dalla fusione.

`array_pop(array)`
: estrae l'ultimo elemento dell'array, che viene 'accorciato'. Restituisce l'elemento in fondo

all'array e, contemporaneamente, modifica l'array in input togliendogli lo stesso elemento.

`array_push(array, valore [,valore...])`
: accoda i valori indicati all'array. Equivale all'uso dell'istruzione di accodamento $array[]=$valore , con il vantaggio che ci permette di accodare più valori tutti in una volta. Restituisce il numero degli elementi dell'array dopo l'accodamento.

`array_shift(array)`
: estrae un elemento come `array_pop()`, ma in questo caso si tratta del primo. Anche in questo caso l'array viene 'accorciato', ed inoltre gli indici numerici vengono rinumerati. Rimangono invece invariati quelli associativi. Restituisce l'elemento estratto dall'array.

`array_unshift(array, valore [,valore...])`
: inserisce i valori indicati in testa all'array. Restituisce il numero degli elementi dell'array dopo l'inserimento.

`implode(stringa, array)`
: è la funzione opposta di `explode()`, e serve a riunire in un'unica stringa i valori dell'array. La stringa indicata come primo parametro viene interposta fra tutti gli elementi dell'array. Restituisce la stringa risultato dell'aggregazione. Suo sinonimo è `join()`.

Ecco qualche esempio sull'uso di queste funzioni:

```
$arr = array('Luca', 'Giovanni', 'Matteo', 'Paolo', 'Antonio', 'Marco',
'Giuseppe');

$n = count($arr);            // $n vale 7
$arr1 = array_reverse($arr); // $arr1 avrà gli elementi invertiti, da 'Giuseppe'
a 'Luca'
echo $arr[1], '<br>';        // 'Giovanni'
echo $arr1[1], '<br>';       // 'Marco'

/* ora $arr sarà:
 * 'Antonio', 'Giovanni', 'Giuseppe', 'Luca', 'Marco', ' Matteo', 'Paolo'
 */
sort($arr);

$a = in_array('Giovanni', $arr);  // $a è vero (TRUE)
$a = in_array('Francesco', $arr); // $a è falso (FALSE)
$ultimo = array_pop($arr);        // $ultimo è 'Paolo' (li avevamo ordinati!)
$ultimo = array_pop($arr);        // ora $ultimo è 'Matteo', e in $arr sono
rimasti 5 elementi
$primo = array_shift($arr);       // primo è 'Antonio'

/* 'Matteo' e 'Antonio' vengono reinseriti in testa all'array;
 * $a riceve il valore 6
 */
$a = array_unshift($arr, $ultimo, $primo);

$stringa = implode(' ', $arr); // $stringa diventa 'Matteo Antonio Giovanni
Giuseppe Luca Marco' */

/* $new_arr conterrà 13 elementi:
 * 'Matteo', 'Antonio', 'Giovanni',
 * 'Giuseppe', 'Luca', 'Marco' (questi sono i 6 provenienti da $arr),
 * 'Giuseppe', 'Marco',' Antonio', 'Paolo',
 * 'Matteo', 'Giovanni', 'Luca' (questi sono i 7 di $arr1). Gli indici andranno
da 0 a 12.
 */
$new_arr = array_merge($arr, $arr1);
```

```
// Impostiamo ora un array con chiavi associative:
$famiglia = array('padre' => 'Claudio', 'madre' => 'Paola', 'figlio' => 'Marco',
'figlia' => 'Elisa');
// creiamo una copia del nostro array per poter fare esperimenti
$fam1 = $famiglia;
// ora $fam1 sarà 'Paola', 'Marco', 'Elisa', 'Claudio', con chiavi da 0 a 3
rsort($fam1);

$fam1 = $famiglia; // ripristiniamo l'array originale

/* di nuovo $fam1 sarà 'Paola', 'Marco', 'Elisa', 'Claudio',
 * ma ciascuno con la sua chiave originale
 * ('madre', 'figlio', 'figlia', 'padre')
 */
arsort($fam1);

$a = array_key_exists('figlia', $fam1); // $a è TRUE
$a = array_key_exists('zio', $fam1);    // $a è FALSE
$a = array_search('Claudio', $fam1);    // $a è 'padre'
$a = array_search('Matteo', $fam1);     // $a è FALSE
```

Le funzioni in PHP: gestire le date

Concludiamo questa panoramica sulle funzioni di PHP con qualche funzione sulla gestione di date e ore. Prima di cominciare, però, dobbiamo fare un accenno al **timestamp**, sul quale si basano queste funzioni. Il timestamp è un numero intero, in uso da tempo sui sistemi di tipo UNIX, che rappresenta il numero di secondi trascorsi a partire dal `1° gennaio 1970`. Ad esempio, il timestamp relativo alle 15.56.20 del 24 aprile 2003 è `1051192580`.

Vediamo dunque queste funzioni e rimandiamo a questo articolo (http://php.html.it/articoli/leggi/929/date-in-php-come-gestirle/), in cui questa funzione viene analizzata approfonditamente:

time()
: È la più semplice di tutte, perchè fornisce il timestamp relativo al momento in cui viene eseguita. Restituisce un intero (timestamp).

date(formato [,timestamp])
: Considera il timestamp in input (se non è indicato, prende quello attuale) e fornisce una data formattata secondo le specifiche indicate nel primo parametro. Tali specifiche si basano su una tabella di cui riassumiamo i valori più usati:

Codice	Descrizione
Y	anno su 4 cifre
y	anno su 2 cifre
n	mese numerico (1-12)
m	mese numerico su 2 cifre (01-12)
F	mese testuale ('January' - 'December')
M	mese testuale su 3 lettere ('Jan' - 'Dec')
d	giorno del mese su due cifre (01-31)
j	giorno del mese (1-31)

w giorno della settimana, numerico (0=dom, 6=sab)

l giorno della settimana, testuale ('Sunday' - 'Saturday')

D giorno della settimana su 3 lettere ('Sun' - 'Sat')

H ora su due cifre (00-23)

G ora (0-23)

i minuti su due cifre (00-59)

s secondi su due cifre (00-59)

La funzione `date()` restituisce la stringa formattata che rappresenta la data.

mktime(ore, minuti, secondi, mese, giorno, anno)

È una funzione molto utile ma che va maneggiata con molta cura, perchè i parametri che richiede in input (tutti numeri interi) hanno un ordine abbastanza particolare, che può portare facilmente ad errori. Sulla base di questi parametri, `mktime()` calcola il timestamp, ma l'aspetto più interessante è che possiamo utilizzarla per fare calcoli sulle date. Infatti, se ad esempio nel parametro mese passiamo `14`, PHP lo interpreterà come `12+2`, cioè "febbraio dell'anno successivo", e quindi considererà il mese come febbraio ed aumenterà l'anno di 1. Ovviamente lo stesso tipo di calcoli si può fare su tutti gli altri parametri. Restituisce un intero (timestamp).

checkdate(mese, giorno, anno)

Verifica se i valori passati costituiscono una data valida. Restituisce un valore booleano.

Vediamo quindi qualche esempio anche per queste funzioni:

```
// $a riceve il timestamp del 24/4/2003 alle 15.56.20
$a = mktime(15,56,20,4,24,2003);
// $b sarà "24 Apr 03 - 15:56"
$b = date('d M y - H:i', $a);
// timestamp delle ore 14 di 60 giorni dopo il 24/4/2003
$a = mktime(14,0,0,4,24+60,2003);

$c = checkdate(5,1,2003);  // vero
$c = checkdate(19,7,2003); // falso (19 non è un mese)
$c = checkdate(4,31,2003); // falso (31 aprile non esiste)
```

Scrivere funzioni personalizzate

Come abbiamo detto nella lezione precedente, oltre alle numerosissime funzioni incorporate in PHP abbiamo la possibilità di **definire delle funzioni** che ci permettono di svolgere determinati compiti in diverse parti del nostro script, o, meglio ancora, in script diversi, semplicemente richiamando la porzione di codice relativa, alla quale avremo attribuito un nome che identifichi la funzione stessa. Vediamo quindi ora come definire una funzione, fermo restando che, al momento di eseguirla, la chiamata si svolge con le stesse modalità con cui vengono chiamate le funzioni incorporate del linguaggio.

Immaginiamo, facendo il solito esempio banale, di voler costruire una funzione che, dati tre numeri, ci restituisca il maggiore dei tre. Vediamo il codice relativo:

```
function il_maggiore($num1, $num2, $num3)
{
  if (! is_numeric($num1)) { return false; }
```

```
  if (! is_numeric($num2)) { return false; }
  if (! is_numeric($num3)) { return false; }

  if ($num1 > $num2)
  {
    if ($num1 > $num3)
    {
      return $num1;
    } else {
      return $num3;
    }
  }
  else
  {
    if ($num2 > $num3) {
      return $num2;
    } else {
      return $num3;
    }
  }
}
```

Come vediamo, la definizione della funzione avviene attraverso la **parola chiave** `function`, seguita dal nome che abbiamo individuato per la funzione, e dalle parentesi che contengono i parametri (o argomenti) che devono essere passati alla funzione. Di seguito, contenuto fra parentesi graffe, ci sarà il codice che viene eseguito ogni volta che la funzione viene richiamata. Il nome della funzione deve essere necessariamente univoco, questo significa che non è possibile definire due funzioni aventi lo stesso nome.

All'interno della funzione vediamo l'istruzione `return`; questa istruzione è molto importante, perchè **termina la funzione** (cioè restituisce il controllo allo script nel punto in cui la funzione è stata chiamata) e contemporaneamente determina anche il valore restituito dalla funzione. Nel nostro esempio, i tre dati ricevuti in input vengono controllati, uno dopo l'altro, per verificare che siano numerici: in caso negativo (il test infatti viene fatto facendo precedere la funzione `is_numeric()` dal simbolo `"!"` di negazione), la funzione termina immediatamente, restituendo il valore booleano `false`.

Una volta verificato che i tre valori sono numerici, vengono posti a confronto i primi due, e poi quello dei due che risulta maggiore viene posto a confronto col terzo, per ottenere così il maggiore dei tre, che viene infine restituito come risultato della funzione. Quando sarà il momento di eseguire questa funzione potremo quindi usare questo codice:

```
$a = 9;
$b = 8;
$c = 15;
$m = il_maggiore($a, $b, $c); // $m diventa 15
```

Avete visto che abbiamo chiamato la funzione **usando dei nomi di variabile diversi** da quelli usati nella funzione stessa: la funzione infatti usa `$num1`, `$num2`, `$num3`, mentre lo script che la richiama utilizza `$a`, `$b`, `$c`. È molto importante ricordare che non c'è nessuna relazione definita tra i nomi degli argomenti che la funzione utilizza e quelli che vengono indicati nella chiamata. Ciò che determina la corrispondenza fra gli argomenti è, infatti, semplicemente la posizione in cui vengono indicati: nel nostro caso, quindi, `$a` diventerà `$num1` all'interno della funzione, `$b` diventerà `$num2` e `$c` diventerà `$num3`.

Avremmo potuto richiamare la nostra funzione anche passando direttamente i valori interessati,

senza utilizzare le variabili:

```
$m = il_maggiore(9, 8, 15); // $m diventa 15

/* $m diventa FALSE, perchè il terzo argomento non è numerico e quindi
 * i controlli all'inizio della funzione bloccano l'esecuzione
 */
$m = il_maggiore(9, 8, 'ciao');
```

Scope e argomenti facoltativi

Lo scope delle variabili

Passiamo ora ad un altro argomento molto importante, che è l'ambito (scope) delle variabili. Infatti, le variabili utilizzate in una funzione esistono solo all'interno della funzione stessa, mentre non sono definite nè per le altre funzioni nè nello script principale. Allo stesso modo, le variabili usate dallo script principale non vengono viste dalle funzioni. Questa è una caratteristica molto importante del linguaggio, perchè ci permette di definire le funzioni con la massima libertà nel dare i nomi alle variabili al loro interno, senza doverci preoccupare che nel nostro script (o in qualche altra funzione) ci siano variabili con lo stesso nome il cui valore potrebbe risultare alterato. Questo significa, per tornare all'esempio precedente, che se avessimo scritto `print $num2` all'esterno della funzione `il_maggiore()`, non avremmo ottenuto alcun risultato, e anzi avremmo ricevuto un errore di tipo notice, in quanto la variabile `$num2`, in quell'ambito, non è definita.

Le variabili utilizzate all'interno di una funzione si chiamano variabili locali. Le variabili utilizzate dallo script principale, invece, sono dette variabili globali.

Normalmente, quindi, se una funzione ha bisogno di un certo dato, è sufficiente includerlo tra i parametri che le dovranno essere passati. Tuttavia, esiste una possibilità per consentire ad una funzione di vedere una variabile globale: si tratta di dichiararla attraverso l'istruzione global.

```
function stampa($var1, $var2) {
global $a;
print $a;
}
$a = 'ciao a tutti';
$b = 'buongiorno';
$c = 'arrivederci';
stampa($b, $c);
```

Questo codice, dopo avere definito la funzione `stampa()`, assegna un valore a tre variabili globali: `$a`, `$b` e `$c`. Viene poi chiamata la funzione `stampa()`, passandole i valori di `$b` e `$c`, che nella funzione si chiamano `$var1` e `$var2` ma che al suo interno non vengono utilizzati. Viene invece dichiarata la variabile globale $a, che è valorizzata con la stringa 'ciao a tutti', e quindi questo è ciò che viene stampato a video dall'istruzione print. Se non ci fosse l'istruzione "global $a", la variabile `$a` risulterebbe non definita all'interno della funzione, e quindi la print genererebbe un errore.

Noi comunque **sconsigliamo** di utilizzare le variabili globali in questo modo, perchè fanno perdere chiarezza allo script e alla funzione stessa.

Termine della funzione e valore restituito. Abbiamo visto in precedenza che la funzione termina con l'istruzione return, la quale può anche essere utilizzata per restituire un valore. Nel caso in cui non sia scopo della funzione quello di restituire un valore, utilizzeremo semplicemente **return**.

Viceversa, nel caso in cui volessimo restituire più di un valore, siccome la sintassi di PHP ci consente di restituire una sola variabile, potremo utilizzare un **array**. Vediamo un esempio, immaginando una funzione il cui scopo sia quello di ricevere un numero e di restituire il suo doppio, il suo triplo e il suo quintuplo:

```
function multipli($num) {
$doppio = $num * 2;
$triplo = $num * 3;
$quintuplo = $num * 5;
$ris = array($doppio, $triplo, $quintuplo);
return $ris;
}
```

Con questo sistema siamo riusciti a restituire tre valori da una funzione, pur rispettando la sintassi che ne prevede uno solo. Ovviamente quando richiameremo la funzione dovremo sapere che il risultato riceverà un array:

$a = 7;
$mul = multipli($a); // $mul sarà un array a 3 elementi

Se ci interessa, possiamo anche usare un costrutto del linguaggio per distribuire immediatamente i tre valori su tre variabili distinte:

list($doppio, $triplo, $quintuplo) = multipli($a);

Il costrutto `list()` serve ad assegnare i valori di un array (quello indicato a destra dell'uguale) ad una serie di variabili che gli passiamo fra parentesi. Ovviamente è possibile utilizzarlo non solo per "raccogliere" il risultato di una funzione, ma con qualsiasi array:

$arr = array('Marco','Paolo','Luca');
list($primo, $secondo, $terzo) = $arr;

In questo caso, `$primo` prenderà il valore 'Marco', `$secondo` il valore 'Paolo', $terzo il valore 'Luca'.

Un'ulteriore precisazione: poco fa abbiamo detto che la funzione termina con l'istruzione `return`. In realtà ciò non è necessario: infatti, nel caso in cui non ci siano valori da restituire, possiamo anche omettere l'istruzione `return`, e l'esecuzione della funzione terminerà quando arriverà in fondo al codice relativo, restituendo il controllo allo script principale (o ad un'altra funzione che eventualmente l'ha chiamata).

Argomenti facoltativi

In determinate situazioni possiamo prevedere delle funzioni in cui non è obbligatorio che tutti gli argomenti previsti vengano passati al momento della chiamata. Abbiamo visto, infatti, nella lezione precedente, che alcune funzioni di PHP prevedono dei parametri facoltativi. La stessa cosa vale per le funzioni definite da noi: se infatti, al momento in cui definiamo le funzioni, prevediamo un valore di default per un certo parametro, quel parametro diventerà facoltativo in fase di chiamata della funzione, e, nel caso manchi, la funzione utilizzerà il valore di default.

Come esempio consideriamo una funzione che stampa i dati anagrafici di una persona:

```
function anagrafe($nome, $indirizzo, $cf='non disponibile') {
  print "Nome: $nome<br />";
  print "Indirizzo: $indirizzo<br />";
  print "Codice fiscale: $cf<br />";
}
```

Questa funzione prevede tre parametri in input, ma per il terzo è previsto un valore di default (la stringa 'non disponibile'). Quindi, se la funzione viene chiamata con soltanto due argomenti, la variabile `$cf` avrà proprio quel valore; se invece tutti e tre gli argomenti vengono indicati, il valore di default viene ignorato. Vediamo due esempi di chiamata di questa funzione:

```
anagrafe('Mario Rossi', 'via Roma 2', 'RSSMRA69S12A944X');
anagrafe('Paolo Verdi', 'via Parigi 9');
```

Nel primo caso otterremo questo output a video:

```
Nome: Mario Rossi
Indirizzo: via Roma 2
Codice fiscale: RSSMRA69S12A944X
```

Nel secondo caso:

```
Nome: Paolo Verdi
Indirizzo: via Parigi 9
Codice fiscale: non disponibile
```

Nella seconda occasione il codice fiscale non è stato passato, e la funzione ha utilizzato il valore di default.

Le variabili GET e POST

La principale peculiarità del web dinamico, come abbiamo detto all'inizio di questa guida, è la possibilità di variare i contenuti delle pagine in base alle richieste degli utenti. Questa possibilità si materializza attraverso i meccanismi che permettono agli utenti, oltre che di richiedere una pagina ad un web server, anche di specificare determinati parametri che saranno utilizzati dallo script PHP per determinare quali contenuti la pagina dovrà mostrare. Come esempio, possiamo immaginare una pagina il cui scopo è quello di visualizzare le caratteristiche di un dato prodotto, prelevandole da un database nel quale sono conservati i dati di un intero catalogo. Nel momento in cui si richiama la pagina, si dovrà specificare il codice del prodotto che deve essere visualizzato, per consentire allo script di prelevare dal database i dati di quel prodotto e mostrarli all'utente.

In alcuni casi, **i dati che devono essere trasmessi allo script sono piuttosto numerosi**: pensiamo ad esempio ad un modulo di registrazione per utenti, nel quale vengono indicati nome, cognome, indirizzo, telefono, casella e-mail ed altri dati personali. In questo caso lo script, dopo averli ricevuti, andrà a salvarli nel database.

In questa lezione non ci occuperemo di come vengono salvati o recuperati i dati da un database, ma del modo in cui PHP li riceve dall'utente. Esistono due sistemi per passare dati ad uno script: il metodo GET e il metodo POST.

Il metodo GET

Il metodo GET consiste nell'accodare i dati all'indirizzo della pagina richiesta, facendo seguire il nome della pagina da un punto interrogativo e dalle coppie nome/valore dei dati che ci interessano. Nome e valore sono separati da un segno di uguale. Le diverse coppie nome/valore sono separate dal segno '&'. Quindi, immaginando di avere la pagina `prodotto.php` che mostra le caratteristiche di un prodotto passandole il codice e la categoria del prodotto stesso, diciamo che, per visualizzare i dati del prodotto A7 della categoria 2, dovremo richiamare la pagina in questo modo:

<a href="prodotto.php?cod=a7&cat=2">

La stringa che si trova dopo il punto interrogativo, contenente nomi e valori dei parametri, viene detta query string. Quando la pagina prodotto.php viene richiamata in questo modo, essa avrà a disposizione, al suo interno, le variabili `$_GET['cod']` (con valore 'a7') e `$_GET['cat']` (con valore '2'). Infatti i valori contenuti nella query string vengono memorizzati da PHP nell'array $_GET, che è un array superglobale in quanto è disponibile anche all'interno delle funzioni.

Quindi, per tornare all'esempio del catalogo, possiamo immaginare di avere una pagina nella quale mostriamo una tabella con il nome di ogni prodotto su una riga, e, di fianco, il link che ci permette di visualizzare le caratteristiche di quel prodotto. In ogni riga, quindi, questo link richiamerà sempre la pagina `prodotto.php`, valorizzando ogni volta i diversi valori di 'cod' e 'cat'.

Il metodo POST

Il metodo POST viene utilizzato con i moduli: quando una pagina HTML contiene un tag `<form>`, uno dei suoi attributi è `method`, che può valere GET o POST. Se il metodo è GET, i dati vengono passati nella query string, come abbiamo visto prima. Se il metodo è POST, i dati vengono invece inviati in maniera da non essere direttamente visibili per l'utente, attraverso la richiesta HTTP che il browser invia al server.

I dati che vengono passati attraverso il metodo POST sono memorizzati nell'array $_POST. Anche questo array, come $_GET, è un array superglobale. Quindi, per fare un esempio attraverso un piccolo modulo:

```
<form action="elabora.php" method="post">
   <input type="text" name="nome">
   <input type="checkbox" name="nuovo" value="si">
   <input type="submit" name="submit" value="invia">
</form>
```

Questo modulo contiene semplicemente una casella di testo che si chiama 'nome' e una checkbox che si chiama 'nuovo', il cui valore è definito come 'sì'. Poi c'è il tasto che invia i dati, attraverso il metodo POST, alla pagina `elabora.php`.

Questa pagina si troverà a disposizione la variabile `$_POST['nome']`, contenente il valore che l'utente ha digitato nel campo di testo; inoltre, se è stata selezionata la checkbox, riceverà la variabile `$_POST['nuovo']` con valore 'si'. Attenzione però: se la checkbox non viene selezionata dall'utente, la variabile corrispondente risulterà non definita.

Abbiamo visto quindi, brevemente, in che modo recuperare i dati che gli utenti ci possono trasmettere. C'è da dire che, modificando l'impostazione `register_globals` su `php.ini`, sarebbe possibile anche recuperare i dati in maniera più semplice. Infatti, se `register_globals` è attiva ('on'), oltre agli array visti sopra avremo anche delle variabili globali che contengono direttamente i valori corrispondenti. Ad esempio, nel primo esempio avremmo a disposizione la variabile `$cod` e la variabile `$cat`, nel secondo avremmo la variabile `$nome` e la variabile (eventuale) `$nuovo`. Fino a qualche tempo fa, erano in molti a lavorare in questo modo, perchè il valore di `register_globals`, di default, era attivo, e quindi buona parte dei programmatori PHP, soprattutto agli inizi, trovavano più naturale utilizzare il sistema più immediato.

A partire dalla versione 4.2.0 di PHP, fortunatamente, il valore di default di register_globals è stato cambiato in off, e gli sviluppatori di PHP sconsigliano di rimetterlo ad on per gravi problemi di sicurezza. Questo perchè, utilizzando gli array superglobali `$_GET` e `$_POST`, si rende il codice più chiaro e anche più sicuro. Se vi dovesse capitare di utilizzare degli script già fatti e doveste

notare dei malfunzionamenti, potrebbe dipendere dal fatto che tali script utilizzano le variabili globali invece degli array superglobali, ed il vostro `register_globals` è a off.

Un'ultima annotazione: gli array `$_GET` e `$_POST` sono stati introdotti nella versione 4.1.0 di PHP. In precedenza, gli stessi valori venivano memorizzati negli array corrispondenti `$HTTP_GET_VARS` e `$HTTP_POST_VARS`, che però non erano superglobali. Questi array sono disponibili anche nelle versioni attuali di PHP, ma il loro uso è sconsigliato, ed è presumibile che in futuro scompariranno.

Le variabili superglobali definite automaticamente da PHP non permettono solamente di accedere ai parametri passati alla pagina, ma esistono variabili che permettono di accedere ad altri valori molto importanti.

La variabile `$_SERVER` contiene informazioni sul server corrente, recuperate attraverso Apache o estratte dagli header della pagina valorizzati dal browser dell'utente che sta navigando. Spesso i valori contenuti in questo array associativo vengono utilizzati per comprendere da dove proviene una richiesta, rilevare l'indirizzo IP che identifica il PC da cui l'utente è acceduto alla nostra pagina oppure conoscere il path di base dell'applicativo.

Le variabili `$_COOKIE` e `$_SESSION` (che verranno analizzate successivamente in modo più approfondito) contengono rispettivamente i valori dei cookie e delle sessioni valide per una determinata pagina. I valori vengono acceduti utilizzando come chiave il nome del cookie/sessione che si decide interrogare.

La variabile `$_FILES` è molto importante perchè contiene informazioni su tutti i file inviati alla pagina attraverso un form. L'array `$_FILES` ha una struttura che permette di recuperare (sempre attraverso il nome del parametro, come per le variabili `$_POST` e `$_GET`) il nome del file caricato, le sue dimensioni ed il nome del file temporaneo salvato su disco sul quale è possibile operare.

Mantenere lo stato: i cookie

Un cookie è una coppia chiave/valore avente una data di scadenza ed un dominio di validità che viene salvata sul PC dell'utente ed inviata (attraverso appositi header HTTP) ad ogni pagina dalla quale possa essere acceduta.

Grazie ai cookie è possibile identificare con buona sicurezza le credenziali di un utente che accede ad una pagina, oppure salvare dei dati per un successivo recupero. Per esempio si potrebbe salvare su un cookie un valore indicante l'ultima pagina visualizzata, in modo da occuparsi di ridirigere l'utente all'ultima pagina visualizzata nel momento in cui si connettesse nuovamente al nostro sito.

La creazione di un cookie è un'operazione molto semplice, che in PHP può essere effettuata utilizzando un'unica chiamata alla funzione `setcookie()`. Questa funzione accetta un numero variabile di parametri. Nell'ordine:

- Il nome del cookie
- Il valore del cookie, che dovrà essere necessariamente un valore scalare (intero o stringa, gli array non possono essere salvati direttamente)
- Un numero indicante la data di scadenza del cookie. Nel caso questo numero sia 0 o non specificato, il cookie durerà fino a che l'utente non chiuderà il suo browser. Nel caso in cui il timestamp specificato risulti in una data precedente a quella attuale, il cookie viene cancellato
- Il path di validità del cookie
- Il dominio di validità del cookie

- Un parametro boolean che indica se trasmettere il cookie solamente attraverso una connessione sicura HTTPS

I cookie creati sono disponibili solamente dalla pagina successiva alla loro creazione.

Dato che la funzione `setcookie()` genera esplicitamente un header HTTP, è necessario che prima del suo utilizzo non sia stato stampato (usando `echo`, `print` o qualunque altro metodo di output) alcun valore, altrimenti verrà generato un errore. Anche una riga vuota all'inizio del file prima del tag di apertura PHP porterà alla generazione di questo errore.

Alcuni esempio di creazione di un cookie:

setcookie('prova_cookie', 'valore cookie', /* dura per un'ora */ time() + 3600);
setcookie('prova_2', 'ciao'); //cookie che dura fino a che l'utente non chiude il browser

Per cancellare un cookie è necessario utilizzare la stessa funzione specificando gli stessi parametri utilizzati in fase di creazione ma utilizzando una data di scadenza precedente a quella attuale.

setcookie('prova_cookie', '', time() - 3600);

Una volta creato un cookie il suo valore sarà accessibile attraverso `$_COOKIE[$nome_cookie]` nelle pagine successive a quella attuale, presupponendo che la data di scadenza non sia trascorsa e che siano rispettate le restrizioni di dominio e cartella.

È buona norma non creare troppi cookie, dato che i browser hanno un limite sia relativo ad uno specifico dominio che ad una specifica cartella. In caso fosse necessario mantenere molti valori, è preferibile salvarli su database o file e salvare nel cookie una chiave che ne permetta l'accesso nella pagine successive. I cookie sono facilmente recuperabili e leggibili, quindi è importante non salvare mai informazioni private o vitali, salvo previa criptazione.

Mantenere lo stato: le sessioni

Le **sessioni** permettono anch'esse di mantenere informazioni relative all'utente tra le varie pagine navigate, ma rispetto ai cookie danno la possibilità di salvare arbitrari tipi di dato (non solo stringhe e numeri) e, se PHP è impostato correttamente, di non preoccuparsi dell'identificazione dell'utente.

Quando un utente accede per la prima volta ad una pagina PHP impostata per creare una sessione, a questi viene associato un ID univoco che verrà utilizzato da PHP come chiave per recuperare l'array `$_SESSION` salvato specificamente per il determinato utente. L'ID è un codice univoco che, al momento della creazione della sessione, viene salvato automaticamente da PHP all'interno di un cookie oppure, se i cookie risultano disabilitati, accodato agli URL relativi generati dallo script PHP.

In questo modo il programmatore può occuparsi solamente di inizializzare la sessione e salvare i propri dati. Ovviamente questo comportamento può essere controllato attraverso le impostazioni del file `php.ini`; potrebbe capitare quindi che il passaggio automatico dell'ID di sessione attraverso gli URL relativi non sia abilitato. In questo caso PHP ci viene incontro generando una costante chiamata `SID` che contiene la chiave di sessione preceduta dal nome. Basta accodare questa costante ai nostri URL per avere un sistema perfettamente funzionante.

L'inizializzazione di una sessione, come già accennato, viene fatta automaticamente da PHP. A noi basterà scrivere:

$_SESSION['nome'] = $valore;

per assicurarci che la variabile di sessione 'nome' abbia il valore assegnato, e che questo valore sia

specifico per l'utente che ha appena eseguito la pagina che contiene il codice.

Non mi addentro nel dettaglio, e vi lascio alla lettura degli articoli di HTML.it, dedicati sia alle sessioni (http://php.html.it/articoli/leggi/875/sessioni-alternative-in-php/) sia ai cookie (http://php.html.it/articoli/leggi/850/tutto-sui-cookie/), che trattano l'argomento in modo molto approfondito.

Le sessioni sono un argomento a prima vista molto semplice, ma spesso ci si trova in situazioni nelle quali è necessario controllare completamente il comportamento dei processi di salvataggio, lettura, creazione e distruzione delle sessioni. A questo scopo ci vengono in aiuto una serie di configurazioni del file `php.ini` ed una serie di funzioni aggiuntive che, anche se non tratterò in questa guida base, consiglio caldamente di studiare ed analizzare.

Accedere ai file

Uno degli aspetti fondamentali della programmazione è quello di poter **accedere a fonti di dato esterne** al fine di recuperare o salvare delle informazioni utili ai fini della nostra applicazione. PHP, come tutti i linguaggi di programmazione, fornisce una ricca libreria per l'accesso ai file ed alle risorse presenti sul server. Con l'avvento di PHP 5, è stato aggiunto anche il supporto a particolari tipi di risorse, chiamate stream, in modo che sia possibile accedere a qualunque tipo di fonte come se si stesse accedendo ad un file (un po' come accade in Linux, dove la maggior parte dei sistemi di input/output hardware possono essere letti e scritti come se fossero file).

Le operazioni principali sui file sono essenzialmente quelle di **lettura, scrittura e posizionamento** (PHP fornisce moltissime operazioni ausiliarie per gestire completamente i file, come operazioni per l'eliminazione e la prova d'esistenza, che analizzeremo in modo più approfondito in seguito). La maggior parte delle operazioni effettuate sui file avvengono applicando delle funzioni che accettano come parametro una risorsa che rappresenta il file. La risorsa (recuperata con fopen, come vedremo in seguito) è un tipo di dato speciale gestito internamente da PHP, che serve al motore del linguaggio di programmazione come indicativo per delle informazioni a basso livello richieste per il corretto funzionamento della libreria utilizzata.

Vediamo un piccolo esempio:

```
<?php

//Presuppongo che la directory corrente abbia i permessi corretti

$fp = fopen('prova.txt', 'w+'); //Apro il file prova.txt in lettura, lo creo se non esiste
fwrite($fp, ciao a tutti, come va?); //Scrivo una stringa sul file
fclose($fp); //Chiudo il file aperto precedentemente

?>
```

Il programma precedente è molto semplice: crea un file `prova.txt`, vi scrive dentro "ciao a tutti, come va?" e lo chiude. L'operazione di apertura di un file avviene attraverso la funzione `fopen` che accetta, nella sua forma base, due parametri: il primo rappresenta il percorso (path) del file sul quale vorremmo operare. Il secondo è una stringa che indica alla funzione quali sono le operazioni che si desiderano svolgere sul file. Per una lista dettagliata di queste rimando alla documentazione ufficiale, ma le più importanti sono:

- **'r'** Apre in sola lettura. Posiziona il puntatore all'inizio del file.
- **'r+'** Apre in lettura e scrittura. Posiziona il puntatore all'inizio del file.
- **'w'** Apre il file in sola scrittura. Posiziona il puntatore all'inizio del file e tronca il file alla lunghezza zero. Se il file non esiste, tenta di crearlo.
- **'w+'** Apre in lettura e scrittura. Posiziona il puntatore all'inizio del file e tronca il file alla lunghezza zero. Se il file non esiste, tenta di crearlo.
- **'a'** Apre in sola scrittura. Posiziona il puntatore alla fine del file. Se il file non esiste, tenta di crearlo.
- **'a+'** Apre in lettura e scrittura. Posiziona il puntatore alla fine del file. Se il file non esiste, tenta di crearlo.

Come è possibile notare dalla descrizione dei parametri elencati precedentemente, si fa riferimento al puntatore di un file. Il puntatore non è altro che un indicatore numerico che specifica la posizione attuale all'interno del file dalla quale verranno eseguite le operazioni richieste. Il posizionamento del puntatore avviene tramite la funzione `fseek`, che accetta come parametri la risorsa del file, un numero di byte, ed una costante che indica se il numero di byte è assoluto (SEEK_SET), se deve essere aggiunto alla posizione corrente (SEEK_CUR) oppure se deve essere aggiunto alla fine del file (SEEK_END).

Un semplice esempio per chiarire:

```
<?php

// Apro il file prova.txt in scrittura e lo riempio con 10 righe di testo
$fp = fopen(prova.txt, w+);
for($i = 0; $i < 10; ++$i)
{
  fwrite($fp, Stringa di prova numero .$i.\n);
}
fclose($fp);

// Ora apro il file in lettura, mi muovo al suo interno, e stampo parti di contenuto
$fp = fopen(prova.txt, r);
fseek($fp, 10, SEEK_SET); //Mi posiziono al 10° carattere
$prova = fread($fp, 20); //Leggo 20 caratteri partendo dalla posizione corrente
echo $prova;
echo "<br />";
echo "La posizione del puntatore all'interno del file è: ".ftell($fp);
fclose($fp);

?>
```

Nel codice precedente ho aperto in scrittura `prova.txt` e l'ho riempito un file con 10 righe di testo. Poi, dopo averlo chiuso, l'ho nuovamente aperto in lettura, ho spostato il puntatore di 10 posizioni, ho letto 20 caratteri con la funzione fread ed ho stampato la stringa letta seguita dalla nuova posizione del puntatore (10 + 20 = 30). La funzione ftell accetta solamente la risorsa rappresentante il file e restituisce la posizione del puntatore al suo interno, mentre la funzione fread accetta come secondo parametro il numero di byte da leggere. Dopo la lettura il puntatore verrà spostato del numero di byte specificato. Giocate un po' con queste funzioni per capirne il corretto funzionamento.

Utilizzare SQLite

Per molti anni PHP è stato affiancato al **database MySQL**. Possiamo quasi dire che la loro crescita è stata parallela ed ha portato grossi miglioramenti nel campo delle applicazioni web opensource. Per molto tempo quindi con PHP è stato fornito nativamente il supporto alle librerie per l'accesso a MySQL. Purtroppo il cambio di licenza di quest'ultimo ha obbligato gli sviluppatori a rimuovere il supporto nativo per MySQL (anche se la libreria è sempre in evoluzione e distribuita).

Al fine di aiutare comunque coloro che necessitano di un database ma hanno solamente accesso alla configurazione minima di PHP, gli sviluppatori hanno deciso di implementare nativamente il supporto ad **un altro sistema di database**: SQLite. Questo sistema, cui abbiamo dedicato un ampio approfondimento (http://database.html.it/articoli/leggi/895/sqlite-miniguida-alluso/), si differenzia molto da MySQL dato che non si basa su un'architettura client server ed è stato sviluppato appositamente per permettere l'accesso molto veloce ai dati. I database creati sono contenuti in un unico file binario (possono anche essere creati database temporanei salvati in memoria) che può essere acceduto tramite le funzioni fornite dalla libreria per eseguirvi query SQL.

La libreria SQLite fornisce **un'interfaccia sia ad oggetti sia procedurale**. Dato che il compito di questa guida introduttiva non comprende la trattazione delle programmazione ad oggetti, ci occuperemo dell'interfaccia a funzioni. Per chi fosse interessato a MySQL rimandiamo alla guida

Per operare su un database è necessario recuperare una risorsa (un po' come abbiamo visto precedentemente per i file) e lavorare su questa con delle funzioni apposite. La risorsa connessa ad un database può essere recuperata utilizzando sqlite_open:

$sq = sqlite_open("miodb.db", 0666, $sqlite_error);

Il primo parametro è il nome del file che conterrà i nostri dati (se non esiste verrà creato), il secondo indica i permessi da associare al database (attualmente il parametro viene ignorato da SQLite anche se l'impostazione consigliata è 0666) mentre il terzo conterrà eventualmente una stringa con il messaggio di errore eventualmente riscontrato durante l'apertura della fonte di dati. In caso sia andato tutto per il verso giusto, `$sq` conterrà la risorsa che ci permetterà di accedere al database `miodb.db`, altrimenti assumerà un valore nullo. Per questo motivo è sempre buona prassi controllare il valore restituito da `sqlite_open` prima di proseguire.

Per effettuare una query sul database possiamo utilizzare la funzione `sqlite_query`, ed analizzare il risultato eventualmente ottenuto (per esempio in caso di operazioni di selezione) attraverso `sqlite_fetch_array`. Un semplice esempio:

```
$sq = sqlite_open("miodb.db", 0666, $sqlite_error);
if(!$sq)
{
   die("Errore Sqlite: ".$sqlite_error);
}

sqlite_query($sq, "CREATE TABLE prova_tbl (campo varchar(10))");
for($i = 0; $i < 10; ++$i)

{
   sqlite_query($sq, "INSERT INTO prova_tbl VALUES ('Prova ".$i."')");
}

$result = sqlite_query($sq, "SELECT * FROM prova_tbl");
```

```
while($data = sqlite_fetch_array($result))

{
   echo $data['campo']."<br />";
}
sqlite_close($sq);
```

Nel codice precedente ci siamo connessi al database, abbiamo controllato che non ci fossero errori ed abbiamo eseguito delle query sulla risorsa recuperata. La prima query ha creato una tabella di nome `prova_tbl` con un campo di nome "campo"; la seconda, eseguita all'interno di un ciclo, si è occupata di inserire dieci valori nella tabella mentre la terza ha recuperato tutti questi valori. All'interno del ciclo while abbiamo recuperato una dopo l'altra le singole righe selezionate ed abbiamo stampato i valori del campo "campo". Come possiamo notare la funzione sqlite_fetch_array restituisce la prossima riga selezionata oppure FALSE nel caso in cui quella precedente fosse l'ultima.

Come è stato possibile notare da questa breve introduzione, SQLite si comporta in modo molto simile ad un database relazionale, con la differenza che non opera in un'architettura client / server e permette l'esecuzione di query molto semplici e compatte.

Una funzionalità molto interessante della libreria, che mi sento di dover trattare prima di chiudere, è quella che permette di registrare delle funzioni PHP da richiamare all'interno delle proprie query SQL. Vediamo un semplice esempio:

```
function trim_upper($string)
{
  return strtoupper(trim($string));
}

$sq = sqlite_open("miodb.db", 0666, $sqlite_error);
if(!$sq)
{
  die("Errore Sqlite: ".$sqlite_error);
}

sqlite_create_function($sq, "trimup", "trim_upper", 1);
$result = sqlite_query($sq, "SELECT trimup(campo) AS campo FROM prova_tbl");
while($data = sqlite_fetch_array($result))
{
  echo $data['campo']."<br />";
}
sqlite_close($sq);
```

Il codice accede al database che abbiamo creato precedentemente e recupera tutte le righe applicando direttamente da SQL la funzione `trim_upper` (che elimina gli spazi all'inizio ed alla fine e rende la stringa maiuscola) al campo selezionato. La funzione viene registrata attraverso `sqlite_create_function` che accetta come parametri la risorsa rappresentante il database, il nome da utilizzare all'interno dell'SQL per richiamare la funzione passata come terzo argomento ed infine il numero di parametri accettati. Grazie a questa interessante funzionalità si può estendere il linguaggio SQL utilizzato da SQLite con un insieme di funzioni adatte a compiere le operazioni più ripetitive sui dati, al fine di rendere il codice più ordinato e pulito.

Interrogare database MySQL

In PHP 5 abbiamo diverse soluzioni per connetterci ai database. Uno dei database più utilizzati in ambito opensource è sicuramente **MySQL**, che conta dalla sua una larga schiera di sviluppatori e supporters. In PHP 5 possiamo accedere a MySQL attraverso **i layer di astrazione** distribuiti nella release standard (PDO ed SDO), ma anche utilizzando **la libreria mysql** (quella utilizzata anche nella versione precedente di PHP) e **la nuova libreria mysqli** che fornisce un supporto più completo al linguaggio ed espone un'interfaccia ad oggetti.

Date lo modifiche apportate al linguaggio che puntano a muovere il paradigma di programmazione tipico di PHP da strutturato ad oggetti, mi pare una buona scelta imparare a conoscere **la libreria mysqli** utilizzando la sua interfaccia a classi piuttosto che basarsi sull'approccio a funzioni.

La connessione ad un database mysql prevede la creazione di un oggetto mysqli tramite il quale effettueremo le nostre operazioni di interrogazione e gestione del database:

```
<?php
$mysqli = new mysqli('host', 'username', 'password', 'dbname');
// ... eseguiamo le nostre operazioni ...
$mysqli->close();
?>
```

Una volta istanziato un oggetto mysqli possiamo operare su di esso:

```
<?php
// ... connessione

$mysqli->autocommit(true);
$mysqli->query("
CREATE TABLE test (
id INT UNSIGNED AUTO_INCREMENT NOT NULL,
title VARCHAR(32) NOT NULL,
content TEXT NOT NULL,
PRIMARY KEY(id)
);
");

// Inseriamo qualche informazione

for($i = 0; $i < 1000; ++$i)

{
$query = sprintf("INSERT INTO test (title, content) VALUES ('%s', '%s')", "Titolo ".$i, "Contenuto di prova ".$i);
$mysqli->query($query);
}


// Selezioniamo e stampiamo le righe inserite

$result = $mysqli->query("SELECT * FROM test", MYSQLI_USE_RESULT);
```

```
while($row = $result->fetch_assoc())

{
printf("<h3>%s</h3><p>%s</p><hr />", $row['title'], $row['content']);
}

$result->close();

// ....

?>
```

I metodi utilizzati nell'esempio precedente sono i seguenti:

- **void autocommit(bool)**: permette di impostare l'oggetto in modo che richiami automaticamente il metodo commit() dopo aver effettuato una query. In caso sia impostato a false e si stia operando su tabelle che supportano le transizioni, è necessario richiamare commit manualmente per applicare le modifiche apportate dalle query;
- **mixed query(string[, int])**: esegue una query SQL sul database utilizzato. Il risultato restituito dipende dalla tipologia di query eseguita: nel caso la query `SELECT, SHOW, EXPLAIN` o `DESCRIBE` viene restituito un oggetto (di cui analizzeremo le proprietà successivamente) altrimenti viene restituito true in caso di query eseguita correttamente, false in caso contrario. Il secondo parametro passato al metodo può essere la costante `MYSQLI_USE_RESULT` o `MYSQLI_STORE_RESULT`: la seconda è quella impostata di default e viene utilizzata per effettuare il buffering dei dati recuperati attraverso la query; invece nel caso si utilizzi la prima costante i dati non sono bufferizzati;

Come accennato le chiamate al metodo query possono restituire un oggetto nel caso in cui la query eseguita preveda un risultato diverso dalle informazioni sulla sua corretta esecuzione. Questo oggetto è un'istanza della classe mysqli_result, ed espone metodi per iterare sui risultati. Vediamo i più interessanti:

- **proprietà num_rows**: restituisce il numero delle righe contenute nel buffer o nel risultato SQL;
- **array fetch_assoc()**: restituisce il successivo risultato sotto forma di un array avente come chiavi i nomi dei campi recuperati e come valori i rispettivi valori. In caso l'iterazione sia terminata viene restituito NULL;
- **array fetch_row()**: opera come fetch_assoc ma utilizza indici numerici per identificare i risultati;
- **array fetch_array()**: restituisce un array che contiene sia indici numerici che stringhe per recuperare i valori;
- **object fetch_field()**: restituisce un oggetto che contiene le informazioni sui campi recuperati dalla query;

Nel caso si utilizzi fetch_field l'oggetto restituito espone le seguenti proprietà:

- **name**: il nome della colonna;
- **orgname**: il nome originale della colonna nel caso sia stato specificato un alias;
- **table**: il nome della tabella a cui appartiene il campo, a meno che questo non sia calcolato (come nel caso di "SELECT (1+1) AS test" per esempio);
- **orgtable**: il nome originale della tabella nel caso in cui sia stato specificato un alias;
  def: il valore di default di questo campo, rappresentato come una stringa;
- **max_length**: la lunghezza massima del campo;

- **flags**: un intero che rappresenta i flag associati al campo;
- **type**: il tipo di dato utilizzato per il campo;
- **decimals**: il numero di decimali utilizzati (solo nel caso di campi numerici);

Prepared statement

Dopo questa introduzione, possiamo passare ad analizzare uno degli aspetti più interessanti della libreria mysqli, i **prepared statement**. Normalmente quando il database esegue una query, effettua prima una compilazione di quest'ultima e poi esegue il codice compilato. Questa operazione viene normalmente effettuata ogni volta che una query viene eseguita, anche nel caso di chiamate successive a query molto simili. I prepared statement permettono di precompilare una query lasciando dei campi variabili: quando la query dovrà essere eseguita potranno essere assegnati solo questi campi e non si dovrà procedere con l'intera compilazione, guadagnando molto in performance.

Vediamo come si prepara ed utilizza un prepared statement:

```
<?php

// ... connessione

$search = '';

$stmt = $mysqli->prepare("SELECT id, title FROM test WHERE title LIKE ?");

$stmt->bind_param('s', $search);

for($i = 0; $i < 5; ++$i)

{
$search = '%'.$i.'%';
$stmt->execute();
$stmt->bind_result($id, $title);
echo "<h3>",$i,"</h3>";
while($stmt->fetch())

{
printf("<strong>%d</strong><span>%s</span><br />", $id, $title);
}
$stmt->free_result();
}

$stmt->close()

// ...

?>
```

Utilizzando il metodo prepare dell'oggetto mysqli possiamo creare un prepared statement, rappresentato da un'istanza della classe mysqli_stmt. La query passata come argomento può contenere una serie di punti di domanda nei posti in cui successivamente inseriremo i valori. I punti di domanda possono essere inseriti solamente nelle posizioni in cui la query SQL si aspetterebbe dei

valori (come quelli degli `INSERT`, gli assegnamenti di `UPDATE` o i valori delle condizioni della clausola `WHERE`). Una volta preparato l'oggetto possiamo operarvi utilizzando i suoi metodi:

- **void bind_param(string, ...)**: associa una o più variabili ai singoli placeholder specificati nella query precompilata. Il primo argomento è una stringa di cui ogni singolo carattere rappresenta il tipo di dato in cui convertire i valori contenuti nelle variabili connesse. I caratteri utilizzabili sono i per gli interi, d per i double, s per le stringhe e b per i blob. I restanti parametri sono i nomi delle variabili che vogliamo connettere.
- **bool execute()**: esegue un prepared statement salvando gli eventuali risultati recuperati in un buffer interno ed estraendo i valori dei placeholder dalle variabili connesse;
- **void bind_result(...)**: associa i singoli campi recuperati dalla query a delle variabili. I valori di queste variabili saranno aggiornati ad ogni chiamata effettuata al metodo fetch;
- **bool fetch()**: passa al record successivo assegnando i valori dei campi recuperati alle variabili connesse. Se non c'è alcun record restituisce false;

La configurazione di PHP

Come molti altri strumenti di sviluppo PHP può essere configurato attraverso un file di configurazione che definisce e guida il comportamento delle zend engine e delle sue estensioni. Le proprietà di configurazione possono essere assegnate e modificate in vari modi, che analizziamo brevemente; va anticipato che alcune proprietà possono essere modificate solamente in alcuni contesti, solitamente per motivi di sicurezza.

La prima soluzione è apportare manualmente le **modifiche al file php.ini** presente nella directory di configurazione di PHP. Ogni volta che si apportano modifiche a questo file è necessario riavviare l'interprete (solitamente riavviando il webserver di supporto) e spesso il file, per motivi di sicurezza, risulta protetto da modifiche da parte degli utenti che usufruiscono di servizi hosting.

Il formato del file di configurazione di PHP è molto semplice e segue gli standard utilizzati da altri strumenti opensource:

- Vengono saltate tutte le righe vuote o precedute da un punto e virgola;
- Le proprietà sono definite da una serie di caratteri senza spazi;
- Ogni riga definisce un'operazione di assegnamento (utilizzando l'operatore uguale);
- I valori possono essere numeri, costanti interne, stringhe o altre espressioni valide interpretate dal motore;

La seconda soluzione possibile, optabile solamente nel caso in cui PHP giri come modulo del webserver Apache, è **sfruttare gli strumenti di configurazione del webserver** stesso per modificare il comportamento di PHP. Attraverso la direttiva `php_flag` possiamo impostare ad On o Off i valori di una variabile di configurazione di PHP; in caso i valori di una variabile siano valori differenti dal booleano è possibile utilizzare php_value:

```
php_value error_reporting E_ALL
php_flag register_globals Off
```

Se il webserver e PHP sono preconfigurati correttamente, le direttive possono essere specificate

sia **in un file .htaccess** che all'interno del file http.conf, magari sfruttando la possibilità di definire comandi in base alla directory.

L'ultima opzione possibile è quella di modificare i valori direttamente all'interno del proprio codice PHP sfruttando la funzione `ini_set` (con `ini_get` possiamo recuperare i valori assegnati o specificati con i parametri di configurazione):

```
<?php
ini_set('include_path', ini_get('include_path').':../includes:');
?>
```

Ovviamente alcune direttive non ha senso siano impostate all'interno del codice, come ad esempio quelle che operano sui dati in ingresso.

Vediamo ora alcuni parametri di configurazione che controllano il comportamento del motore di PHP:

allow_call_time_pass_reference(boolean): Abilita o meno la possibilità di forzare gli argomenti delle funzioni ad essere passati per riferimento. Questo parametro è deprecato e potrebbe non essere più supportato nelle versioni future di PHP/Zend. Si incoraggia il metodo di specificare quale parametro debba essere passato per riferimento al momento della dichiarazione della funzione. Si suggerisce di impostare l'opzione a off per essere certi che lo script funzioni correttamente con questa impostazione, in modo da predisporsi ad eventuali modifiche future del linguaggio (si riceverà un warning ogni volta che si utilizza questa opzione e i valori saranno passati per valore anzichè per riferimento). Passare i valori per riferimento al momento della chiamata della funzione viene sconsigliato per motivi di chiarezza del codice. La funzione può modificare il parametro in modo non previsto se non indica questo come passato per riferimento. Per evitare effetti secondari inattesi, è meglio indicare soltanto al momento della dichiarazione della funzione quali parametri saranno passati per riferimento.

short_open_tag(boolean): Indica se abilitare o meno la forma abbreviata dei tag di apertura del PHP (`<? ?>`). Se si desidera utilizzare il PHP in combinazione con l'XML, occorre disabilitare questa opzione per potere abilitare la riga `<?xml ?>`. In alternativa occorre stampare il testo con il PHP, ad esempio: <?php echo '<?xml version="1.0"'; ?>. Inoltre, se disabilitato, occorre utilizzare la versione lunga dei tag di apertura del PHP (`<?php ?>`). Questo parametro influisce anche su `<?=`, la quale è identica a `<? echo`. L'uso di questa abbreviazione richiede l'abilitazione di `short_open_tag`. È ormai buona norma disabilitare la forma abbreviata ed utilizzare quella estesa, quindi è consigliabile impostare a Off questo valore.

- memory_limit(integer): Questo parametro imposta la dimensione massima in byte di memoria occupabile dallo script. Questo aiuta a impedire che script scritti male utilizzino tutta la memoria del server. Per potere utilizzare questo parametro occorre abilitarlo al momento della compilazione. Pertanto occorrerà includere nella configurazione la linea: --enable-memory-limit. Si noti che occorre impostare il parametro a -1 se non si desidera impostare limiti di memoria.

post_max_size(integer): Imposta la dimensione massima dei dati post. Questa impostazione influenza anche gli upload dei file. Per permettere upload di file di grandi dimensioni, il valore impostato deve essere maggiore di upload_max_filesize. Anche il limite di memoria, memory_limit, se abilitato, può limitare gli upload di file. In termini generali memory_limit dovrebbe essere maggiore di post_max_size. Il valore assegnabile è soggetto alle regole

sintattiche del file PHP.ini, quindi è possibile utilizzare delle abbreviazioni per specificare megabyte o gigabyte di dati: 20M o 1G. Se la dimensione dei dati post è maggiore di post_max_size, le variabili superglobale $_POST e $_FILES sono vuote. Questo può essere rilevato in diversi modi, ad esempio passando una variabile $_GET allo script che processa i dati, tipo <form action="edit.php?processed=1">, e verificare se $_GET['processed'] è impostata.

include_path(string): Elenco di directory in cui le funzioni require(), include() e fopen_with_path() cercheranno i files. Il formato è tipo la variabile d'ambiente PATH: una lista di directory separate da due punti in Unix, punto e virgola in Widnows. L'uso di . nel percorso di include indica, negli include relativi, la directory corrente.

extension_dir(string): Directory in cui il PHP cerca i moduli caricabili dinamicamente.

extension(string): Specifica quale modulo dinamico caricare quando viene avviato l'interprete PHP;

upload_tmp_dir(string): Directory temporanea utilizzata per il transito dei file durante l'upload. Deve avere i permessi di scrittura per gli utenti utilizzati dal PHP per girare. Se non indicata il PHP utilizzerà il default di sistema.

upload_max_filesize(integer): La dimensione massima di un file inviato. Il valore assegnabile è soggetto alle regole sintattiche del file PHP.ini, quindi è possibile utilizzare delle abbreviazioni per specificare megabyte o gigabyte di dati: 20M o 1G.

Alcune volte potrebbe capitare di dover specificare opzioni relative alle singole estensioni utilizzate all'interno dell'interprete PHP. Quando una proprietà è specifica di un determinato contesto questa viene preceduta dal nome che identifica questo contesto seguito da un punto. Vediamo alcune delle direttive contestuali utili:

output_buffering(boolean): è possibile abilitare il buffering automatico dell'output per tutti i file settando la direttiva ad On. È possibile anche limitare le dimensioni del buffer ad una dimensione predefinita, impostando la proprietà ad un intero;

output_handler(string): è possibile inviare tutto l'output bufferizzato ad una funzione che si occupi di effettuarne delle trasformazioni. La direttiva output_handler permette di specificare il nome della funzione da utilizzare di default prima di restituire l'output. Per esempio è possibile impostare il valore a ob_gzhandler per comprimere l'output, oppure a mb_output_handler per modificare automaticamente la codifica dei dati in uscita;

SMTP(string): Usato solo sotto Windows: Nome DNS o indirizzo IP del server SMTP che PHP deve usare per spedire posta elettronica con la funzione mail();

smtp_port(int): Usato solo sotto Windows: Numero della porta del server specificato da SMTP al quale connettersi quando si inviano email usando mail(); il valore predefinito è 25.

sendmail_from(string): Quale campo "From:" devono avere i messaggi inviati da PHP sotto Windows.

sendmail_path(string): Dove trovare il programma sendmail, solitamente /usr/sbin/sendmail oppure /usr/lib/sendmail. configure cerca di trovare il file e lo imposta di default, ma se non riesce

a localizzarlo, lo si può impostare qui. I sistemi che non usano sendmail devono impostare questa direttiva al wrapper che i rispettivi sistemi di posta offrono, se esistenti. Per esempio, gli utenti di Qmail possono normalmente impostarla a /var/qmail/bin/sendmail o /var/qmail/bin/qmail-inject. qmail-inject non necessita di nessuna opzione al fine di processare correttamente la mail. Questi parametri funzionano anche su Windows. Se si impostate smtp, smtp_port e sendmail_from saranno ignorate e verrà eseguito il comando indicato.

Vi sono moltissime altre proprietà con le quali è possibile ottenere un ottimo controllo del comportamento del motore di PHP, quindi è sempre buona norma consultare la documentazione ufficiale per comprendere come controllare le tecnologie che si intende utilizzare

Guida PHP teorica

di Gabriele Farina

Introduzione

Questa guida rappresenta una riscrittura aggiornata della Guida Teorica a PHP scritta da Edoardo Valsesia. L'aggiornamento si è reso necessario per adattare la guida al nuovo motore PHP versione 5. Nelle lezioni cercherò di trattare l'argomento PHP 5 in un modo un po' più teorico rispetto a come viene affrontato nelle altre guide presenti su HTML.it. Manterrò parte della trattazione dei contenuti fatta dall'autore precedente, aggiornandoli ove necessario ed aggiungendo dei paragrafi in cui tratterò argomenti specifici di PHP 5. Prenderò come acquisite alcune conoscenze base quali l'integrazione di PHP ed HTML.

Prima di dedicarvi alla lettura vi ricordo che nella programmazione è fondamentale la pratica, quindi vi consiglio di provare appena possibile ciò che avrete appreso per solidificare le conoscenze.

Buona lettura.

Le variabili

Come per tutti i linguaggi di programmazione anche con il PHP è possibile utilizzare le variabili, che sono rappresentate dal simbolo del dollaro ($) seguito dal nome della variabile.

```
$variabile = 1;
$Variabile = "Questa è una variabile";
```

Queste sono variabili, e sono differenti non tanto per il valore loro assegnato quanto per il loro nome: infatti, in PHP le variabili sono case-sensitive.

Ogni variabile in PHP può assumere un nome che inizia con una lettera dell'alfabeto o un underscore (_) e segue con una combinazione qualsiasi di lettere, numeri o underscore. È bene ricordarsi che PHP, durante il processo di inizializzazione, crea delle variabili speciali (chiamate **variabili superglobali**) che contengono diversi tipi di informazione e che mi appresterò a trattare tra poco. Queste variabili sono accessibili da qualunque posizione dello script ed hanno dei nomi riservati che non andrebbero sovrascritti se non in casi particolari. Le variabili in questione sono:

- **$_GET**: un array contenente tutti i parametri passati alla pagina tramite metodo GET (accodando all'URL, dopo un punto di domanda (?) una serie di assegnazioni di valori separate da &);
- **$_POST**: un array contenente tutti i parametri passati alla pagina tramite il metodo POST (solitamente attraverso un Form oppure attraverso delle chiamate create manualmente);
- **$_COOKIE**: un array contenente tutti i cookie validi nella pagina corrente con il rispettivo valore;

- **$_REQUEST**: un array che contiene i valori delle variabili precedenti, tutti insieme. In caso di omonimia delle variabili, queste sono soprascritte dando precedenza in base al valore della direttiva variables_order impostata nel file di configurazione di PHP (solitamente `php.ini`);
- **$GLOBALS**: un array contenente tutte le variabili che risultano globali nello scope corrente;
- **$_SERVER**: un array contenente delle variabili impostate dal Web Server oppure direttamente legate all'ambiente di esecuzione dello script corrente (come ad esempio il browser dell'utente o il suo indirizzo IP). Vi sono molte variabili associate a questo array, pienamente descritte nella documentazione ufficiale di PHP. Le più utili sono:
    - PHP_SELF: il nome del file dello script correntemente in esecuzione;
    - DOCUMENT_ROOT: la root da cui viene eseguito lo script corrente;
    - REMOTE_ADDR: l'indirizzo IP dell'utente che sta eseguendo lo script. Questo valore viene impostato in base ad un header impostato dal browser, quindi non andrebbe utilizzato come discriminante in situazioni di sicurezza critiche;
- **$_FILES**: un array contenente informazioni sui file inviati alla pagina tramite POST. Ad ogni chiave dell'array corrisponde un altro array contenente i dettagli sul file. Questi dettagli sono:
    - name: il nome del file caricato;
    - tmp_name: il path della cache temporanea del file;
    - size: le dimensioni in byte del file;
    - type: il mime type del file, se recuperabile;
    - error: un numero indicante lo stato dell'upload del file, che può essere utilizzato per controllare che l'upload sia avvenuto correttamente;
- **$_ENV**: un array contenente tutte le variabili d'ambiente accessibili da PHP;
- **$_SESSION**: un array contenente tutte le variabili di sessione accessibili dalla pagina corrente;

A fronte di tutto questo è importante ricordarsi di non sovrascrivere questi valori, dato che sono spesso fondamentali per la corretta esecuzione dello script.

Le costanti

Le costanti sono dei contenitori immutabili per dei valori semplici (stringhe e numeri), che possono essere accedute attraverso il loro nome senza che questi sia preceduto dal classico simbolo del dollaro ($).

Le costanti in PHP possono essere definite manualmente oppure essere impostate automaticamente da PHP in base al contesto ed alle librerie caricate. Le costanti vengono impostate manualmente usando l'istruzione `define()`:

```
define('MIA_COSTANTE', 1);
define('SECONDA_COSTANTE', prova);
```

È convenzione specificare dei nomi composti solamente da caratteri maiuscoli o underscore. Le costanti possono essere accedute in questo modo:

```
echo MIA_COSTANTE;
echo "<br />;
echo SECONDA_COSTANTE;
```

Per controllare che una costante sia definita effettivamente è necessario utilizzare la funzione `defined`, che accetta come argomento la stringa che identifica il nome della costante da controllare.

```
/*
Il codice che segue avrà un comportamento differente da quello che ci aspettiamo.
NON_DEFINITA viene trasformata in una stringa NON_DEFINITA, viene restituito
un notice da PHP e l'espressione viene valutata vera, eseguendo quindi il codice
tra graffe che invece vorremmo saltare.
*/


If(NON_DEFINITA)
{
// ....
}

// Questo è corretto
if(defined('NON_DEFINITA'))
{
// ...
}
```

PHP definisce automaticamente moltissime costanti, molte delle quali specifiche per le librerie importate. Le più importanti, indipendenti dalle librerie, e che spesso risultano utili sono le seguenti:

- **__FILE__**: il path del file in cui ci troviamo. In caso il file sia incluso da un altro in esecuzione __FILE__ avrà comunque il nome del file incluso;
- **__CLASS__**: il nome della classe in cui ci troviamo attualmente;
- **__FUNCTION__**: il nome della funzione in esecuzione;
- **__METHOD__**: il nome del metodo in esecuzione;
- **__LINE__**: il numero di linea corrente;

I tipi di dato - I

Il PHP supporta diversi tipi di dati, che non devono essere impostate dal programmatore ma sono automaticamente assunte dal motore a meno che il programmatore stesso ne forzi il tipo attraverso apposite funzioni quali `settype` (pratica comunque sconsigliata, conviene modificare il valore piuttosto che il tipo di dato di una variabile). I dati possono essere:

- Integer
- Float
- String
- Array
- Object
- Resource

Vediamoli uno ad uno

Integer

Gli Integers, o interi, possono assumere diversi valori numerici esprimibili in differenti notazioni.

```
$a = 18; # decimale
$a = -18; # decimale negativo
$a = 022; # notazione ottale; equivalente a 18 decimale
$a = 0x12; # notazione esadecimale, equivalente a 18 decimale
```

Probabilmente, almeno per iniziare, utilizzerete soprattutto i numeri decimali, ma sappiate che il linguaggio accetta anche altre notazioni rispetto a quella decimale.

Float

Questo tipo di dati sono semplicemente i numeri in virgola mobile, ad esempio 9.876; la sintassi per utilizzarli è anche qui alquanto semplice:

```
$a = 9.876;
$a = 9.87e6;
```

Come vedete PHP accetta anche la notazione esponenziale.

Strings

Sulle stringhe c'è molto più da dire rispetto ai tipi di dati precedenti. La sintassi di base per le stringhe è:

```
$string = "Io sono una stringa";
```

Se vengono utilizzate le virgolette (""), il contenuto della stringa viene espanso (o, tecnicamente, "interpolato") , come nell'esempio successivo:

```
$num = 10;
$string = "Il numero è $num";
```

che visualizzerà "Il numero è 10". Come in tutti i linguaggi, comunque, anche con il PHP ci sono i caratteri speciali che vanno fatti precedere da un simbolo di escape; ad esempio, provate il seguente esempio:

```
$num = 10;
$string = "Il numero è "$num"";
```

Chi pensa che l'output di tale codice sia 'Il numero è "10"' si sbaglia: a parte il fatto che, così come è scritto, lo script darebbe un errore di compilazione, le virgolette sono caratteri speciali, ma non per questo non è permesso utilizzarle; la sintassi corretta per il comando riportato sopra è:

```
$num = 10;
$string = "Il numero è \"$num\"";
```

Altri caratteri speciali sono:

```
\n -> newline
\r -> carriage return
\t -> tabulazione
\\ -> backslash
\$ -> simbolo del dollaro
```

L'alternativa ai caratteri di escape, quando non ci siano contenuti da espandere, sono gli apici (''); ad esempio:

```
$string = '$ è il simbolo del dollaro';
```

visualizzerà proprio ciò che è contenuto fra gli apici. Attenzione a non cadere nel più consueto degli errori:

```
$num = 10;
$string = 'Il numero è $num';
```

che non visualizzerà "Il numero è 10" bensì "Il numero è $num". Quindi possiamo affermare che, con gli apici, il contenuto della stringa viene riportato letteralmente, ossia com'è effettivamente scritto al suo interno.

Una sintassi alternativa per la definizione delle stringhe ormai quasi caduta in disuso è la sintassi `HEREDOC`:

```
$string = <<<EOT
Questa è una stringa
multilinea
e fatta utilizzando la sintassi
HEREDOC
EOT;
```

Questa sintassi presuppone l'utilizzo di un separatore di inizio e fine stringa. Preceduto da `<<<` e chiuso utilizzando il punto e virgola. Può essere utilizzato un separatore qualsiasi formato da caratteri, ma è convenzione utilizzare `EOT`.

I tipi di dato - II

Array

Il PHP supporta sia gli array scalari sia gli array associativi. In PHP, un array di valori può essere esplicitamente creato definendone gli elementi oppure la sua creazione può avvenire inserendo valori all'interno dell'array, ad esempio:

```
$a = array ("abc", "def", "ghi");
```

crea l'array definendo esplicitamente gli elementi dell'array, al contrario dell'esempio che segue:

```
$a[0] = "abc";
$a[1] = "def";
$a[2] = "ghi";
```

In questo caso, l'array viene creato con tre elementi; ricordiamo che il primo elemento di un array viene identificato dal numero "0": se ad esempio la lunghezza di un array è "5", esso conterrà sei elementi; l'elemento contrassegnato dall'indice "0", infatti, è il primo dell'array. Se invece, per aggiungere elementi ad un array (supponiamo che sia quello precedentemente creato) si utilizzano le parentesi quadre vuote, i dati vengono accodati all'array; ad esempio:

```
$a[] = "lmn";
$a[] = "opq";
```

In questo caso, l'array si allunga di 2 elementi e risulta:

```
$a[0] = "abc";
$a[1] = "def";
```

```
$a[2] = "ghi";
$a[3] = "lmn";
$a[4] = "opq";
```

Questo esempio è molto utile quando si vogliano accodare degli elementi all'array senza ricorrere a specifiche funzioni e senza dover andare a leggere il numero di elementi contenuti nell'array: tutto sarà accodato automaticamente e correttamente.

Gli array associativi si basano invece su coppie "name-value"; un esempio potrebbe essere:

```
$a = array(
"nome" => "Mario",
"cognome" => "Rossi",
"email" => "mario@rossi.com",
);
```

È interessante la possibilità della funziona array di annidare le entries, come nell'esempio che segue:

```
$a = array(
"primo" => array(
"nome" => "Mario",
"cognome" => "Rossi",
"email" => "mario@rossi.com",
),
"secondo" => array(
"nome" => "Marco",
"cognome" => "Verdi",
"email" => "mario@verdi.com",
)
);
```

Eseguire su questo array un comando del tipo:

```
<? echo $a["secondo"]["email"]; ?>
```

visualizzerà "mario@verdi.com".

Objects

In PHP si possono utilizzare anche gli oggetti. Dato che PHP 5 ha ampliato enormemente il supporto alla programmazione ad oggetti questa verrà trattata successivamente in modo approfondito. Vediamo comunque un esempio:

```
class visualizza
{
 public function esegui_visualizza () {
  echo "Visualizza un messaggio";
 }
}
$obj = new visualizza();
$obj->esegui_visualizza();
```

Iniziamo definendo la classe "visualizza", che contiene la funzione "esegui_visualizza" che non fa altro che visualizzare un semplice messaggio a video. Con lo statement "new" inizializziamo l'oggetto "$obj" e richiamiamo la funzione visualizza con l'operatore -> su $obj. Più dettagli, come detto, in seguito.

Operatori di base

Gli operatori utilizzabili con PHP sono simili a quelli utilizzati con gli altri linguaggi di programmazione; per comodità, li divideremo in differenti "famiglie": gli operatori aritmetici, gli operatori di assegnazione a, in generale, tutti gli altri operatori.

Gli operatori aritmetici

Gli operatori aritmetici sono i più semplici, e sono:

Addizione

$a + $b

Sottrazione

$a - $b

Moltiplicazione

$a * $b

Divisione

$a / $b

Resto della divisione

$a % $b

Fermiamoci per un attimo sugli ultimi due per notare come il risultato della divisione non è approssimato all'intero più vicino, ma riporta tutto il numero risultante; il numero dei caratteri dopo il punto da considerare è definito nel file `php.ini` alla riga:

precision = 14

Quindi, verranno riportati "solo" 14 numeri dopo la virgola, a patto che ci siano. Nell'esempio:

```
$a = 12;
$b = 5;
$c = $a / $b;
echo $c;
```

il risultato è 2.4 e verrà visualizzato proprio come 2.4, non come 2.40000000000000. Quindi, i 14 decimali vengono visualizzati solamente se esistono, e non indiscriminatamente come inutili zeri.

Il resto della divisione viene riportato da solo, senza il risultato della divisione stessa: se nell'esempio precedente avessimo utilizzato "%" al posto di "/", il risultato sarebbe stato "2".

Assegnazione

Spesso, purtroppo, gli operatori di assegnazione vengono confusi con l'operatore di uguaglianza; l'operatore "=" ha un suo significato, che non va confuso con quello di "==". L'operatore di assegnazione è il simbolo dell'uguale (=) che attribuisce ad una variabile un valore; ad esempio

$a = 2;

imposta per "$a" il valore "2". L'errore che si fa più spesso è scrivere qualcosa del tipo:

if ($a=2) {

```
// istruzioni;
}
```

che, letto da occhi inesperti, potrebbe sembrare un'espressione per affermare che, se $a è UGUALE a 2 deve venire eseguito il codice fra parentesi graffe. Ebbene, non è assolutamente così: se avessimo voluto scrivere ciò che è appena stato detto, avremo dovuto utilizzare:

```
if ($a == 2) {
// istruzioni; }
```

Altri operatori

Gli operatori che il PHP ci mette a disposizione sono molti, e vedremo in questa pagina i più importanti che non abbiamo ancora avuto modo di esaminare, in particolari gli operatori booleani, quelli matematici e quelli di incremento-decremento. Ad ogni operatore faremo seguire una breve descrizione.

$a & $b

operatore "And" ($a e $b);

$a && $b

come sopra, ma con una precedenza più alta;

$a | $b

operatore "Or" ($a oppure $b);

$a || $b

come sopra, ma con una precedenza più alta;

$a ^ $b

operatore "Xor" ($a oppure $b ma non entrambi);

!$a

operatore "Not" (vero se $a non è vera);

$a == $b

operatore di uguaglianza, vero se $a ha lo stesso valore di $b;

$a != $b

l'opposto di quanto sopra;

$a < $b;

$a è minore di $b;

$a <= $b

$a è minore o uguale a $b;

$a > $b

$a è maggiore di $b;

$a >= $b

$a è maggiore o uguale a $b;

$a ? $b : $c

questo, da utilizzarsi più con le espressioni che con le variabili, valuta $b se $a è vera, e valuta $c se $a è falsa;

++$a

incrementa di uno $a e la restituisce; $a++ restituisce $a e la incrementa di uno

--$a

$a--

come i due precedenti, ma il valore è decrementato.

Strutture di controllo: if, else e else if

If

Non possono mancare in un linguaggio di programmazione le strutture di controllo, che permettono al programmatore di far compiere delle azioni al programma nel caso si verifichino (o non si verifichino) determinate condizioni.

`If` permette di eseguire un blocco di codice se avviene (o non avviene) una determinata condizione; la sua sintassi è:

if (condizione) statement

Ad esempio, vogliamo che uno script ci indichi se due variabili sono uguali:

```
$a = 2;
$b = 2;
if ($a == $b) {
echo "\$a è uguale a \$b e valgono $a.\n";
}
```

`If` può anche essere utilizzato in maniera differente da quella appena esposta: eccone un esempio:

```
<? $a = 2; $b = 2; if ($a == $b) : ?>
$a è uguale a $b.
<? endif; ?>
```

il cui operato è identico a quello esposto sopra anche se molto meno leggibile.

`If` può essere utilizzato anche senza le parentesi graffe, utilizzando `endif` quando si intende terminare il blocco "if"; ad esempio:

```
if ($a == $b)
echo "\$a è uguale a \$b e valgono $a.\n";
endif;
```

Else

`Else` viene in "completamento" di `if`: con `if`, infatti, stabiliamo che succeda qualcosa all'avverarsi di una condizione; con `else` possiamo stabilire cosa accade nel caso questa non si

avveri. Un esempio potrebbe essere:

```
$a = 2;
$b = 3;
if ($a == $b) {
echo "\$a è uguale a \$b e valgono $a.\n";
} else {
echo "\$a è diversa da \$b.\n\$a vale \"$a\" mentre \$b vale \"$b\".\n";
}
```

Elseif

`Elseif` permette di specificare casualità non definite da `if`; un esempio potrebbe essere: "Se $a è uguale a $b visualizza $a, se $a è diversa da $b visualizza un messaggio d'errore, avvisa se $a non esiste, avvisa se $b non esiste". Con i soli `if` ed `else` non si potrebbe fare, ma con `elseif` diventa semplice:

```
if ($a == $b) {
echo "\$a è uguale a \$b.\n";
} elseif ($a != $b) {
echo "\$a è diversa da \$b.\n";
} elseif (!$a) {
echo "\$a non esiste.\n";
} elseif (!$b) {
echo "\$b non esiste.\n";
}
```

Notate due cose: possono esserci, in un blocco, tutti gli elseif di cui avete bisogno e, per chi conosca il Perl, attenzione a non scrivere `elsif` al posto di `elseif`: il significato è lo stesso ma "elsif" non viene riconosciuto dal PHP così come elseif non viene riconosciuto dal Perl!

Strutture di controllo: while, for, switch

While

La condizione `while` si comporta esattamente come in C; la sintassi di base è:

```
while (espressione) statement
```

Come `if`, inoltre, `while` può essere utilizzato con o senza parentesi graffe, aggiungendo nel secondo caso lo statement `endwhile`. I due esempi che seguono si equivalgono:

```
/* Primo esempio: */
/* $a viene incrementato e visualizzato */
/* finchè il suo valore non supera "5" */
$a = 1;
while ($a <= 5) {
print $a++;
}

/* Secondo esempio */
```

```
$a = 1;
while ($a <= 5)
print $a++;
endwhile;
```

Tradotte, queste espressioni fanno in modo che, finchè (`while`) $a è minore o uguale a "5", $a viene incrementato di un'unità e visualizzato.

For

Anche `for` si comporta esattamente come avviene in C o in Perl; dopo il `for`, devono essere inserite tre espressioni che, finchè restituiscono "TRUE" permettono l'esecuzione dello statement che segue: considerate questo esempio:

```
for ($a = 0 ; $a <=10 ; $a++) {
print $a;
}
```

che visualizzerà i numeri da "0" a "10". Nelle tre espressioni fra parentesi abbiamo definito che:

- $a ha valore 0;
- $a è minore o uguale a 10;
- $a è incrementata di una unità;

Quindi, per ogni valore di $a a partire da "0" fino a "10" $a viene visualizzato. È possibile omettere alcune operazioni (ricordandosi comunque di specificare sempre il punto e virgola come separatore) nel caso in cui l'inizializzazione, il controllo o la post esecuzione siano effettuate in altri luoghi oppure non debbano essere effettuate.

Switch

`Switch` permette di sostituire una serie di `if` sulla stessa espressione e, ovviamente, di agire dipendentemente dal valore di questa:

```
switch ($i) {
case 0:
echo "\$i vale 0";
break;
case 1:
echo "\$i vale 1";
break;
}
```

Abbiamo qui introdotto l'istruzione `break` che permette di uscire da un blocco nel caso si avveri una determinata condizione. Il costrutto switch è spiegato ampiamente nella guida base, quindi rimando al paragrafo relativo per ulteriori spiegazioni.

Le funzioni con iniziale A

In questa lezione analizzeremo alcune delle funzioni fornite nativamente da PHP. Le funzioni native di PHP sono talmente tante che necessiteremmo di centinaia di pagine solamente per trattarle in

modo superficiale. Per questo motivo ho deciso di dare una rapida occhiata ad alcune di queste, lasciandovi il compito di seguire la guida ufficiale per i dettagli o per lo studio delle altre funzioni.

Nelle altre guide, quella base e quella pratica, sono trattate le funzioni per l'interfacciamento ai database SQLite (http://database.html.it/articoli/leggi/895/sqlite-miniguida-alluso/) e MySQL (http://database.html.it/guide/leggi/87/guida-mysql/). Consiglio caldamente di leggerle per avere una panoramica più completa.

Le funzioni sono una delle parti più importanti di un linguaggio di programmazione, perchè permettono di eseguire determinate operazioni all'interno di uno script.

Le funzioni messe a disposizione dal PHP sono moltissime, e vederle tutte sarebbe inutile; ci soffermeremo invece su quelle più importanti ordinate alfabeticamente.

- **abs**: restituisce il valore assoluto di un numero:

```
$a= -3.4;
$aa = abs($a);
echo $aa, "n";
```

restituirà 3.4

- **acos**: restituisce l'arcocoseno dell'argomento:

```
$arg = 1;
$arc_cos = acos($arg);
echo "$arc_cosn";
```

restituirà 0.

- **array**: si veda la precedente spiegazione riguardo i tipi di dati;
- **asin**: restituisce l'arcoseno dell'argomento;
- **atan**: restituisce l'arcotangente dell'argomento;

Le funzioni con iniziale B

- **base64_decode**: decodifica una stringa codificata in MIME base64 (vedi sotto);
- **base64_encode**: codifica dati in MIME base64; ad esempio con:

```
$str = "Ciao, io sono pippo\n";
echo "$str\n";
$enc_str = base64_encode($str);
echo "$enc_str\n";
$dec_str = base64_decode($enc_str);
echo "$dec_str\n";
```

si passa allo script la stringa "$str" che viene prima codificata e visualizzata, poi decodificata e nuovamente visualizzata;

- **basename**: restituisce, dato un percorso, la componente di questo identificata da un nome di file; ad esempio:

```
$path = "/var/www/php/index.php";
```

```
$base = basename($path);
echo "$base\n";
```

restituirà "index.php";

- **bcadd**: somma due numeri;

```
$num = bcadd(1.334, 4.44554, 2);
echo "$num\n";
```

restituirà 5.77; la funzione "bcadd" prende come primi due argomenti due numeri e, come terzi argomento opzionale, il numero di cifre da visualizzare dopo la virgola;

- **bccomp**: compara due numeri: la funzione prende come argomento due numeri e, opzionalmente, un ulteriore numero che determina il numero di decimali da considerare dopo la virgola per considerare i due numeri uguali; restituisce "0" nel caso i due numeri siano uguali, "+1" se il numero di sinistra è maggiore di quello di destra e "-1" nel caso opposto. Considerate il seguente esempio:

```
$comp = bccomp(0.334, 0.301, 2);
echo $comp;
```

che restituirà "1"; ma se, al posto del "2" avessimo inserito uno oppure non avessimo inserito niente, il risultato sarebbe stato "0".

- **bcdiv**: divide due numeri, con le stesse modalità descritte per "bcadd" e "bccomp";
- **bcmult**: moltiplica due numeri, ed è possibile aggiungere un ulteriore parametro per limitare il numero di cifre dopo la virgola:

```
$molt = bcmul(2.31, 3.21, 2);
echo "$molt\n";
```

restituirà 7.41;

- **bcpow**: eleva a potenza due numeri, con la possibilità di specificare il numero di cifre dopo la virgola:

```
$pot = bcpow(2.3, 3, 2);
echo "$pot\n";
```

eleverà 2.3 alla terza potenza, approssimando il risultato alla seconda cifra decimale;

- **bcsqrt**: calcola la radice quadrata di un numero, con la possibilità di approssimare il numero di cifre dopo la virgola aggiungendo un secondo elemento alla funzione (come avveniva per altre funzioni matematiche viste sopra;
- **bcsub**: sottrae un numero da un altro, anche qui con la possibilità di approssimare le cifre dopo la virgola:

```
$num = bcsub(2, 5);
echo "$num\n";
```

restituirà "-3";

- **bin2hex**: converte una stringa di dati dal formato binario a formato esadecimale;

Le funzioni con iniziale C

- **ceil**: restituisce il valore intero più alto riferito al numero passato come argomento alla funzione:

```
$num = ceil(3.22112);
echo "$num\n";
```

  restituisce "4";

- **chdir**: cambia la directory di lavoro:

```
$dir = "/var/www/";
chdir($dir);
```

  restituisce TRUE se l'operazione ha successo, FALSO in caso contrario, ad esempio nel caso la directory non sia leggibile;

- **checkdate**: controlla che una data sia valida; per considerarsi valida, una data deve avere:
  - l'anno compreso fra "0" e "32767";
  - il mese compreso fra "1" e "12";
  - il giorno è compreso fra "0" ed il numero relativo al numero di giorni del mese a cui si fa riferimento;
- **chgrp**: tenta di cambiare il gruppo di un file a "gruppo"; la funzione accetta come argomenti il nome del file a cui si vogliono cambiare i permessi ed il nuovo gruppo di appartenenza:

```
chgrp(filename, gruppo);
```

  Nei sistemi Windows non funziona ma restituisce sempre vero.

- **chmod**: è equivalente al comando di sistema Unix "chmod" ed ha la stessa sintassi di chgrp;
- **chop**:è un alias per `rtrim` e cancella uno spazio vuoto o altri caratteri speciali dalla fine della stringa (la parte più a destra); spesso è utilizzato per eliminare i caratteri "\n" quando si riceve un argomento dallo standard input; il carattere eliminato può essere letto con:

```
$carattere = chop($string);
echo "$carattere\n";
```

- **chown**: cambia il proprietario di un file, come l'analogo comando di sistema Unix. Accetta come argomento il nome del file ed il nome del nuovo proprietario:

```
$file = "prova.txt";
chown($file, $user);
```

  Nei sistemi Windows, non fa niente e restituisce sempre vero (ed è peraltro inutile inserire questa funzione all'interno di uno script che non supporta il comando "chown");

- **chr**: restituisce il carattere ASCII specificato dal rispettivo numero; immagino sappiate, ad esempio, che la combinazione "Alt + 0126" restituisce la tilde (~); lo si può vedere con il seguente codice:

```
$ascii= "0126";
$char = chr($ascii);
```

```
echo "$char\n";
```

- **chunk_split**: divide una stringa in parti di "n" caratteri; il numero è passabile alla funzione dopo la stringa da dividere. Se non impostato, di default è assunto come 76; l'esempio

```
$string = "Questo è un corso per imparare il linguaggio php":
$split = chunk_split($string, 5);
```

restituirà:

```
Quest
o è u
n cor
so pe
r imp
arare
il l
ingua
ggio
php
```

La funzione è utile per l'encoding MIME base64 visto precedentemente con la funzione `base64_encode`;

- **closedir**: chiude una directory precedentemente aperta con la funzione `opendir()` - vedi;
- **copy**: crea la copia di un file:

```
$file = "prova.txt";
copy($file, "$file.bak");
```

- **cos**: restituisce il valore del coseno dell'argomento;
- **count**: conta gli elementi in una variabile; ad esempio:

```
$arr[0] = "abc";
$arr[1] = "def";
$arr[2] = "ghi";
$count = count($arr);
echo $count;
```

restituirà "3", visto che all'interno dell'array "$arr" sono presenti 3 elementi ($arr[0], $arr[1], $arr[2]);

- **crypt**: critta una stringa; la sintassi della funzione `crypt()` è:

```
crypt(string, salt);
```

In pratica, dovremo passare alla funzione la stringa che dovrà essere crittata e, opzionalmente, il seme con sui crittarla; se questo non è passato alla funzione, sarà generato in maniera random dal PHP stesso. Un esempio di crittazione di una stringa potrebbe essere il seguente:

```
$var = "Questa è una variabile";
$crypt = crypt($var, "aa");
echo $crypt;
```

che restituirà la stringa crittata;

- **current**: restituisce il primo elemento di un array:

```
$arr[0] = "abc";
$arr[1] = "def";
$arr[2] = "ghi";
$current = current($arr);
echo $current;
```

visualizzerà "abc";

Le funzioni con iniziale D

- **date**: visualizza la data in formato che è possibile definire; la funzione riconosce come validi i seguenti formati:

a: am/pm;
A: AM/PM
d: giorno del mese in due cifre, da "0" a "31";
D: giorno del mese in formato testo, ad esempio "Mon";
F: mese, in formato testuale, ad esempio "March";
h: ora nel formato "01", "12";
H: ora nel formato "00", "23";
g: ora nel formato "1", "12";
G: ora nel formato "0", "23";
i: minuti, nel formato "00", "59";
j: giorno del mese nel formato "1", "31";
l: giorno della settimana, ad esempio "Monday";
L: specifica se l'anno è bisestile o meno ("1" oppure "0");
m: mese nel formato "01", "12";
n: mese nel formato "1", "12";
M: mese in formato testuale corto, ad esempio "Jan";
s: secondi da "00" a "59";
S: suffisso inglese per gli ordinali, "st", "nd", "rd", "th";
t: numero di giorni nel mese corrente, da "28" a "31";
w: giorno della settimana in formato numerico ("0"=domenica);
Y: anno in quattro cifre, ad esempio "2000";
y: anno in due cifre, ad esempio "00";

Ad esempio, si potrebbe scrivere:

```
echo (date("l d F y H:i:s a"));
```

per avere la data corrente, che ad esempio potrebbe essere:

Friday 23 June 00 11:45:48 am

- **debugger_off**: disabilita il debugger PHP;
- **debugger_on**: abilita il debugger PHP;

- **decibin**: converte un numero da decimale a binario; ad esempio, il numero "10" decimale è "1010" in formato binario, e con del semplice codice PHP potremo scrivere:

```
$bin = decbin(10);
echo $bin, "\n";
```

che restituirà appunto "1010";

- **dechex**: converte un numero da decimale a esadecimale; la sintassi è identica a quella utilizzata per `decbin()`;
- **decoct**: converte un numero da formato decimale a formato ottale; la sintassi è la stessa utilizzata per `decbin()`;
- **define**: definisce una costante; come abbiamo visto nel capitolo sulle costanti, queste sono simili alle variabili solamente che non hanno il simbolo del dollaro davanti; per definire una costante si utilizza la seguente sintassi:

```
define("COSTANTE", "Questa è una costante");
echo COSTANTE;
```

L'esempio riportato sopra visualizzerà "Questa è una costante";

- **defined**: controlla che una certa costante esista: un esempio potrebbe essere:

```
define("COSTANTE", "Questa è una costante");
if (defined("COSTANTE")) {
echo "La costante è definita\n";
} else {
echo "La costante non è definita\n";
}
```

che visualizza un messaggio a seconda che la costante sia o meno definita;

- **die**: visualizza un messaggio ed esce dal programma:

```
if (defined($num)) {
echo "\$num è definito\n";
} else {
die ("\$num non è definito; impossibile proseguire\n");
}
```

- **dirname**: quando si specifica un path, riporta il path senza il nome del file finale: se ad esempio il path è "/home/yourname/public_html/index.php" la funzione restituirà solamente "/home/yourname/public_html":

```
$path = "/home/yourname/public_html";
echo(dirname($path));
```

Avrete notato che questa funzione fa l'esatto contrario della funzione `basename()`; combinando le due funzioni, si può avere il path completo di un file, compreso il suo nome;

- **diskfreespace**: restituisce lo spazio di disco libero; se volessimo ad esempio vedere quanto spazio rimane nella directory root della macchina, potremo scrivere:

```
echo(diskfreespace("/"));
```

Le funzioni con iniziale E

- **each**: restituisce il primo valore di un array utilizzando le keys 0, 1, key e value; se l'array è associativo, si può scrivere:

```
$array = array ("nome" => "valore", "nome2" => "valore2");
while (list($key, $val) = each ($array)) {
echo "$key => $val\n";
}
```

che restituirà:

```
nome => valore
nome2 => valore2
```

Se invece l'array non è associativo, il codice sarà:

```
$array = array ("nome", "valore", "nome2", "valore2");
while (list($key, $val) = each ($array)) {
echo "$key => $val\n";
}
```

ed il risultato:

```
0 => nome
1 => valore
2 => nome2
3 => valore2
```

Come avrete certamente notato, la funzione `each()` viene spesso utilizzata insieme a `list()`, che vedremo in seguito;

- **echo**: visualizza una o più stringhe; non penso ci sia molto da dire su questa funzione, vista sia la sua semplicità sia le numerose implicazioni in cui l'abbiamo vista in uso.
- **ereg_replace**: sostituisce un'espressione regolare con determinati valori; alla funzione devono essere passati tre argomenti: il primo indica il testo da sostituire, il secondo è il testo utilizzato per la sostituzione ed il terzo è la stringa da modificare; ad esempio:

```
$stringa = "Questo è un corso su ASP";
echo ereg_replace("ASP", "PHP", $stringa);
```

Notate che si sarebbe potuto scrivere anche:

```
echo ereg:replace("ASP", "PHP", "Questo è un corso su ASP");
```

che non avrebbe avuto molto senso, comunque.

- **ereg**: esegue il matching di un'espressione regolare. L'esempio fornito con la documentazione è alquanto eloquente:

```
if (ereg("([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})", $date, $regs)) {
echo "$regs[3].$regs[2].$regs[1]";
} else {
```

```
echo "Invalid date format: $date";
}
```

Tutto questo ciclo è fatto per controllare che una data sia in formato corretto. Vediamo il significato di "[0-9]{4})-([0-9]{1,2})-([0-9]{1,2}". Per chi conosca le espressioni regolari, non sarà difficile tradurre quanto sopra con "un numero da 0 a 9 ripetuto quattro volte seguito da un '-', da un numero da 0 a 9 ripetuto una o due volte, da un '-' e da un numero da 0 a 9 ripetuto una o due volte". Come spesso accade, leggere un'espressione regolare è molto più semplice che tradurla nel linguaggio parlato.

- **eregi_replace**: funziona esattamente come `ereg_replace()`, solamente che in questo caso l'espressione regolare è sostituita in maniera "case insensitive", ossia ignorando i caratteri maiuscoli e minuscoli;
- **eregi**: funziona esattamente come `ereg()`, solamente che in questo caso l'espressione regolare è sostituita in maniera "case insensitive";
- **error_log**: invia un messaggio di errore al file di log del Web Server, direttamente alla porta TCP dalla quale è arrivata la richiesta o in un file. La sintassi è:

```
error_log(message, mesage_type, destination);
```

Message_type è un numero che specifica dove deve arrivare il messaggio. Può essere:

0: il messaggio è inviato al logger del PHP o nel file specificato da "error_log";
1: il messaggio è inviato per email al parametro (probabilmente un valido indirizzo email) specificato in "destination"; 2: il messaggio è inviato al debugger;
3: il messaggio è inserito in append al parametro specificato in destination";

- **escapeshellcmd**: se si richiama un comando esterno da una shell, con questo comando si fa in modo che i metacaratteri della shell vengano fatti precedere da un carattere di escape per evitare che il comando produca degli errori;
- **exec**: esegue un programma esterno;
- **exit**: esce da uno script; il comando `exit()` è utile nei casi si voglia fermare uno script in caso qualcosa non soddisfi determinate condizioni, ad esempio:

```
if (condizione) {
esegui il blocco;
} else {
exit;
}
```

Ricordate che `exit()` non riporta un messaggio di errore come fa `die()`: se vi interessa dare "spiegazioni" sul perchè lo script termina, utilizzate `die()`, ma ricordate che non è possibile scrivere:

```
exit "Esco dal programma\n";
```

o meglio, è possibile ma non ha alcun effetto se non quello di uscire;

- **exp**: eleva "e" (2.71828.....) alla potenza riportata come argomento della funzione:

```
echo exp(3);
```

restituirà: 20.0855369...

- **explode**: divide una stringa secondo un determinato pattern. Ad esempio, volendo dividere una stringa contenente tre nomi separati da virgole possiamo scrivere:

```
$nomi = "Tizio,Caio,Sempronio";
list ($nome1, $nome2, $nome3) = explode (",", $nomi);
echo "$nome1\n$nome2\n$nome3\n";
```

che restituirà:

```
Tizio
Caio
Sempronio
```

`Explode()` è una versione semplificata di `split()`, che vedremo in seguito. Entrambe le funzioni, inoltre, sono molto utili nel caso ci sia la necessità di leggere determinati file contenenti delle liste;

Le funzioni con iniziali F e G

- **fclose**: chiude un puntatore ad un file precedentemente aperto con `fopen()`. Si veda `fopen()` per maggiori informazioni;
- **feof**: testa un puntatore ad un file per vedere se si è alla fine dello stesso;
- **fgetc**: restituisce il primo carattere del puntatore precedentemente aperto con `fopen()`; se ad esempio il puntatore $file punta al file "/tmp/prova.txt" che contiene solamente la riga "Ciao", un codice come il seguente:

```
$char = fgetc($file);
echo "$char\n";
```

restituirà "C" (ovviamente senza virgolette);

- **file_exists**: controlla se un file esiste, riportando TRUE in caso positivo o FALSE in caso negativo; ad esempio:

```
if (file_exists($file)) {
print "$file esiste;
}
```

Può essere molto utile utilizzare questa funzione nel caso sia necessario agire su uno o più file, in modo da agire sullo stesso solo nel caso questo esista senza rischiare di incorrere in inspiegabili "anomalie" dello script;

- **filegroup**: restituisce il gruppo al quale appartiene il file:

```
$filename = "/tmp/prova.txt";
$group = filegroup($filename);
echo "$filename appartinene al gruppo $group\n";
```

Ovviamente, la funzione è implementata nei soli sistemi multiuser;

- **filesize**: restituisce la grandezza di un file:

```
$filename = "/tmp/ptova.txt";
$size = filesize($filename);
echo "$filename -> $size\n";
```

- **filetype**: determina il tipo di file; i valori possibili sono: fifo, char, dir, block, link, file e unknown;
- **flock**: applica il locking ad un file; specificamente, `flock()` opera su un puntatore ad un file precedentemente aperto e le operazioni possibili sono:

  1: per il lock in lettura;
  2: per il lock in scrittura;
  3: per rimuovere il lock, di qualsiasi tipo sia;
  4: per impedire che `flock()` blocchi un file mentre applica il lock;

  Ad esempio, per applicare `flock()` ad un puntatore "$file" precedenemente definito occorrerà scrivere:

```
/* Per impedire che il file sia letto*/
flock($file, 2);
/* Codice per lavorare sul file */
.....
/* Per rimuovere il flock */
flock($file, 3);
```

- **fopen**: apre un file oppure un'URL. La sintassi è:

```
fopen(filename, mode);
```

  Ovviamente a "filename" corrisponde il nome del file o l'URL dello stesso, ed a "mode" la modalità con il quale questo deve essere aperto: si ha qui la possibilità di scegliere fra:

  r: apre il file in sola lettura;
  r+: apre il file in lettura ed in scrittura;
  w: apre il file in sola scrittura;
  w+: apre il file in lettura e scrittura;
  a: apre il file in sola scrittura e inserisce il puntatore alla fine del file ("w" lo inserisce alla fine)
  a+: apre il file in lettura e scrittura inserendo il puntatore alla fine del file;

  Ad esempio, per aprire un file locale in sola lettura scriveremo:

```
$file = fopen("/tmp/prova.txt", "r");
```

  Per un URL, invece:

```
$file = fopen("http://www.myhost.com/index.html", r");
```

  Per tutte le successive operazioni sul file, poi, dovremo agire direttamente sul puntatore

($file) e non direttamente sul file;

Le funzioni con iniziali H, I, J, K e L

- **header**: invia un qualsiasi header HTTP; ad esempio:

```
header("Pragma: no-cache");
```

Questa funzione è molto utile in diversi casi: ad esempio, per forzare un'autorizzazione, per inviare un errore "301" o via dicendo;

- **hexdec**: restituisce il valore decimale di un numero esadecimale;
- **implode**: come risulta dal nome, questa funzione non è che l'opposto di `explode`: la sintassi è identica, ma in questo caso restituisce una stringa con i valori separati dal primo argomento della funzione;
- **in_array**: restituisce valore vero se in un array è presente un determinato valore; un esempio potrebbe essere:

```
$numeri = array("1", "2", "3", "4", "5");
$num = 2;
if (in_array($num, $numeri)) {
print "$num è presente nell'array \$numeri\n";
}
```

- **is_array**: controlla se una data variabile è un array:

```
if (is-array($var)) {
echo "\$var è un array\n";
}
```

La stessa cosa viene fatta, con ovviamente la differenza dell'argomento, dalle funzioni:

- is_dir;
- is_double;
- is_executable;
- is_file;
- is_float;
- is_int;
- is_integer;
- is_link;
- is_long;
- is_object;
- is_readable;
- is_real;
- is_string;
- is_writeable.

- **isset**: restituisce TRUE nel caso la variabile esista, falso nel caso opposto; ad esempio, per vedere se esiste o meno una variabile, è possibile scrivere:

```
$a = 10;
```

```
echo isset($a), "\n";
echo isset($b), "\n";
```

che restituirà 1 e 0; ricordiamo che 1 è considerato valore di successo (TRUE), 0 di insuccesso (FALSE);

- **join**: unisce gli elementi di un array con una determinata stringa; l'uso è identico a quello di `implode()`;
- **key**: prende una chiave da un array associativo; un semplice esempio potrebbe essere:

```
$array = array("1" => "uno");
$chiave = key($array);
echo "$chiave\n";
```

che restituirà "1". Questa funzione è utile per estrarre tutte le chiavi di array associativi complessi;

- **link**: crea un hard link; la sintassi è:

```
link(target, link);
```

Le informazioni sui link (a proposito dell'esistenza del file a cui punta il link stesso) possono essere visualizzate con `linkinfo()`;

- **list**: assegna delle variabili come se fossero parti di un array; riprendiamo l'esempio fatto con each:

```
$array = array ("nome" => "valore", "nome2" => "valore2");
while (list($key, $val) = each ($array)) {
echo "$key => $val\n";
}
```

In questo caso, `list()` è utilizzato per "stilare" una lista di variabili che verranno estratte dall'array, senza ovviamente dare loro un valore ma lasciando alle parti successive del codice l'assegnazione dei loro valori. È inoltre utile notare che le variabili create da `list()` non assumono un solo valore, ma per ogni chiamata assumono un diverso valore, a seconda degli elementi presenti nell'array.

Le funzioni con iniziali M, O, P e R

- **mail**: funzione per l'invio di email; la funzione ha sintassi:

```
mail(To, Subject, Message, Altri_headers);
```

Supponendo di voler inviare un'email a "nome@host.com" con subject "Prova" e volendo specificare il nome del mittente, possiamo scrivere:

```
mail("nome@host.com", "Subject", "Questo è il corpo dell'email",

"From: mittente <mittente@host.net>);
```

Come vedete, inviare delle email tramite script PHP ed utilizzando la funzione "mail" è molto semplice. Ovviamente, nel file di configurazione, dovrete aver specificato la

locazione di sendmail (o analogo programma per l'invio delle email);

- **max**: restituisce il valore più alto di una serie di variabili, ad esempio:

```
$num = 1;
$num2 = 23;
$num3 = 0.3;
$max = max($num, $num2, $num3);
echo $max, "\n";
```

restituirà "23". Opposto a `max()` è **min()**, che adotta la stessa sintassi di `max()`;

- **mkdir**: crea una directory, di cui si deve specificare il percorso ed i permessi:

```
mkdir("/tmp/prova", 0777);
```

creerà la directory "/tmp/prova" con permessi impostati a 0777;

- **opendir**: apre una directory, della quale sarà possibile leggere gli elementi con `readdir()` e, successivamente, chiuderla con `closedir()`;
- **phpinfo**: è la funzione più "rappresentativa" del PHP, in quanto visualizza moltissime informazioni sul PHP stesso: l'uso dovrebbe essere noto:

```
phpinfo();
```

- **phpversion**: visualizza la versione di PHP che si sta utilizzando;
- **popen**: apre un puntatore ad un processo che deve essere chiuso con `pclose()`;
- **print**: visualizza una stringa a video come `echo()`;
- **rand**: genera un valore numerico in maniera casuale; se si volesse un valore compreso fra 10 e 20, si potrebbe scrivere:

```
$random = rand(10, 20);
```

- **range**: crea un array contenente un range di valori interi specificato; ad esempio, per creare un array con valori da 1 a 10 sarà necessario scrivere:

```
$array = range(1, 10);
```

- **rename**: rinomina un file: ad esempio, si usa:

```
rename("oldname", "newname");
```

per rinominare "oldname" come "newname";

- **rmdir**: come l'analogo somando unix, rimuove una directory; questo può essere fatto solo se:
  - la directory è vuota;
  - i permessi sulla directory lo consentono.
- **round**: arrotonda un numero:

```
$numero = round(2,3); /* restituisce 2 */
$numero = round(2.5); /* restituisce 3 */
$numero = round(2.6); /* restituisce 3 */
```

Come avrete notato, i decimali da 0 a 4 sono approssimati all'intero precedente, da 5 a 9 all'intero successivo

Le funzioni con iniziali S, T e U

- **shuffle**

ordina in modo casuale gli elementi di un array; ad esempio, per poter visualizzare gli elementi di un array in maniera casuale si potrebbe scrivere:

```
$num = range(0,10);
shuffle($num);

while (list(,$numero) = each($num)) {
  echo "$numero ";
}
```

- **sin**

restituisce il seno dell'espressione;

- **sizeof**

calcola il numero di elementi presenti in un array. Se ad esempio si volesse calcolare il numero di elementi in un array ed agire di conseguenza, si potrebbe scrivere:

```
$array = array("1", "2", "3", "4", "5");
$size = sizeof($array);

if ($size <= 10) {
  echo "L'array contiene meno di 10 elementi\n";
} else {
  echo "L'array contiene più di 10 elementi\n";
}
```

- **sleep**

mette lo script in pausa per un determinato numero di secondi, specificato come argomento della funzione; ad esempio, `sleep(10)` farà in modo che lo script venga sospeso per 10 secondi, per poi continuare normalmente;

- **split**

divide una stringa a seconda di un determinato pattern; ad esempio:

```
$linea = "tizio||caio||sempronio";
list ($uno, $due, $tre) = split("\|\|", $linea, 3);
print "1 => $uno\n2 => $due\n3 => $tre\n";
```

Da notare il fatto che è stato necessario inserire un carattere di escape (\) prima di ogni "|" nell'espressione da utilizzare per dividere la riga;

- **sqrt**

Restituisce la radice quadrata dell'argomento.

- **strcmp**

Esegue una comparazione su due stringhe: ad esempio:

```
$cmp = strcmp("Ciao", "Ciao a tutti");

if ($cmp == "0") {
  print "Le stringhe sono identiche\n";
} elseif ($cmp < 0) {
  print "La seconda riga è più lunga della prima\n";
} elseif ($cmp > 0) {
  print "La prima riga è più lunga della prima\n";
}
```

Restituisce "`La seconda riga è più lunga della prima`". La funzione, infatti, restituisce "0" se le stringhe sono uguali, un valore minore di zero se la seconda è più lunga della prima e maggiore di zero se la prima è più lunga della seconda.

- **system**

Esegue un programma di sistema, ne restituisce l'output e ritorna allo script.

- **tan**

Restiruisce la tangente dell'argomento.

- **unset**

Elimina il valore di una variabile.

- **usleep**

Come `sleep()`, ma questa funziona blocca lo script per N microsecondi.

Ovviamente, la lista delle funzioni non termina qui ma, essendocene altrettante meno utili almeno per chi inizia a programmare con questo linguaggio, abbiamo preferito fermarci a questo punto. Altre specifiche le tratteremo nelle lezioni successive.

Funzioni relative alla crittazione

Php offre agli sviluppatori una serie di funzioni relative alla crittatura, legate alla libreria mcrypt (http://mcrypt.sourceforge.net/); tale libreria supporta moltissimi algoritmi di crittazione, alcuni più utilizzati ed altri meno. Gli algoritmi sono:

- DES
- TripleDES
- Blowfish
- 3-WAY
- SAFER-SK64
- SAFER-Sk128
- TWOFISH
- TEA
- RC2
- GOST

- RC6
- IDEA

Per funzionare con tale libreria, il PHP deve essere stato compilato con l'opzione `--with-mcrypt`. I comandi fondamentali sono quattro, tutti con la medesima sintassi:

- **mcrypt_cfb()**: cipher feedback; codifica byte per byte;
- **mcrypt_cbc()**: cipher block chaining: utile per l'encoding dei file con un grande margine di sicurezza;
- **mcrypt_ecb()**: electronic codebook: utilizzata per dati random, dove il livello di sicurezza non è altissimo;
- **mcrypt_ofb()**: output feedback: simile a cfb, ma è data maggiore attenzione agli errori.

La sintassi in generale è:

$encrypted = mcrypt_XXX(algoritmo, chiave, input, encode/decode)

dove:

- **XXX** è il metodo che si intende utilizzare (cfb, cbc, cfb o ofb);
- **algoritmo** è l'algoritmo che si intende utilizzare, con la sintassi:

MCRYPT_ALGORITMO

Ad esempio, si potrebbe utilizzare

MCRYPT_BLOWFISH

oppure

MCRYPT IDEA

- **chiave** altro non è che la chiave con cui si andranno a crittare i dati;
- **input** sono i dati da crittare;
- **encode/decode** indica alla funzione se si devono crittare o decrittare i dati; per questo, si usano rispettivamente:

MCRYPT_ENCRYPT

e

MCRYPT_DECRYPT

Vediamo ora un esempio: volendo crittare una semplice stringa di testo con chiave di crittatura "La mia chiave" utilizzando CFB con l'algoritmo IDEA, dovremo scrivere:

```
$stringa = "Una semplice stringa di testo";
$chiave = "La mia chiave";
$encrypted = mcrypt_cfb(MCRYPT_IDEA, $chiave, $stringa, MCRYPT_ENCRYPT);
```

Chiunque voglia poi leggere i nostri dati crittati ($encrypted) dovrà ovviamente conoscere la chiave, il metodo e l'algoritmo utilizzati; quindi potrebbe scrivere qualcosa del tipo:

```
$chiave = "La mia chiave";
$stringa = mcrypt_cfb(MCRYPT_IDEA, $chiave, $encrypted, MCRYPT_DECRYPT);
```

La sua variabile "$stringa", quindi, conterrà "Una semplice stringa di testo".

Funzioni legate al protocollo FTP - I

Fra i vari protocolli, PHP ci mette a disposizione una vasta libreria di funzioni legate al protocollo FTP (FILE TRANSFER PROTOCOL), per il trasferimento di file da un computer all'altro in una rete. Vediamone le principali.

ftp_connect

Questa è la funzione "principale" nel senso che ci permette di stabilire una connessione FTP fra la nostra macchina ed il server FTP remoto. La sua sintassi è:

$stream = ftp_connect(host, port);

dove host è il nome del server a cui intendiamo connetterci e port (opzionale) è la porta alternativa alla quale ci si vuole connettere; se questa non è specificata, viene utilizzata la porta di default per il protocollo FTP, ossia la 21. Nella variabile $stream, inoltre, viene immagazzinato appunto lo stream di dati che il client (in questo caso il PHP) riceve dal server, ossia i messaggi di connessione accettata (con i vari dettagli) o di connessione rifiutata.

Ad esempio, per connetterci alla porta di default del server FTP "ftp://ftp.host.com" utilizzeremo:

$stream = ftp_connect("ftp://ftp.host.com");

ftp_login

Dopo la connessione, abbiamo bisogno di identificarci in modo che il server ci permetta lo scambio dei dati. molti saranno abituati a non vedere tale fase visto che, con i più diffusi client FTP grafici essa è svolta in automatico utilizzando le informazioni di login (username e password) inseriti come opzioni per il collegamento, ma sappiate che questa è una fase di vitale importanza per la connessione. La sintassi della funzione è:

$login = ftp_login(stream, username, password);

Se ad esempio in precedenza ci eravamo collegati all'host "ftp.host.com", utilizzando la variabile "$stream", adesso potremo procedere al login vero e proprio con:

$login = ftp_login($stream, "utente", "password");

La variabile $login ci servirà per capire se il login è andato o meno a buon fine e conterrà il valore "1" per il successo, "0" altrimenti. Ad esempio, per vedere se continuare lo scambio di dati in seguito all'autorizzazione potremo utilizzare il valore assegnato a tale variabile e scrivere:

```
if ($login == "1") {
... # Fai il resto delle operazioni
} else {
echo "Autorizzazione non riuscita\n";
}
```

Una volta connessi, potremo sapere su che macchina stiamo lavorando con la funzione `ftp_systype()` che ha sintassi:

$system = ftp_systype($stream);

ftp_pwd

Questa funzione invoca il comando "pwd", ovvero "Print work directory", che potremo tradurre come "Visualizza la directory corrente". Per vedere a che directory veniamo connessi dopo il login, potremo scrivere:

$directory = ftp_pwd($stream);

dove $stream è sempre la variabile che abbiamo utilizzato per la connessione con `ftp_connect()`.

Funzioni legate al protocollo FTP - II

ftp_cdup e ftp_chdir

Queste due funzioni servono rispettivamente a muoversi alla directory superiore e a muoversi in una determinata directory all'interno del server.
La prima si utilizza con sintassi:

$var = ftp_cdup($stream);

La seconda invece:

$newdir = ftp_chdir($stream, "nuova_directory");

Se ad esempio al login siamo nella directory "/" e volessimo spostarci in "/var/wwwdata" potremo scrivere:

$newdir = ftp_chdir($stream, "/var/wwwdata");

ftp_mkdir e ftp_rmdir

Queste due funzioni invocano il comando "mkdir "(crea una directory) e "rmdir" (rimuovi una directory). La prima restituisce il nome della nuova directory, la seconda solamente i valori true o false. Potremo creare un piccolo loop e scrivere:

```
# Posizioniamoci in "/var/wwwdata".
$mydir = ftp_chdir($stream, "/var/wwwdata/");

# Creiamo la directory "prova" come sottodirectory di "/var/wwwdata"
$newdir = ftp_mkdir($stream, "prova")

# Cancelliamo la directory appena creata!
$deleted_dir = ftp_rmdir($stream, $newdir);

# Possiamo ora controllare il tutto con:
if ($deleted_dir == "1") {
print "Operazione completata con successo.\n";
} else {
print "Qualcosa non è andato per il verso giusto.\n";
}
```

Ovviamente l'esempio non ha molto senso in una vera connessione (perchè creare una directory e subito cancellarla?), ma è stato proposto per comprendere come utilizzare al meglio queste due funzioni.

ftp_nlist

Questa funzione è analoga al comando "dir", ossia il comando utilizzato per vedere i nomi dei file presenti in una directory. La sua sintassi è:

$list = ftp_nlist($stream, directory);

Ad esempio, possiamo portarci nella directory "/var/wwwdata" e leggerne i file con:

$newdir = "/var/wwwdata";
$list = ftp_nlist($stream, $newdir);

I risultati sono contenuti in un array, quindi un 'echo "$list"' non avrebbe alcun senso.

ftp_get

Funzione che richiama il comando GET, per scaricare un file dal server remoto. Dobbiamo specificare per la funzione, oltre al solito stream, il nome del file locale, il nome del file remoto e la modalità di trasferimento (FTP_ASCII o FTP_BINARY); la sintassi completa è:

$file = ftp_get($stream, local_filename, remote_filename, mode);

Ad esempio, volendo scaricare dal server il file "data.txt" (supponiamo di essere già all'interno della directory che lo contiene) inserendolo nella directory "/tmp" con nome "file.txt" in ASCII mode, scriveremo:

$file = ftp_get($stream, "/tmp/file.txt", "data.txt", FTP_ASCII);

Per vedere se l'operazione ha avuto o meno successo, possiamo operare in due modi: controllare se effettivamente il file c'è nel nostro disco oppure controllare il valore della variabile $file: se ha valore "1" allora l'operazione è stata completata, se ha valore "0" nessun file sarà stato scaricato sul nostro disco.

ftp_put

Questa funzione fa esattamente il contrario di `ftp_put()`, ossia carica un file sul server. La sua sintassi è:

$file = ftp_put($stream, remote_filename, local_filename, mode);

Le opzioni sono identiche alle precedenti, quindi possiamo fare l'esempio contrario del precedente: carichiamo il file locale "/tmp/file.txt" nella directory remota (siamo già in questa directory) con il nome "data.txt". Tutto in ASCII mode, ovviamente:

$file = ftp_put ($stream, "data.txt", "/tmp/file.txt", FTP_ASCII);

Anche qui, possiamo controllare in due modi: valutando il valore di $file oppure invocando la funzione `ftp_nlist()` per vedere se fra i file c'è anche "data.txt".

Funzioni legate al protocollo FTP - III

ftp_size

Restituisce le dimensioni di un dato file. La sintassi è:

$size = ftp_size($stream, remote_filename);

Per tornare agli esempi precedentemente fatti, vediamo di conoscere la grandezza del file "data.txt", che si trova nella directory in cui siamo al momento; basterà scrivere:

$size = ftp_size($stream, "data.txt");

in modo che la variabile $size contenga le dimensioni del file "data.txt".

ftp_mdtm

Restituisce la data di ultima modifica di un file, restituendola come Unix timestamp. La sintassi è:

$date = ftp_mdtm($stream, remote_filename);

Ad esempio, volendo sapere la data di ultima modifica del file "data.txt" possiamo scrivere:

$date = ftp_mdtm($stream, "data.txt");

Anche in questo caso, la variabile "$data" conterrà la data di ultima modifica del file oppure il valore "-1" in caso di insuccesso (file inesistente o casi del genere).

ftp_rename e ftp_delete

Come apparirà chiaro dai nomi, queste due funzioni servono per rinominare un file e per cancellarlo. La prima ha sintassi:

$name = ftp_rename($stream, oldname, newname);

dove "oldname" è il nome originario del file e "newname" è il nuovo nome che vogliamo assegnare al file.
Ad esempio, per rinominare il file "data.txt" in "dati.dat" possiamo scrivere:

$name = ftp_rename($stream, "data.txt", "dati.dat");

La variabile $name conterrà "1" se l'operazione ha avuto successo, "0" altrimenti (file inesistente o casi simili).

La funzione `ftp_delete()`, invece, si utilizza con sintassi:

$delete = ftp_delete($stream, file);

Ad esempio, per eliminare il file "dati.dat" presente nella "current-directory" possiamo scrivere:

$delete = ftp_delete ($stream, "dati.dat");

Anche in questo caso la variabile può contenere valore "1" (il file è stato eliminato) o "0" (qualcosa non è andato per il verso giusto).

ftp_quit

A questo punto, il nostro lavoro sul server è terminato e possiamo disconnetterci utilizzando la

funzione `ftp_quit()` che ha la semplice sintassi:

$quit = ftp_quit($stream).

È sempre consigliato invocare questa funzione invece di chiudere il programma in esecuzione, più che altro per una questione di rispetto verso il server.

Le estensioni

Nei precedenti capitoli abbiamo visto le principali funzioni legate al linguaggio per la creazione delle nostre pagine dinamiche. La differenza fra quelle funzioni e la famiglia di quelle che andremo a vedere è sostanziale: le prime sono presenti direttamente all'interno del motore (built-in), le seconde sono presenti in librerie aggiuntive che devono essere installate sul sistema e richiamate in maniera particolare.

Prima di tutto, vediamo di capire il meccanismo di caricamento dinamico di queste librerie: aprendo il file `php.ini` vedremo un paragrafo dedicato alle "Extension", ossia estensioni nel linguaggio stesso: per spiegarci meglio, potremo dire che queste sono un insieme di librerie che vengono richiamate al momento dell'esecuzione di uno script come avviene, in maniera analoga, per il caricamento dei moduli con un webserver.

La sintassi per il caricamento di queste estensioni di linguaggio è molto semplice: una volta che avremo installato la libreria sul sistema, non ci resta che aggiungere nel file `php.ini` la riga:

extension=libreria

È qui necessario un discorso mirato per i sistemi Unix ed i sistemi Windows: per entrambi la sintassi è identica, ma ovviamente il nome delle librerie e la loro estensione no.

Nei sistemi Windows, una libreria si riconosce dall'estensione ".dll", mentre per Unix questa è ".so": quindi, a seconda del sistema, dovremo utilizzare il corretto nome per la libreria e, soprattutto, la corretta estensione. Ovviamente, per chi abbia entrambi i sistemi installati e riesca da uno dei sistemi a vedere l'altro è chiaro che le dll non possono essere caricate su un sistema Unix e viceversa.

Nello scrivere il nome della libreria che ci interessa caricare, non dobbiamo soffermarci sul percorso completo, ma è necessario solamente il nome della stessa, ad esempio `pgsql.so` per i database Postgres. Questo perchè, nello stesso file, è presente un'altra linea di configurazione che definisce in quale directory sono presenti queste librerie: leggendo il file `php.ini` potrete trovare la riga `extension_dir = directory` che istruirà il motore sulla locazione standard delle librerie. Quindi, quando specifichiamo con `extension` una libreria che vogliamo sia caricata per l'esecuzione di uno script, vengono di fatto uniti `extension_dir` ed `extension`: se ad esempio `extension_dir` è `/usr/lib/php` e abbiamo impostato `extension=pgsql.so`, il PHP saprà che per caricare la libreria `pgsql.so` dovrà cercarla in `/usr/lib/php/pgsql.so`.

Inoltre, con l'istruzione `extension= ....` è possibile specificare non solo una libreria ma tutta la seri di librerie che ci possono fare comodo: ad esempio possiamo avere qualcosa del tipo:

extension=pgsql.so
extension=mysql.so
extension=gd.so
extension=imap.so

```
extension=ldap.do
extension=xml.so
```

e via dicendo; come noterete dalla seconda riga, poi, è possibile specificare anche due o più librerie per uno stesso "campo" in questo caso, ci sono due librerie per due database (Postgres e MySQL) che, oltre a non entrare in conflitto l'una con l'altra, potrebbero teoricamente anche essere utilizzate contemporaneamente, a patto che questo abbia un'utilità.

Nel caso si cerchi di richiamare una funzione non built-in all'interno di uno script, lo script visualizza una messaggio d'errore che ci avverte che stiamo cercando di utilizzare una funzione non riconosciuta: ad esempio, per i database, devono essere caricate le estensioni come detto prima e, solo dopo aver compiuto questo passo, sarà possibile utilizzare le funzioni ad essi relativi senza incorrere in messaggi d'errore, sempre che queste siano utilizzate in maniera corretta, ovviamente.

Il perchè delle estensioni

A questo punto, ci sarà sicuramente chi si chiederà il perchè di questa librerie aggiuntive (a volte molto importanti o addirittura essenziali) che devono essere scaricate, installate e richiamate indirettamente. La spiegazione è semplice: per non appesantire inutilmente il motore. Di per sè questo ha già un grande numero di funzioni built-in, che potremo definire le funzioni "principali" per il linguaggio, sebbene alcune possano sembrare di poca utilità. Immaginate ora che il motore avesse al suo interno anche tutte le funzioni relative a tutti i database che supporta: avrebbe un altro centinaio (abbondante) di funzioni al suo interno; e questo non sarebbe un male se il 95% di queste non fosse inutilizzato.

Chi, infatti, ha la necessità di lavorare con il PHP ed una decina di database differenti? È sicuramente meglio installare l'estensione per il database che si deve utilizzare e non appesantire il motore con funzioni che certamente non utilizzeremo mai. Inoltre, pensate anche alle funzioni della libreria GD: senza dubbio sono interessanti per i risultati che permettono di ottenere, ma quanti in realtà le utilizzano? È più semplice fornire tutte queste funzioni in un file separato e lasciare che questo venga installato ed utilizzato da chi ne ha veramente bisogno.

Se ci pensate, è quello che avviene per tutti i linguaggi di programmazione: esistono le primitive (che, in linea di massima, possiamo definire come le funzioni essenziali e built-in in un sistema) e le funzioni in qualche modo esterne che vengono richiamate all'occorrenza. Rimanendo nei linguaggi di scripting, un paragone può essere fatto ad esempio con il Perl: anch'esso ha una nutrita schiera di funzioni built-in nell'interprete, alle quali si aggiungono la quasi infinità di moduli che il programmatore può utilizzare all'interno delle proprie creazioni.

La programmazione ad oggetti

Con l'avvento di PHP 5 il modo di concepire la programmazione ad oggetti in PHP è cambiato radicalmente. Il modello ad oggetti semplificato che era presente fino alla versione 4 non è che un'ombra scialba di quello attuale. Prima gli oggetti erano solamente una funzionalità di supporto poco utilizzata e veramente troppo poco flessibile. Ora gli sviluppatori possono puntare su un supporto alla programmazione ad oggetti che avvicina il linguaggio ai concorrenti più blasonati.

In PHP 5 la definizione di una classe rispecchia molto più da vicino le esigenze degli sviluppatori entrerprise. Vediamo di analizzare la sintassi utilizzata con dei brevi esempi.

La **definizione di una classe** avviene utilizzando una sintassi simile a quella precedente, anche se

possiamo notare grosse differenze. In primo luogo il costruttore della classe ora non deve avere lo stesso nome della classe stessa ma deve chiamarsi `__construct`; in secondo luogo abbiamo la possibilità di definire dei distruttori (implementati nel metodo `__destruct`) che verranno chiamati ogni qualvolta il garbage collector di PHP distruggerà l'oggetto. Oltre a queste piccole differenze, finalmente è possibile specificare la visibilità di metodi ed attributi attraverso le parole chiave public, protected e private. Vediamo un semplice esempio di classe:

```
<?php

class Prova
{
   public $attr_public;
   private $attr_private;
   protected $attr_protected;

   public function __construct()
   {
      // operazioni di inizializzazione
   }

   public function __destruct()
   {
      // operazioni eseguite prima della distruzione
   }

   public function publicMethod()
   {
   }

   protected function protectedMethod()
   {
   }

   private function privateMethod()
   {
   }
}

?>
```

Oltre queste differenze ora è possibile (e necessario) specificare esplicitamente quali metodi o proprietà dovranno essere statiche. Solo le proprietà o i metodi statici possono essere acceduti utilizzando l'operatore `::`, mentre prima PHP permetteva l'utilizzo di quell'operatore per qualunque metodo.

Quindi la definizione di un metodo proprietà statica presuppone l'anteposizione della parola chiave `static` prima della definizione.

Ovviamente l'**ereditarietà** è ancora supportata, ma grazie alle aggiunte è notevolmente migliorata. Difatti possiamo sfruttare la possibilità di definire la visibilità di metodi e proprietà per facilitare il design dell'applicazione.

```
<?php
```

```
class ExtProva extends Prova
{
   public function __construct()
   {
      parent::__construct();
   }

   private function provaProtected()
   {
      $this-> protectedMethod();
   }
}

?>
```

Le classi astratte

Oltre a questo, PHP ha ampliato notevolmente le sue potenzialità introducendo le interfacce e le classi astratte. Un'**interfaccia** è una classe speciale che definisce solamente la struttura che dovranno obbligatoriamente avere le classi che la implementano. Non può essere istanziata.

```
<?php

interface ProvaInterfaccia
{
   public function mustBeImplemented();
}

class Prova implements ProvaInterfaccia
{
   public function mustBeImplemented()
   {
      // implementazione
   }
}

?>
```

Se una classe non dovesse implementare tutti i metodi definiti nell'interfaccia, PHP restituirebbe un errore.

Le **classi astratte** invece sono un sistema che permette di definire classi parzialmente completate che lasciano l'implementazione di alcuni metodi alle sottoclassi. Una classe astratta deve essere definita utilizzando la parola chiave abstract; lo stesso vale per quei metodi astratti della classe. Vediamo un esempio:

```
<?php

abstract class ProvaAbs
{
```

```
    public function prova()
    {
        $this->abstractMethod();
    }

    abstract protected function  abstractMethod();
}

class Prova extends ProvaAbs
{
    protected function  abstractMethod()
    {
        // metodo richiamato da ProvaAbs::prova()
    }
}

?>
```

Grazie alle interfacce ed alle classi astratte, PHP ha rafforzato notevolmente il suo supporto ai tipi di dato, permettendo l'introduzione del type hinting per i tipi di dato complessi. Possiamo specificare il tipo di oggetto che ci aspettiamo per un parametro di un metodo/funzione demandando a PHP il compito di effettuare il controllo:

```
<?php

function prova(ProvaAbs $arg)
{
    $arg->prova();
}

?>
```

In questo modo possiamo assicurarci un'integrità delle chiamate che prima era quasi impossibile se non con particolari controlli effettuati a runtime.

Le costanti di classe e altre funzionalità

Un'altra aggiunta interessante è che ora è possibile definire **costanti di classe**, accedendovi utilizzando l'operatore usato per le proprietà o i metodi statici:

```
<?php

class Prova
{
    const MIA_COSTANTE = 'valore';
}

echo Prova::MIA_COSTANTE;

?>
```

Un'altra funzionalità molto interessante, che prima era utilizzabile solamente attraverso l'estensione overload ma ora è built-in, è la possibilità di definire dei metodi particolari per catturare tutte le **richieste in lettura/scrittura** alle proprietà e le chiamate ai metodi, nel caso non fossero trovati nell'albero dei simboli di una classe.

Vediamo un semplice esempio:

```
<?php

class Prova
{
   public $prova;
   private $data;

   public function __construct()
   {
      $this->data = array();
   }

   public function __set($name, $value)
   {
      echo Scritta la proprietà .$name.<br/>;
      $this->data[$name] = $value;
   }

   public function __get($name)
   {
      echo Richiesta la proprietà .$name.<br/>;
      return $this->data[$name];
   }

   public function test()
   {
      echo test method;
   }

   public function __call($name, $args)
   {
      echo Richiamato il metodo .$name. con .count($args). argomenti;
   }
}

$prova = new Prova;
$prova->prova = 10;
echo $prova->prova;
$prova->ciao = ciao;
echo $prova->ciao;
$prova->metodoInesistente(1, 2, 3, 4, array());

?>
```

Prima di terminare questa breve trattazione della programmazione ad oggetti, vorrei parlare di un

aggiunta fatta a PHP che permette di caricare automaticamente i file contenenti le definizioni delle classi nel momento in cui non vengano trovate in fase di esecuzione. PHP richiama automaticamente la funzione **__autoload** quando non trova una classe:

```
<?php

function __autoload($classname)
{
   echo Classe richiesta: .$classname;
   require_once $classname..php;
}

$prova = new Prova();

?>
```

Ovviamente utilizzare direttamente il nome della classe non è un sistema corretto. Ricordiamoci sempre di controllare che il path risultante o richiesto sia realmente accessibile e sia corretta la sua richiesta.

Nuove classi built-in

Oltre al potenziamento del modello ad oggetti, gli sviluppatori hanno deciso di aggiungere alla quinta versione di PHP una serie di classi built-in molto interessanti. Queste classi arricchiscono il comportamento di PHP e la potenzialità del suo modello ad oggetti aggiungendo il supporto per strutture dato molto potenti derivate dai pattern di sviluppo più comuni ed utilizzati.

Il primo che ho deciso di trattare è il pattern **Iterator**. Un iteratore è un oggetto che si comporta come una lista da cui è possibile recuperare ciclicamente gli elementi, con la differenza che la lista non è (normalmente) già salvata in memoria ma gli elementi successivi a quello corrente vengono generati su richiesta. Questo permette iterazioni su liste virtualmente infinite o di cui non si conoscono a priori le dimensioni. Gli iteratori possono essere utilizzati in congiunzione con il costrutto `foreach` come se si trattasse di normali array.

Vediamo un esempio molto semplice di iteratore in PHP:

```
<?php
<?php

class MiaListaIterator implements Iterator
{
   private $array;
   private $valid;

   public function __construct($array)
   {
      $this->array = $array;
      $this->valid = false;
   }

   public function rewind()
```

```
    {
        $this->valid = (FALSE !== reset($this->array));
    }

    public function current()
    {
        return current($this->array);
    }

    public function key()
    {
        return key($this->array);
    }

    public function valid()
    {
        return $this->valid;
    }

    public function next()
    {
        $this->valid = (FALSE !== next($this->array));
    }
}

class MiaLista implements IteratorAggregate
{
    private $range;

    public function __construct($max)
    {
        $this->range = range(0, $max);
    }

    public function getIterator()
    {
        return new  MiaListaIterator($this->range);
    }
}

$prova = new MiaLista(500);
foreach($prova as $item)
{
    echo $item.<br />;
}

?>
```

Come potete notare dal codice intervengono le interfaccee `Iterator` ed `IteratorAggregate`. La prima serve per permettere a PHP di assicurarsi che l'iteratore recuperato segua una struttura precisa che permette di operare sull'elemento come se fosse un array

(recuperando la chiave corrente, il prossimo elemento, l'elemento corrente, o riavvolgendo), mentre la seconda serve per assicurarsi che ogni oggetto utilizzato da foreach come un'array possa restituire un iteratore in modo corretto, tramite getIterator.

L'esempio in realtà non è molto significativo perchè lo stesso comportamento potrebbe essere ottenuto utilizzando un normale array. Ma pensiamo al recupero di memoria che potrebbe avvenire in questo caso:

```
<?php

class MiaListaIterator implements Iterator
{
   private $max;
   private $valid;
   private $current;

   public function __construct($max)
   {
      $this->max  = $max;
      $this->current = 0;
      $this->valid = true;
   }

   public function rewind()
   {
      $this->current = 0;
      $this->valid = true;
   }

   public function current()
   {
      return $this->current;
   }

   public function key()
   {
      return $this->current;
   }

   public function valid()
   {
      return $this->valid;
   }

   public function next()
   {
      $this->valid = ++$this->current > $this->max;
   }
}

class MiaLista implements IteratorAggregate
```

```
{
   private $range;

   public function __construct($max)
   {
      $this->range = $max;
   }

   public function getIterator()
   {
      return new  MiaListaIterator($this->range);
   }
}

$prova = new MiaLista(5000000000);
foreach($prova as $item)
{
   echo $item.<br />;
}

?>
```

Con questo codice iteriamo su un numero molto elevato di elementi senza consumare eccessiva memoria. Sarebbe stato molto duro ottenere lo stesso risultato utilizzando un array di cinque miliardi di elementi. La SPL implementa iteratori per molte tipologie di dato e strutture (array, directory e molto altro), quindi consiglio di studiarla (http://database.html.it/guide/leggi/87/guida-mysql/) per evitare di ripetere le operazioni già fatte nativamente da altri.

Un'altra classe che ritengo molto interessante è **ArrayObject,** che permette ad un oggetto di assumere il comportamento di un normale array, con la possibilità di accedere ai suoi elementi utilizzando le parentesi quadre.

```
<?php

class Users extends ArrayObject
{
   private $db;

   public function __construct($db)
   {
      $this->db = $db;
   }

   public function offsetGet($index)
   {
      $result = $this->db->query("SELECT * FROM users WHERE name = '".$index."'");
      return $result->fetch_assoc();
   }

   public function offsetSet($index, $value)
   {
      $this->db->query("UPDATE users SET surname = '".$value."' WHERE name = '".$index."'");
```

```
    }

    public function offsetExists($index)
    {
        $result = $this->db->query("SELECT * FROM users WHERE name = '".$index."'");
        return $result->num_rows > 0;
    }

    public function offsetUnset($index)
    {
        $this->db->query("DELETE FROM users WHERE name = '".$index."'");
    }
}

$utenti = new Users;
echo $utenti['Gabriele'];
$utenti['Paolo'] = 'Rossi';
unset($utenti['Federico']);

?>
```

Questo esempio vi fa comprendere il funzionamento di `ArrayObject`: una volta implementati i metodi sopra elencati, possiamo tranquillamente operare sull'istanza dell'oggetto come se fosse un'array. Ovviamente sconsiglio di utilizzare un sistema simile per gestire gli utenti, ma `ArrayObject` può risultare molto utile in molte situazioni di programmazione avanzata.

Guida PHP pratica

di Gabriele Farina

Introduzione

La guida che vi state apprestando a leggere illustrerà con degli **esempi pratici** il mondo della programmazione con PHP 5. Per chi non l'avesse ancora fatto, consiglio la lettura della guida base e della guida avanzata in modo che si abbia completa padronanza dei concetti che qui andremo a mostrare.

La guida si articola in una serie di esempi molto semplici ma utili al fine di analizzare le situazioni pratiche più comuni che vengono affrontate normalmente dall'utente alle prime armi.

Per provare gli esempi sarà necessario aver configurato correttamente il proprio PC (sia esso equipaggiato con Linux o Windows) con PHP 5.1.2, MySQL 4.1 o superiore ed Apache. Su questo sito potete trovare due guide esaustive (una per Linux (http://php.html.it/guide/leggi/92/guida-php-su-linux/) e una per Windows (http://php.html.it/guide/leggi/94/guida-php-su-windows/)) che illustrano il processo di installazione e configurazione di un ambiente di sviluppo adatto.

Nei seguenti paragrafi parleremo di operazioni utili sui file, invio mail, gestione di database e sistema di ricerca. È fondamentale ricordare che la programmazione ha regole sintattiche ferree, ma su alcune scelte logiche e funzionali risulta molto soggettiva. Io cercherò di scrivere gli esempi nel modo più semplice e standard possibile, ma vi avverto che durante l'analisi di eventuali altri script potreste incorrere in scelte progettuali o organizzative molto diverse.

Per un'introduzione più completa a PHP rimando alla guida base (http://php.html.it/guide/leggi/99/guida-php-di-base/).

I primi esempi con PHP

Prima di iniziare con gli esempi più interessanti, è necessario comprendere come è strutturato un comune script PHP. Normalmente una semplice pagina PHP con il compito di visualizzare dei dati, avrà un aspetto simile:

```
<?php

require_once 'include/config.php';
require_once 'library/info.php';

$bands = array('Dark Tranquillity', 'Stratovarius', 'Blind Guardian', 'In Flames', 'Metallica');

?>

<html>
```

```
  <head>
    <title>Prova</title>
  </head>
  <body>
    <h3>I miei gruppi preferiti</h3>
    <ul>
    <?php
    $info = new InfoManager;
    foreach($bands as $band_name)
    {
      $band_info = $info->findInfos($band);
      echo "<li>".$band_name;
      if(!is_null($band_info))
      {
        echo '  (<a href="'.$band_info->site.'">'.$band_info->site.'</a>)';
      }
      echo "</li>";
    }
    ?>
    </ul>
  </body>
</html>
```

Premetto che questo, a mio parare, non è uno dei modi migliori per organizzare contenuti complessi o applicazioni alle quali lavoreranno più persone, ma ritengo prematuro trattare il discorso template in questa sede. Vi lascio alla lettura degli articoli presenti su HTML.it relativi a smarty ed agli altri template engine (http://php.html.it/articoli/leggi/909/template-engine-separare-programmazione-e-design/).

Tornando a noi il codice, che ovviamente non funzionerà se eseguito sulla vostra macchina causa la mancanza dei file config.php e info.php, si occupa di includere un file di configurazione comune, una libreria e visualizzare una lista di band musicali con eventualmente il loro sito internet tra parentesi. L'inclusione di file esterni è un processo molto importante nella programmazione e viene effettuato utilizzando i costrutti require, include o le loro varianti.

Nel nostro caso abbiamo deciso di salvare dentro `include/config.php` alcune variabili di configurazione comuni per tutte le pagine, dentro `lib/info.php` la definizione della classe InfoManager e di richiedere l'inclusione di questi file nel momento in cui la nostra pagina fosse richiamata da apache. La differenza tra require ed include è sottile ma molto significativa: con il primo richiediamo obbligatoriamente l'inclusione di un file. Se questi non esiste, PHP genererà un errore fatale e terminerà l'esecuzione. Con il secondo invece richiediamo un file che non riteniamo fondamentale per il corretto funzionamento del sistema. In questo caso PHP restituirà solamente un warning in caso di malfunzionamento e procederà con l'esecuzione dello script. Il suffisso "_once" permette di assicurarsi che il file richiesto sia incluso una sola volta.

Un'altra informazione che mi sento di fornirvi prima di continuare è quella che riguarda la gestione dell'output in PHP. Solitamente è buona norma (nel caso in cui non si utilizzino sistemi di template) costruire all'interno di variabili stringa i blocchi di codice HTML che dovremmo mandare in output, e poi utilizzare una sola istruzione `echo` alla fine del codice. Questo perché le operazioni di output sono molto dispendiose in termini di risorse, e risparmiare sulla chiamate ad `echo` (o agli altri sistemi di output) velocizza l'esecuzione delle pagine.

Un contatore di visite personalizzato

Nella guida base a PHP ed in quella avanzata ho trattato le operazioni per la gestione dei file. Ricordo che PHP gestisce internamente i file come delle risorse quelle quali effettuare operazioni di lettura, scrittura od analisi. Questo però non è sempre necessario, soprattutto nel caso in cui si effettuino operazioni statistiche o operazioni di alto livello quali il recupero delle singole righe o la lettura dell'intero contenuto di un file in un solo comando.

Per presentare un esempio pratico, ho deciso di mostrare e commentare un semplice blocco di codice che vi permetterà di aggiungere un contatore di visite personalizzato alle vostre pagine. Il blocco di codice in questione sarà una classe molto semplice, che eventualmente potrete estendere al fine di cambiarne il comportamento specifico.

Cominciamo con la definizione delle classi che utilizzeremo nel nostro codice:

```
<?php

interface DataProvider
{
   public function storeVisits($visits);
   public function readVisits();
}

class FileDataProvider implements DataProvider
{
   const FILE_PATH = 'counter.txt';

   private $fp;

   public function __construct()
   {
      if(!file_exists(FileDataProvider::FILE_PATH))
      {
         $fp = fopen(FileDataProvider::FILE_PATH, 'w+');
         fwrite($fp, '0');
         fclose($fp);
      }

      $this->fp = fopen(FileDataProvider::FILE_PATH, 'r+');
   }

   public function readVisits()
   {
      return intval(file_get_contents(FileDataProvider::FILE_PATH));
   }

   public function storeVisits($visits)
   {
      ftruncate($this->fp, 0);
      fseek($this->fp, 0);
      fwrite($this->fp, strval($visits));
```

```
    }

    public function __destruct()
    {
        fclose($this->fp);
    }
}

class SimpleCounter
{
    private $provider;

    public $visits;

    public function __construct(DataProvider $provider)
    {
        $this->provider = $provider;
        $this->visits = $this->provider->readVisits();
        if($this->isValid())
        {
            $this->provider->storeVisits(++$this->visits);
        }
    }

    protected function isValid()
    {
        return true;
    }
}

?>
```

Il codice è molto semplice:

- definisce un'interfaccia che si occupa di rappresentare la struttura base di delle classi che potranno occuparsi di recuperare e salvare informazioni sulle visite;
- implementa l'interfaccia con un sistema che salva le informazioni su disco;
- definisce una classe che, quando costruita, recupera le informazioni da un DataProvider, controlla la validità dell'utente corrente ed eventualmente incrementa le visite. IsValid è stata messa per permettere alle classi che estenderanno SimpleCounter di specificare sistemi di validazione personalizzati (come il controllo dell'IP) per creare counter più completi.

L'utilizzo della classe (che salveremo dentro `lib/counter.php`) è il seguente:

```
<?php

require_once 'lib/counter.php';
$counter = new SimpleCounter(new FileDataProvider);
echo "Questa pagina &egrave; stata visitata ".$counter->visits." volte";

?>
```

Ad ogni aggiornamento della pagina verrà incrementato il numero delle visite. Prima di testare lo

script è molto importante assicurarsi che la directory corrente abbia i permessi impostati correttamente, altrimenti il file non potrà essere creato o scritto.

Se volessimo impostare lo script in modo che l'aggiornamento delle visite non avvenga ad ogni reload della pagina da parte dello stesso utente ma che, una volta effettuata la prima visita, non venga aggiornato il numero per un determinato lasso di tempo, possiamo utilizzare vari sistemi. Il più semplice è sicuramente l'**utilizzo di un cookie**:

```
<?php

class CookieCounter extends SimpleCounter
{

   public function __construct(DataProvider $provider)
   {
      parent::__construct($provider);
   }

   public function isValid()
   {
      if(!isset($_COOKIE['counter_cookie']))
      {
         setcookie('counter_cookie', '1', time() + 3600);

         return true;
      }

      return false;
   }
}

?>
```

Salviamo questo file in `lib/cookiecounter.php` e modifichiamo leggermente il file `index.php`:

```
<?php

require_once 'lib/counter.php';
require_once 'lib/cookiecounter.php';

$counter = new CookieCounter(new FileDataProvider);

echo "Questa pagina &egrave; stata visitata ".$counter->visits." volte";

?>
```

Ecco fatto: un gioco da ragazzi. Ovviamente se non avete abilitati i cookie lo script non funzionerà.

Eseguire l'upload di un file

Un'altra delle operazioni molto frequenti è quella che concerne l'upload di file online. Quando si parla di upload bisogna tenere in considerazione una serie di elementi per il corretto funzionamento dei propri script. Per prima cosa è necessario che il file `php.ini` sia configurato in modo da limitare correttamente le dimensioni dei file da caricare: se nella nostra applicazione è possibile effettuare l'upload di file molto grossi, è necessario modificare la variabile di configurazione `max_upload_size` affinché il suo valore sia adatto alla nostra situazione.

In secondo luogo è importante ricordarsi che i form HTML che devono inviare i file devono avere l'attributo enctype impostato a "multipart/form-data", altrimenti i file non saranno inviati correttamente. Infine è bene ricordare che i file caricati non sono salvati all'interno della variabile globale `$_POST` ma all'interno di `$_FILES`, dalla quale saranno accessibili anche altre informazioni addizionali come il nome del file temporaneo locale in cui è stato salvato il file inviato, le dimensioni del file ed altro.

Vediamo nel dettagli come effettuare l'upload di un file. Prima di tutto creiamo un semplice form HTML che si occuperà di richiedere all'utente l'inserimento di un file:

```
<html>
   <head>
      <title>File upload</title>
   </head>
   <body>
      <form method="post" action="testupload.php" enctype="multipart/form-data">
         <input type="hidden" name="action" value="upload" />
         <label>Carica il tuo file:</label>
         <input type="file" name="user_file" />
         <br />
         <input type="submit" value="Carica online" />
      </form>
   </body>
</html>
```

Tengo a precisare che potremmo specificare un campo di input nascosto chiamato MAX_FILE_SIZE avente come valore la dimensione massima in byte del file da caricare. Questo valore, anche se controllato sia dal browser che da PHP, non è comunque sicuro, dato che può essere facilmente aggirato. È buona norma effettuare sempre la validazione dei dati da PHP prima di salvarli o elaborarli per evitare spiacevoli sorprese.

Tornando a noi, passiamo alla creazione del file `testupload.php`:

```
<?php

define("UPLOAD_DIR", "./uploads/");

if(isset($_POST['action']) and $_POST['action'] == 'upload')
{
   if(isset($_FILES['user_file']))
   {
      $file = $_FILES['user_file'];
      if($file['error'] == UPLOAD_ERR_OK and is_uploaded_file($file['tmp_name']))
```

```
        {
            move_uploaded_file($file['tmp_name'], UPLOAD_DIR.$file['name']);
        }
    }
}

?>
```

Queste poche righe di codice si occupano di controllare che i dati provenienti dalla pagina precedente siano corretti, controllano la validità del file caricato ed effettuano praticamente l'upload salvando i dati ove necessario.

L'array `$_FILES` permette di accedere ai seguenti valor:

- **name**: il nome del file caricato originariamente dall'utente;
- **tmp_name**: il nome temporaneo del file salvato da PHP in una cartella locale;
- **type**: il mime type del file (nel caso in cui il browser fornisca questa informazione);
- **size**: le dimensioni in byte del file;
- **error**: un codice numerico che indica l'eventuale errore durante il caricamento del file. Ad ogni numero possibile è associata una delle seguenti costanti:
    - UPLOAD_ERR_OK (0): Non vi sono errori, l'upload è stato eseguito con successo;
    - UPLOAD_ERR_INI_SIZE (1): Il file inviato eccede le dimensioni specificate nel parametro upload_max_filesize di php.ini;
    - UPLOAD_ERR_FORM_SIZE (2): Il file inviato eccede le dimensioni specificate nel parametro MAX_FILE_SIZE del form;
    - UPLOAD_ERR_PARTIAL (3): Upload eseguito parzialmente;
    - UPLOAD_ERR_NO_FILE (4): Nessun file è stato inviato;
    - UPLOAD_ERR_NO_TMP_DIR (6): Mancanza della cartella temporanea;

Con tutte queste informazioni risulta molto semplice analizzare un'operazione di upload e comportarsi di conseguenza. La funzione `is_uploaded_file` controlla che un file sia effettivamente uno di quelli caricati dall'utente, mentre `move_uploaded_file` effettua fisicamente lo spostamento del file temporaneo nel fiel di destinazione specificato.

Aggiungendo alcuni accorgimenti sulle dimensioni ed impacchettando tutto in una libreria è possibile gestire in modo molto completo il caricamento di file online senza rischiare di incorrere in problemi o malfunzionamenti. Come sempre è importante ricordarsi che il path di destinazione di `move_uploaded_file` deve essere un path valido.

Invio di email da una pagina web

Eccoci di fronte ad un argomento molto interessante e molto discusso. Spesso ci si ritrova a dover sviluppare applicazioni che necessitano di inviare email ad alcuni indirizzi di posta, siano queste per motivi pubblicitari, per motivi di sicurezza o per motivi di segnalazione.

Come la maggior parte delle altre operazioni eseguibili da PHP che hanno a che vedere con applicazioni o librerie esterne, anche l'invio delle mail necessita di una corretta impostazione del file php.ini. La funzione `mail()`, quella utilizzata per inviare qualunque tipo di email (sia testuale, sia HTML, sia con allegati) utilizzerà il server SMTP specificato nel file di configurazione. Quindi è importante impostare un indirizzo corretto, oppure il sistema non funzionerà e non sarà possibile eseguire correttamente i nostri script.

La funzione mail ha una struttura molto semplice che descriverò tra breve. Purtroppo però è una funzione a basso livello, che necessita di una conoscenza completa dello standard MIME per essere utilizzata correttamente. Lo standard MIME permette di definire messaggi formati da parti aventi tipologie di contenuto differente, quali, nel nostro caso, porzioni testuali, porzioni in HTML e file allegati.

La funzione mail accetta quattro parametri:

- il primo parametro indica l'indirizzo di posta del destinatario;
- il secondo l'oggetto della mail;
- il terzo il testo del messaggio;
- il quarto è opzionale ma è quello più potente: permette di specificare manualmente gli header del messaggio al fine di configurarlo correttamente per le nostre esigenze.

Normalmente l'invio di una mail testuale nel quale si desidera specificare anche il destinatario viene effettuato con questa breve riga di codice:

mail("g.farina@html.it (mailto:g.farina@html.it)", "mesaggio di prova", "Questo è un messaggio di prova testuale", "From: test@html.it (mailto:test@html.it)");

Questa chiamata a funzione restituirà un valore booleano rappresentante l'esito della chiamata a funzione, stamperà eventuali errori riscontrati durante l'invio (è sempre una buona idea far precedere la chiamata da una @ per evitare questo comportamento e non rovinare il layout della pagina in caso di errori) e, se tutto sarà andato per il verso giusto, invierà la mail.

La funzione mail non effettua alcuna validazione dei dati inseriti, quindi sarà opportuno che vengano specificati dei valori corretti o accettabili. Buona norma è quella di valutare la validità dell'indirizzo email di destinazione nel caso in cui sia specificato da un utente esterno attraverso un'espressione regolare:

```
$mail = $_POST['email'];
if(eregi("^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*(\.[a-z]{2,3})$", $mail))
{
   //Invio la mail
}else
{
   //restituisco un errore
}
```

Purtroppo questo sistema valida il formato della mail ma non la reale validità dell'indirizzo. Nel caso fosse necessario attestare la validità di questo, esistono sistemi che si connettono ai server SMTP della mail per sapere se l'indirizzo esiste o è fasullo.

Utilizzando la funzione `mail()` è possibile inviare email anche in formato HTML:

mail("g.farina@html.it (mailto:g.farina@html.it)", "mesaggio di prova", "Questo è un messaggio di prova testuale", "`MIME-Version: 1.0\r\nContent-type: text/html; charset=iso-8859-1\r\n`From: test@html.it (mailto:test@html.it)");

Ogni riga dell'intestazione deve essere separata da `\r\n` rispetto alla precedente. Dobbiamo anche assicurarci che nel messaggio siano contenuti solo `\n`, in modo da evitare spiacevoli inconvenienti.

L'**aggiunta di allegati al messaggio** è un processo leggermente più complesso che necessita della comprensione delle specifiche MIME. Dato che il discorso rischierebbe di diventare pesante e noioso, consiglio a tutti di iniziare utilizzando qualcuna delle centinaia di librerie presenti su internet che svolgono questi compiti, e di lasciarsi qualche ora libera per studiarsi la reference delle specifiche mime ed implementare una propria versione del sistema.

Il modello di un'applicazione dinamica

Eccoci arrivati ad uno dei punti più delicati e dolenti da toccare: la **struttura di un sito internet dinamico** sviluppato in PHP. Che la dinamicità data dal linguaggio possa portare solamente vantaggi ad una struttura che necessita di avere un sito sempre aggiornato, è ormai un dato di fatto. I server sono sempre più potenti, e quindi l'esigenza di avere materiale statico è solamente legata a situazioni di carico particolari oppure nel caso in cui non fosse realmente necessario alcun aggiornamento al sito.

Quello che rimane, e rimarrà, uno dei punti più discussi invece è quello che riguarda l'organizzazione delle proprie applicazioni web. Cercherò di spiegare brevemente il modello che solitamente adotto io, consigliandovi comunque di provare vie alternative per trovare quello di cui avete bisogno e con cui vi trovate più a vostro agio. D'altronde il modo migliore di programmare è soggettivo, e così deve rimanere.

Nel mondo dello sviluppo PHP, come in tanti altri ambienti, si sta affermando sempre di più l'**utilizzo di pattern** (MVC sopra tutti) adatti a separare il più nettamente possibile le sezioni alle quali dovranno lavorare persone con competenze diverse, permettendo in questo modo una minor dipendenza del lavoro ed una maggior produttività.

Inizialmente sviluppare in PHP significava scrivere codice di scripting all'interno delle nostre pagine HTML, riempiendolo di include o require per riutilizzare il codice e rendendo a noi ed a chi si occupava della grafica il lavoro molto pesante e ripetitivo. Erano moltissime le situazioni in cui una piccola modifica concettuale comportava svariate ore di lavoro e problemi a catena. Per non parlare del fatto che le query SQL necessarie per la gestione dei dati erano integrate nel sistema, rendendo il lavoro più complesso anche a chi si occupava di database.

Ora invece **il lavoro viene organizzato in modo diverso** e, a mio parere, decisamente più produttivo. Il concetto di separazione è stato sviluppato per diversi anni nel mondo dello sviluppo web, ed ha portato all'affermazione del pattern MVC, alla nascita dei template engine ed all'affermarsi dello sviluppo con XHTML e CSS. Ora la pratica comune è quella di separare la parte HTML del codice all'interno di template contenenti placeholder speciali che andranno sostituiti dinamicamente dai valori prodotti dal codice PHP. In questo modo, una volta decise le variabili di interfacciamento, ambo le parti saranno libere di lavorare sulla loro sezione specifica senza dipendere direttamente dall'altra. A questo si unisce l'affermarsi dell'utilizzo della persistenza degli oggetti, che permette a sua volta una buona separazione con la parte relativa ai database ed una miglior organizzazione dei propri progetti.

Purtroppo non posso addentrarmi troppo su questo discorso (spero di poterlo fare in un'altra sede) dato che il tempo scarseggia e dobbiamo parlare di molte altre cose. Per chi ne fosse interessato consiglio comunque di documentarsi sull'argomento. Nel codice che presenterò continuerò ad utilizzare una struttura "old style" perché si adatta meglio all'esposizione ed all'analisi dei contenuti.

Accedere al database MySQL

Nella guida base a PHP avevo accennato al fatto che da qualche tempo la libreria per l'interfacciamento con MySQL non viene fornita installata di default, ma è necessario abilitarla durante l'installazione. Sqlite (http://php.html.it/guide/lezione/2667/utilizzare-sqlite/) è il database il cui supporto è distribuito ufficialmente con PHP.

Se da un certo punto di vista questa cosa potrebbe essere interessante e utile (Sqlite è molto veloce,

snello ed oltretutto non si basa su un'architettura client/server ma è solamente una libreria per interrogare file relazionali), c'è da dire che nella maggior parte delle situazioni reali si utilizzerà un database relazionale per lavorare. Ho deciso di trattare MySQL soprattuto per il fatto che con l'uscita di PHP 5 è stata rilasciata la libreria `mysqli` che fornisce un accesso ad oggetti al database e risulta molto più comoda e potente da utilizzare.

Dopo esserci assicurati che la libreria `mysqli` sia disponibile sul nostro sistema, possiamo iniziare ad utilizzarla. La prima operazione da effettuare è ovviamente quella della connessione al server. In questi esempi presuppongo che ci sia un server MySQL in esecuzione sulla macchina locale, che sia presente un database di prova chiamato "test_html_it" e che l'utente utilizzato (root) non abbia impostata alcuna password.

La connessione può avvenire **utilizzando due diversi metodi**. Il primo è il più classico ma è anche quello sconsigliato, e prevede l'utilizzo di una funzione (`mysqli_connect`) avente lo stesso comportamento di `mysql_connect` con la piccola differenza che possiamo specificare come parametro aggiuntivo il nome del database al quale connetterci:

$mysql = mysqli_connect('localhost', 'root', '', 'test_html_it');

Il secondo metodo è invece più interessante e prevede la creazione di un oggetto utilizzando la classe mysqli:

$mysql = new mysqli('localhost', 'root', '', 'test_html_it');

I metodi potrebbero sembra simili, ma il primo restituisce una risorsa che dovrà essere gestita esplicitamente con chiamate a funzioni mysqli_*, mentre il secondo metodo restituisce l'istanza di un oggetto 'mysqli' che potrà essere interrogato direttamente. Il metodo ad oggetti segue meglio la filosofia di PHP 5 (anche se una migliore gestione delle eccezioni non avrebbe fatto sicuramente male a nessuno) e quindi è quello che approfondiremo.

Una volta aperta la connessione saremo liberi di lavorarci e di chiuderla alla fine delle nostre operazioni utilizzando il metodo close(): $mysql->close();

Interrogare e modificare una tabella

Una volta eseguita la connessione al nostro server MySQL (ricordiamoci sempre di controllare che questa sia andata a buon fine e che le informazioni passate siano corrette) siamo pronti per interrogare il database. L'interrogazione di una tabella di un database avviene tramite il metodo `query()` dell'oggetto `mysqli`, che esegure una query sul database selezionato restituendo eventualmente un recordset che rappresenta i risultati al seguito di un'operazione di selezione.

Tramite il metodo `query` possiamo eseguire qualunque tipo di query SQL permessa all'account con il quale siamo connessi, ricordandoci che, per motivi di sicurezza, non saranno eseguite query multiple. Nel caso fosse necessario eseguire più di una query (usando il punto e virgola come separatore) possiamo utilizzare il metodo `multi_query`, che ha un comportamento analogo ma esegue una o più query.

```
$mysql->query("INSERT INTO autori VALUES (', 'Gabriele', 'Farina')");
$results = $mysql->query("SELECT * FROM autori WHERE user LIKE 'gab%'");
echo "Autori corrispondenti a <b>gab%</b>: ".$results->num_rows."<br />";
$i = 1;
while($row = $results->fetch_assoc())
{
   sprintf("%d. %s %s <br />", $i, $row['surname'], $row['name']);
```

```
   ++$i;
}
```

Nel codice precedente inseriamo un record nella presunta tabella "autori", recuperiamo tutti quelli che hanno il nome che inizia per "gab", stampiamo il totale e la loro lista. Come possiamo notare nel caso in cui si effettui un'operazione di selezione viene restituito un oggetto (`mysqli_result`) che contiene metodi per iterare sulle righe recuperate, per conoscerne il totale e molto altro. La proprietà `num_rows` di questo oggetto definisce il numero di righe recuperate; invece attraverso `fetch_assoc`() viene restituito sotto forma di array associativo il prossimo record trovato. Esistono altri metodi fetch_* che permettono di recuperare i dati in formati differenti.

Una delle cose più interessanti dell'estensione **MySQL Improved** è il supporto per i prepared statement (http://php.html.it/articoli/leggi/1740/accesso-ai-database-in-php-con-lo-zend-framework/2/). Un prepared statement è una query SQL di qualunque tipo che viene parzialmente compilata dall'engine MySQL e che può essere eseguita un numero multiplo di volte con parametri differenti e con alte prestazioni.

Il metodo `prepare()` permette di creare un prepared statement:

```
<?php

$mysql = new mysqli('localhost', 'root', '', 'test_html_it');

$stmt_1 = $mysql->prepare("SELECT * FROM numeri WHERE valore = ?");
$stmt_2 = $mysql->prepare("INSERT INTO numeri VALUES (?)");

for($i = 0; $i < 100; ++$i)
{
   $stmt_2->execute($i);
}

for($i = 0; $i < 100; $i += 7)
{
   $results = $stmt_1->execute($i);
   $result = $results->fetch_assoc();
   echo "Valore recuperato: ".$result['value']."<br />";
}

$mysql->close();

?
```

Nel codice precedente abbiamo preparato due query compilate, una di selezione ed una di inserzione. Poi abbiamo popolato una tabella con cento numeri ed abbiamo recuperato i multipli di sette. L'esempio è completamente inutile, ma fa comprendere il funzionamento dei prepared statement.

Il punto di domanda rappresenta un parametro alla query: in base al numero di punti di domanda dovranno essere passati un egual numero di parametri al metodo `execute()` dell'oggetto restituito da `prepare()`. È importante ricordare che i punti di domanda non vengono sostituiti all'interno di stringhe, e che sono automaticamente quotati se necessario.

Grazie ai prepared statement possiamo effettuare centinaia di query simili ottenendo dei netti miglioramenti nelle performance ed una migliore organizzazione del codice.

Le tabelle e la struttura dei file

Come ultimo esempio pratico per chiudere questa breve guida ho deciso di implementare un sito di annunci molto semplice che sfrutti la libreria `mysqli` per interfacciarsi con il database.

Per prima cosa andiamo a creare le due tabelle che ci serviranno per il progetto utilizzando queste due query SQL:

```
CREATE TABLE authors (
   id       INT UNSIGNED NOT NULL AUTO_INCREMENT,
   name       VARCHAR(100) NOT NULL,
   surname    VARCHAR(100) NOT NULL,

   PRIMARY KEY(id)
);

CREATE TABLE articles (
   id       INT UNSIGNED NOT NULL AUTO_INCREMENT,
   author_id    INT UNSIGNED NOT NULL,
   title       VARCHAR(100) NOT NULL,
   article       TEXT NOT NULL,

   PRIMARY KEY(id),
   KEY(author_id)
);
```

Il nostro sito di comporrà di 3 pagine:

- index.php, in cui visualizzeremo una lista paginata di tutti gli articoli stampati in ordine di inserimento;
- show.php, in cui visualizzeremo i dettagli di un articolo;
- insert.php, in cui inseriremo un nuovo articolo;

Nel codice che presenterò non mi occuperò di effettuare operazioni di validazione o pulizia dell'input, e quindi informo che il codice non è assolutamente adatto per essere utilizzato in un ambiente di produzione. Quando si lavora è sempre bene tener presente che non bisogna mai prendere per sicuramente corretto l'input degli utenti, e bisogna sempre cercare di validarlo ed eventualmente ripulirlo per incorrere in spiacevoli inconvenienti.

Fatta questa premessa possiamo procedere con l'impostazione del nostro sito internet.

La visualizzazione degli articoli

La prima operazione che compiamo è quella che permette la visualizzazione degli articoli. Creiamo la nostra pagina PHP, poi procederò con i commenti e l'analisi:

```
<?php

$limit = 5; // articoli per pagina
```

```
$mysql = new mysqli('localhost', 'root', '', 'html_it_articles');
if(!$mysql)
{
   die("Errore di connessione al database, impossibile procedere");
}

$result = $mysql->query("SELECT COUNT(*) AS tot FROM articles")->fetch_assoc();

$page = isset($_GET['p']) ? $_GET['p'] : 1;
$totals = $result['tot'];
$totals_pages = ceil($totals / $limit);

$articles = $mysql->query("
   SELECT
      AR.id AS id,
      AR.title AS title,
      CONCAT(SUBSTR(AR.article, 1, 200),  ' ...') AS content,
      CONCAT(AU.surname, ' ', AU.name) AS author
   FROM
      articles AR,
      authors AU
   WHERE
      AR.author_id = AU.id
   ORDER BY id DESC
   LIMIT ".(($page - 1) * $limit).",".$limit);
?>
<html>
   <head>
      <title>Articoli</title>
   </head>
   <body>
      <ul>
         <li><a href="index.php">Lista articoli</a></li>
         <li><a href="insert.php">Inserisci un articolo</a></li>
      </ul>
      <p>Articoli totali: <?php echo $totals; ?></p>
      <table width="500px">
         <?php
         while($article = $articles->fetch_assoc())
         {
            printf('<tr>
                  <td>%d. <a href="show.php?id=%d">%s</a> (%s) </td>
               </tr>
               <tr>
                  <td><p>%s</p></td>
               </tr>
               <tr>
                  <td><hr /></td>
               </tr>',
```

```
                $article['id'],
                $article['id'],
                $article['title'],
                $article['author'],
                $article['content']
                );
        }
        ?>
    </table>
    <p>Pagina <?php echo $page; ?> di <?php echo $totals_pages; ?> <br />
    <?php
    if($page - 1 > 0)
    {
        echo '<a href="?p='.($page - 1).'">&lt; prev</a> | ';
    }else
    {
        echo '&lt; prev | ';
    }
    if($page + 1 <= $totals_pages)
    {
            echo '<a href="?p='.($page + 1).'">next &gt;</a>';
    }else
    {
            echo 'next &gt;';
    }
    ?>
    </p>
  </body>
</html>
```

Il codice è volutamente semplice e lineare: per prima cosa ci connettiamo al database mysql e ci assicuriamo che la connessione sia andata a buon fine. Poi recuperiamo il numero totale degli articoli presenti nel database e calcoliamo il numero di pagine da visualizzare in base al limite impostato all'inizio dello script.

Successivamente recuperiamo la lista di articoli che devono essere visualizzati nella pagina corrente, limitando la selezione in base alla pagina nella quale si sta navigando. Una volta effettuata la selezione procediamo con la produzione di codice HTML: per prima cosa stampiamo il totale degli articoli trovati, poi iteriamo sui risultati al fine di poter stampare le righe della tabella contenenti le informazioni richieste.

Infine, a tabella terminata, costruiamo una semplicissima barra di navigazione che permette di muoverci tra le pagine.

L'inserimento di un articolo e la visualizzazione del dettaglio

La visualizzazione del dettagli di un articolo è molto semplice e consiste nella selezione dei dati specifici di un articolo in base ad un parametro passato come argomento alla pagina (l'id dell'articolo).

```
<?php
```

```
$mysql = new mysqli('localhost', 'root', '', 'html_it_articles');
if(!$mysql)
{
   die("Errore di connessione al database, impossibile procedere");
}

if(!isset($_GET['id']))
{
   header('Location: index.php');
}

$article = $mysql->query("
   SELECT
      AR.id AS id,
      AR.title AS title,
      AR.article AS content,
      CONCAT(AU.surname, ' ', AU.name) AS author
   FROM
      articles AR,
      authors AU
   WHERE
      AR.author_id = AU.id AND
      AR.id = ".$_GET['id'])->fetch_assoc();
?>
<html>
   <head>
      <title>Articolo (<?php echo $article['id']; ?>)</title>
   </head>
   <body>
      <ul>
         <li><a href="index.php">Lista articoli</a></li>
         <li><a href="insert.php">Inserisci un articolo</a></li>
      </ul>
      <h3><?php echo $article['title']; ?></h3>
      <i><?php echo $article['author']; ?></i>
      <p>
         <?php echo $article['content']; ?>
      </p>
   </body>
</html>
```

Anche processo di inserimento di un articolo è molto semplice: lasceremo selezionare all'utente l'autore, specificare un titolo ed un testo e salveremo tutto nella tabella creata in precedenza rindirizzando la navigazione alla pagina principale.

Ecco il codice necessario:

```
<?php

$mysql = new mysqli('localhost', 'root', '', 'html_it_articles');
if(!$mysql)
```

```
{
   die("Errore di connessione al database, impossibile procedere");
}

if(isset($_POST['action']) and $_POST['action'] == 'insert')
{
   $mysql->query("INSERT INTO articles VALUES (", '".$_POST['author']."',
'".addslashes($_POST['title'])."', '".addslashes($_POST['article'])."')");
   header('Location: index.php');
}

$authors = $mysql->query("SELECT id, CONCAT(surname, ' ', name) AS fullname FROM authors
ORDER BY surname ASC");
?>
<html>
   <head>
      <title>Inserimento articolo</title>
   </head>
   <body>
      <ul>
         <li><a href="index.php">Lista articoli</a></li>
         <li><a href="insert.php">Inserisci un articolo</a></li>
      </ul>
      <h3>Inserisci un articolo</h3>
      <form action="" method="post">
         <input type="hidden" name="action" value="insert" />
         <label>Autore:</label> <select name="author">
            <?php
            while($author = $authors->fetch_assoc())
            {
               echo "<option value=".$author['id'].">".$author['fullname']."</option>";
            }
            ?>
         </select><br />
         <label>Titolo:</label> <input type="text" name="title" size="55"/><br />
         <label>Text:</label><br />
         <textarea name="article" rows="6" cols="60"></textarea><br />
         <input type="submit" value="Salva" />
      </form>
   </body>
</html>
```

L'unica cosa su cui fare attenzione è il fatto che il form rimanda alla stessa pagina (non avendo specificato alcuna action) e che quindi dobbiamo scrivere del codice che ci permetta di sapere se sono stati inviati dei dati in post ed in caso affermativo salvi i dati su database.

Ricordatevi che è fondamentale controllare i dati che arrivano in input in modo da non rischiare di incorrere in attacchi di injection o problemi di visualizzazione dovuti ad input scorretto. Nel mio esempio non ho fatto alcun controllo del genere, ma è molto importate nelle applicazioni reali se si vogliono evitare problemi.

Guida PHP e MySql pratica

di Claudio Garau

Introduzione

Oggi, ogni sito che vada al di là del semplice esperimento ludico ha bisogno di un database. Grazie a questo componente si può semplificare la procedura di inserimento, modifica e visualizzazione dei contenuti, rendere facilmente scalabile ('espandibile') il sito iniziale, rendere quasi indolore l'eventuale cambio di grafica o di struttura di navigazione. Il database, in poche parole, separa i contenuti del sito dalla loro presentazione grafica e li rende indipendenti e manipolabili.

Per poter utilizzare un database c'è naturalmente bisogno di un linguaggio in grado di interrogarlo e di interfacciare una pagina Web con i dati in esso contenuti. Una delle soluzioni che si sono imposte per semplicità di utilizzo, qualità del codice, diffusione è l'accoppiata PHP (come linguaggio) e MySQL (come database).

In questa guida vedremo, con esempi e soluzioni pronte all'uso, come si usa PHP per inserire, leggere, gestire i dati presenti in un database MySQL. Lo faremo con esempi pratici e specifici, senza lasciarci affascinare troppo dalle trattazioni teoriche e dalle lunghe introduzioni. Proprio per questa "fisionomia", la guida è dedicata agli utenti che per la prima volta si avvicinano a questo tema.

Prima di entrare nel vivo dei contenuti, è consigliabile avere pronto, sul proprio computer o online, un ambiente PHP/MySQL funzionante. Per sapere come configurarlo si può fare riferimento, in base al sistema operativo che si utilizza, alla Guida PHP su Windows Guida PHP su Windows o alla Guida PHP su Linux Guida PHP su Linux. Esistono anche alcuni software che sono in grado di installare in un sol colpo su Windows entrambi i programmi: li trovate descritti nell'articolo PHP, Apache e MySql su Windows in un clic PHP, Apache e MySql su Windows in un clic..

Basi di dati e modello relazionale

Le basi di dati, chiamate anche database o banche dati, sono degli archivi che contengono dei dati e integrano informazioni relative alla struttura dei dati stessi; in particolare, è possibile affermare che un database viene strutturato in modo da permettere procedure per la manipolazione dei dati attraverso apposite applicazioni, in esso i dati sono suddivisi sulla base di argomenti disposti in ordine logico, chiamati tabelle, suddivisi a loro volta per categorie chiamate campi.

I campi sono destinati a definire le caratteristiche delle informazioni archiviate, conservate all'interno di record, per cui ogni tabella può essere rappresentata come un piano cartesiano in cui le colonne conservano le proprietà dei dati mentre le righe memorizzano i dati stessi.

Si veda di seguito un semplice esempio di tabella, chiamata "Clienti" e destinata ad ospitare i dati relativi alla clientela di un'attività commerciale:

id_cliente	nome	cognome	indirizzo	email
1	Saverio	Bianchi	Via Milano	saveriobiachi@suamail.it

2	Mario	Rossi	Via Roma	mariorossi@suamail.it

Come è possibile osservare, le colonne della tabella sono contrassegnate da delle voci che sono anche i nomi dei campi, ognuna di queste voci contiene le informazioni relative alle caratteristiche dei dati contenuti nelle righe e al tipo di dato ad essi associato; nello specifico del caso proposto in esempio, il campo denominato "id_clienti" è associato ad un tipo di dato numerico intero positivo, mentre tutti gli altri campi sono associati ad un tipo di dato stringa.

Il campo id_clienti svolge la funzione di **chiave primaria**, ad esso è infatti associato un valore che è univoco per ogni record all'interno della tabella, si tratta di un campo fondamentale in quanto consente di possedere un elemento grazie al quale distinguere i diversi record archiviati; associare manualmente un identificatore per ogni record non sarebbe possibile in presenza di grandi quantità di informazioni, diventa quindi necessario ricorrere ad un'applicazione che sia in grado di svolgere questa operazione al posto dell'utilizzatore

Come anticipato, le procedure per la manipolazione dei dati, cioè operazioni come il loro inserimento, il loro aggiornamento, la loro ricerca e la loro rimozione, sono possibili grazie ad appositi programmi detti DBMS (Database Manager System) che consentono di avere una rappresentazione dei dati sotto forma di concetti.

Un database può essere composto da più tabelle, queste hanno la funzione di separare argomenti diversi che però possono avere tra loro delle relazioni; si introduce così nella descrizione dei database il cosiddetto **modello relazionale**, un modello che si fonda sull'assunto che le informazioni siano rappresentate da valori compresi all'interno di relazioni chiamate tabelle, per cui un database di tipo relazionale viene visto come un insieme di relazioni tra valori e il risultato di qualunque manipolazione dei dati può essere restituito sotto forma di tabelle.

Per chiarire quanto appena esposto, si pensi per esempio che nello stesso database che contiene la tabella "Clienti" proposta in precedenza sia presente anche una seconda tabella chiamata "Prodotti", destinata a contenere le informazioni relative ai prodotti in vendita presso l'attività commerciale:

id_prodotto	prodotto	marca	prezzo
1	stampante	Superprint	69,00
2	mouse	Mouseburger	7,00

Ora si immagini di avere una terza tabella, chiamata per esempio "Acquisti" e contenuta sempre nello stesso database, all'interno della quale memorizzare i dati relativi ai prodotti acquistati e ai clienti che li hanno comprati:

id_acquisto	id_prodotto	id_cliente	data
1	2	2	22-02-09
2	1	1	01-01-09

Mettendo in relazione gli identificatori delle due tabelle proposte in precedenza, si ha quindi la possibilità di relazionare ogni singolo acquisto con il cliente che lo ha effettuato e con il relativo prodotto, il tutto è possibile senza dover specificare ogni volta tutti i dati relativi ai prodotti disponibili o ai diversi nominativi escludendo qualsiasi ambiguità.

Ma nello stesso tempo la terza tabella mostra come sussistano delle relazioni all'interno di essa, interrogandola attraverso un DBMS sarà infatti possibile estrarre soltanto i record relativi ad un determinato cliente, ad un determinato cliente in una determinata data, alle vendite di uno specifico prodotto in un intervallo di date e molto altro.

Le basi di dati per le applicazioni Web based sono generalmente impostate sulla base dello stesso principio fondato sulle relazioni, ad esempio: in un blog vi saranno delle relazioni tra utenti e post pubblicati, tra commenti e relativi post, tra i commenti e gli utenti che li hanno scritti e così via.

Per la gestione di database relazionali sono necessarie particolari applicazioni dette RDBMS (Relational Database Manager System), MySQL è l'RDBMS Open Source più utilizzato per la creazione di applicazioni Web based realizzate in PHP, quindi durante i successivi capitoli di questa trattazione MySQL sarà l'applicazione di riferimento per l'archiviazione dei dati nei database e per la loro manipolazione.

La connessione a MySQL

Brevi segnalazioni prima di procedere

- nel corso di questa trattazione, le funzioni PHP non verranno precedute dall'operatore di silence (@), per permettere la visualizzazione di eventuali errori in sede di sviluppo; si raccomanda invece di utilizzare sempre l'operatore di silence in fase di produzione;
- tutto il codice utilizzato per questa guida può essere scaricato (http://www.html.it/guide/download/php_mysql/blog.zip) ed utilizzato liberamente;
- le versioni dei server utilizzate nella guida sono le seguenti: PHP versione 5.2.10, MySQL versione 4.1.22-standard, Apache versione 2.2.11 (Linux).

Perché un'applicazione realizzata in PHP possa utilizzare le informazioni contenute all'interno di un database questa deve poter avere accesso ad esse, a questo scopo l'applicazione dovrà poter comunicare con l'RDBMS che gestisce la base di dati, ciò è possibile attraverso un procedura iniziale e necessaria chiamata "connessione"; per evitare ambiguità è bene chiarire che la procedura di connessione avviene tra lo script e il programma che gestisce la base di dati e non tra lo script e la base di dati stessa; una volta terminata la procedura necessaria per la connessione all'RDBMS, sarà possibile avviarne una seconda chiamata di "selezione" del database da utilizzare.

Per aprire una connessione da un'applicazione in PHP al database manager MySQL, si utilizza una funzione nativa del linguaggio chiamata `mysql_connect()`, essa restituisce un identificativo di connessione MySQL in caso di successo, diversamente restituisce *FALSE*;

questa funzione richiede il passaggio di tre parametri che sono argomenti della funzione:

1. **hostname**: è il nome dell'host (o macchina ospitante) relativa al database manager MySQL a cui si desidera effettuare una connessione, esso identifica univocamente una postazione in Rete e può essere espresso sotto forma di indirizzo IP o stringa eventualmente seguita dal numero della porta attraverso cui l'RDMS attende le chiamate da parte dei client (i computer degli utenti che intendono interrogare i database), nel caso di un'installazione locale l'hostname è generalmente chimato "localhost";
2. **username**: è il nome dell'utente abilitato alla connessione e alla manipolazione di uno o più database; MySQL prevede un utente iniziale che è quello di root a cui sono associati i privilegi per la manipolazione delle basi di dati gestite, l'utilizzatore potrà poi creare altri utenti a cui associare un username e privilegi comparabili o inferiori a quelli previsti per il root;
3. **password**: per questioni di sicurezza è buona norma associare una password ad ogni nuovo utente MySQL creato, questa permetterà di autenticarlo al momento della connessione con il Database manager.

I tre parametri da passare a `mysql_connect()` possono essere espressi sia sotto forma di variabili che sotto forma di valori puri, l'ordine da rispettare è quello proposto in elenco, per cui sarà possibile utilizzare sia una forma del genere:

```
// hostname
$nomehost = "localhost";
```

```
// utente per la connessione a MySQL
$nomeuser = "username";
// password per l'autenticazione dell'utente
$password = "password";
// connessione tramite mysql_connect()
$connessione = mysql_connect($host,$user,$pass);
```

che la forma seguente

```
// connessione a MySQL tramite mysql_connect()
$connessione = mysql_connect("localhost","username","password");
```

Un buon metodo per permettere ad un'applicazione realizzata in PHP di connettersi a MySQL è quello di utilizzare una classe:

```
<?php
class MysqlClass
{
  // parametri per la connessione al database
  private $nomehost = "localhost";
  private $nomeuser = "username";
  private $password = "password";

  // controllo sulle connessioni attive
  private $attiva = false;

  // funzione per la connessione a MySQL
  public function connetti()
  {
   if(!$this->attiva)
   {
    $connessione = mysql_connect($this->nomehost,$this->nomeuser,$this-
>password);
       }else{
        return true;
       }
    }
}
?>
```

Per quanto la programmazione per oggetti possa presentare qualche complessità in più rispetto a quella che segue il paradigma procedurale, il meccanismo utilizzato dalla classe appena proposta risulta abbastanza semplice:

- i parametri per la connessione al DBMS vengono associati al modificatore "private" che li renderà disponibili soltanto all'interno della classe di appartenenza, secondo le regole di visibilità introdotte dal paradigma OOP nelle ultime versioni di PHP;
- viene effettuato un controllo sull'eventuale esistenza di connessioni attive;
- viene definita una funzione personalizzata (denominata `connetti()`) che potrà essere richiamata nel caso si voglia stabilire una connessione a MySQL; in questo caso alla funzione viene associato il modificatore *public*, ciò vuol dire che gli attributi e i metodi di cui sono dotati gli oggetti ad essa relativi saranno accessibili anche esternamente alla classe;
- nel caso in cui non sia già attiva alcuna connessione, i parametri necessari ad essa verranno passati alla funzione primitiva `mysql_connect()` che si occuperà di stabilirla, diversamente la richiesta di accesso al DBMS non verrà inviata.

Questa classe, che verrà implementata con nuove funzionalità nel corso della trattazione, potrà

essere salvata in un file chiamato ad esempio *funzioni_mysql.php* (o se si preferisce il classico *config.php*); essa potrà essere richiamata in qualsiasi momento da qualsiasi file con una semplice inclusione; perché la classe possa essere utilizzata dovrà essere istanziata, nello stesso modo sarà possibile utilizzare la funzione per la connessione a MySQL tramite una semplice chiamata:

```
// inclusione del file contenente la classe
include "funzioni_mysql.php"
// istanza della classe
$data = new MysqlClass();
// chiamata alla funzione di connessione
$data->connetti();
```

Chiusura di una connessione a MySQL

L'operazione opposta a quella che prevede la connessione a MySQL è quella relativa alla sua chiusura, a questo scopo PHP mette a disposizione una funzione nativa denominata `mysql_close()`; la chiusura di una connessione è una procedura importante perché consente di liberare risorse utili per il sistema, generalmente una connessione viene chiusa automaticamente quando termina l'esecuzione dello script che la richiama, ma è comunque buona norma utilizzare `mysql_close()` per evitare l'insorgere di possibili problemi o inutili sprechi di risorse.

La funzione restituisce *TRUE* nel caso in cui la chiusura della connessione abbia successo, diversamente restituisce *FALSE* quando invece si verifica un errore o un malfunzionamento che non permette la chiusura della connessione che le viene passata come parametro (identificativo di connessione). Se non viene specificato alcun parametro, allora la funzione chiude l'ultima connessione che è stata aperta.

Anche in questo caso è possibile introdurre una funzione personalizzata all'interno della classe *MysqlClass* proposta in precedenza e sfruttare le potenzialità del paradigma Object Oriented:

```
// funzione per la chiusura della connessione
public function disconnetti()
{
        if($this->attiva)
        {
                if(mysql_close())
                {
         $this->attiva = false;
             return true;
                }else{
                        return false;
                     }
        }
 }
```

Si analizzino le diverse componenti della funzione proposta:

- viene definita una funzione chiamata `disconnetti()` il cui compito sarà quello di chiudere eventuali connessioni attive;
- la funzione effettua un controllo sulla base del quale stabilirà se portare avanti o meno la procedura di chiusura, infatti questa sarà attuata soltanto nel caso in cui sia presente una connessione attiva;
- nel caso in cui sia stata aperta una connessione verrà allora richiamata la funzione nativa `mysql_close()` per la sua chiusura;

- la funzione prevede TRUE come valore di ritorno nel caso in cui la connessione sia stata chiusa con successo, FALSE in caso contrario.

La funzione per la disconnessione va utilizzata soltanto quando non è più necessario che l'applicazione mantenga un contatto aperto con MySQL, quindi, se successivamente a quella di connessione sono presenti istruzioni per interrogare o manipolare i dati, la funzione di chiusura andrà richiamata soltanto dopo di esse; anche in questo caso la chiamata alla funzione avverrà per istanza:

```
// chiamata alla funzione di disconnessione
$data->disconnetti();
```

È ovvio che questa chiamata potrà avvenire non prima dell'inclusione del file in cui è presente la classe che contiene la funzione.

L'estensione MySQLi

L'estensione **MySQLi** (MySQL improveded) è stata messa a disposizione di PHP per sfruttare alcune nuove funzionalità messe a disposizione dalle versioni di MySQL 4.1.3 e successive ed è disponibile per PHP 5 e release superiori.

MySQLi fornisce nuovi strumento per lo sviluppatore che desidera realizzare applicazioni in grado di interfacciarci con questo RDBMS:

- introduce la possibilità di sfruttare un approccio basato sul paradigma object oriented;
- permette l'utilizzo di un protocollo binario per la comunicazione che assicura prestazioni più elevate;
- fornisce il supporto per le prepared statements cioè istruzioni SQL precedentemente utilizzate e mantenute in cache per successive chiamate, con innegabili vantaggi a carico di ottimizzazione e prestazioni;
- mette a disposizione funzionalità avanzate per il debugging;
- permette di utilizzare stored procedures, query multiple e transazioni.

Le funzionalità messe a disposizione da MySQLi possono essere utilizzate sia all'interno di un contesto procedurale che in un'applicazione sviluppata utilizzando l'approccio per oggetti; infatti, per la maggior parte delle funzioni disponibili tramite questa estensione esiste un metodo corrispondente, ad esempio: la funzione `mysqli_fetch_array()` (molto simile alla funzione `mysql_fetch_array())` messa a disposizione dall'estensione MySQL) corrisponde al metodo `mysqli_result::fetch_array`. Per gli esempi di questa trattazione verrà privilegiato l'approccio OOP.

Utilizzando il paradigma per oggetti in MySQLi, la connessione al DBMS avviene per istanza, come nell'esempio seguente:

```
// connessione a MySQL con l'estensione MySQLi
$ mysqli = new mysqli("localhost", "username", "password", "nome_database");
```

Nel codice proposto viene effettuata un'istanza dell'oggetto `$mysqli` appartenente alla classe "mysqli", si noti come al costruttore non venga passato lo stesso numero di parametri necessari per la funzione `mysql_connect()`, cioè nome di host, nome utente, password utente, ad essi si aggiunge infatti il parametro relativo al nome del database predefinito per l'applicazione, non ci sarà quindi bisogno di una funzione come `mysql_select_db()`.

Si veda ora il seguente esempio pratico applicato sul database *mioblog*:

```
<?php
// connessione a MySQL con l'estensione MySQLi
$mysqli = new mysqli("localhost", "username", "password", "mioblog");

// verifica dell'avvenuta connessione
if (mysqli_connect_errno()) {
           // notifica in caso di errore
        echo "Errore in connessione al DBMS: ".mysqli_connect_error();
           // interruzione delle esecuzioni i caso di errore
        exit();

}
else {
           // notifica in caso di connessione attiva
        echo "Connessione avvenuta con successo";
}

// chiusura della connessione
$mysqli->close();
?>
```

`mysqli_connect_errno()` è una funzione che restituisce il numero identificativo di un eventuale errore prodotto durante un tentativo di connessione, nell'esempio proposto, se la funzione restituisce un valore significa che l'ultima connessione tentata è andata fallita, sarà quindi necessario bloccare l'esecuzione del codice seguente tramite la funzione `exit()`, nello stesso modo la funzione `mysqli_connect_error()` permetterà di visualizzare l'errore prodotto in connessione; diversamente, stabilita la connessione, questa potrà essere chiusa utilizzando il metodo `close()`, quest'ultimo non è da utilizzare obbligatoriamente ma ne è consigliabile l'impiego per le stesse motivazioni espresse in precedenza a proposito della funzione `mysql_close()`.

Per effettuare un'interrogazione tramite l'estensione MySQLi è possibile utilizzare un metodo apposito denominato query(), il metodo accetta come argomento la query SQL che si desidera lanciare al Database server.

Si veda il seguente esempio:

```
# estrarre risultati con il metodo mysqli_result::fetch_array
// query argomento del metodo query()
$query = " SELECT titolo_post FROM post ORDER BY data_post DESC ";
// esecuzione della query
$result = $mysqli->query($query);
// conteggio dei record restituiti dalla query
if($result->num_rows >0)
 {
// generazione di un array numerico
  while($row = $result->fetch_array(MYSQLI_NUM))
  {
   echo $row[0];
   echo "<br />\n";
  }
 }

// liberazione delle risorse occupate dal risultato
$result->close();
```

Il metodo `mysqli_result::fetch_array` genera un array dal risultato di un'interrogazione

passato come argomento al metodo `query()`, questo array potrà essere di tre tipi: numerico, associativo o di entrambi i tipi precedenti; nell'esempio appena esposto il metodo produce un array di tipo numerico grazie all'argomento MYSQLI_NUM, quindi verranno associati degli indici numerici ai diversi valori dei record risultanti dalla query.

L'esempio successivo, mostra come ottenere lo stesso risultato del codice precedente generando un array associativo grazie al quale i diversi record avranno come indice il nome dei campi, il tutto è possibile utilizzando l'argomento MYSQLI_ASSOC in luogo di MYSQLI_NUM:

```
# generazione di un array associativo
while($row2 = $result->fetch_array(MYSQLI_ASSOC))
{
echo $row['titolo_post'];
echo "<br />\n";
}
```

MySQL_BOTH permette invece di utilizzare sia gli indici numerici che quelli associativi; si noti l'utilizzo del metodo `mysqli_result::num_row` che restituisce il numero di record restituiti da una query.

MySQLi si presenta come una valida alternativa all'estensione MySQL, soprattutto per quanto riguarda le applicazioni che dovranno essere fruite da un alto numero di utenti, si tratta però di un'estensione ancora incompleta che al momento non è dato sapere se verrà supportata sufficientemente in futuro, per cui conoscere le funzioni disponibili grazie all'estensione MySQL rimane fondamentale.

Creazione del database

L'esempio pratico su cui si baserà questa trattazione prevede la creazione di un semplicissimo blog, cioè un sito internet aperto alla partecipazione degli utenti grazie ad un sistema di commenti associati ai diversi articoli postati nelle pagine Web; niente che debba essere utilizzato in fase di produzione, semplicemente uno spunto che potrebbe essere di aiuto durante lo sviluppo delle proprie applicazioni.

La creazione di un database, come quasi tutte le operazioni di manipolazione dei dati, in PHP è possibile passando un'istruzione scritta in linguaggio SQL, cioè il linguaggio che permette di comunicare con il DBMS, come parametro alla funzione denominata mysql_query(), che sarà possibile utilizzare per la creazione di un'altra funzione personalizzata da inserire, associata al modificatore "public", nella classe che gestisce le interazioni col DBMS:

```
//funzione per l'esecuzione delle query
public function query($sql)
 {
  if(isset($this->attiva))
  {
  $sql = mysql_query($sql) or die (mysql_error());
  return $sql;
  }else{
  return false;
  }
 }
}
```

La funzione proposta funziona secondo una logica abbastanza semplice ma possono essere utili alcuni approfondimenti:

prima di eseguire l'operazione prevista, la funzione controlla sempre che sia presente una connessione attiva;
nel caso in cui l'applicazione sia in comunicazione con il DBMS allora verrà eseguita la query passando alla funzione nativa mysql_query() l'istruzione rappresenta dalla variabile $sql;
nel caso in cui la query non dovesse avere successo a causa di un errore, allora la funzione mysql_error() si occuperà di intercettare l'eccezione e di informare l'utilizzatore relativamente alla tipologia del malfunzionamento in corso; se per esempio si tentasse di creare un database chiamato mioblog e questo dovesse essere già presente tra quelli gestiti da MySQL, si riceverebbe una notifica simile alla seguente: "Impossibile creare il database 'mioblog'; il database esiste", infatti, per evitare ambiguità il DBMS non permette la creazione di due basi di dati aventi lo stesso nome.
nel caso in cui non sia attiva alcuna connessione la funzione restituirà semplicemente FALSE.

La funzione mysql_error() può essere utilizzata per intercettare gli errori prodotti dall'esecuzione di qualsiasi funzione PHP per l'interazione con MySQL, quindi anche la parte relativa alla connessione al DBMS della funzione connetti() può essere modificata in questo modo:

```
$connessione = mysql_connect($this->nomehost,$this->nomeuser,$this->password) or die
(mysql_error());
```

L'istruzione necessaria per la creazione di un database chiamato "mioblog" è molto semplice e si basa sul comando SQL CREATE DATABASE, per cui all'interno dell'applicazione proposta il codice da eseguire sarà il seguente:

```
// inclusione del file contenente la classe
include "funzioni_mysql.php"
// istanza della classe
$data = new MysqlClass();
// connessione a MySQL
$data->connetti();
// chiamata alla funzione per la creazione del database
$data->query("CREATE DATABASE mioblog");
// disconnessione
$data->disconnetti();
```

Selezione del database

Ora che è stata creata la base di dati per il blog, perché questa possa essere utilizzata è necessario procedere con un'operazione denominata "selezione del database", questa procedura è possibile grazie ad un'apposita funzione nativa di PHP denominata mysql_select_db() che accetta come parametri il nome del database da selezionare e l'identificativo della connessione corrente aperta tramite mysql_connect().

Per evitare di affollare di funzioni la classe che gestisce le comunicazioni con MySQL è possibile gestire la fase di selezione all'interno della stessa funzione per la connessione al DBMS, prima di

fare questo sarà bene però creare un nuovo parametro, chiamato per esempio $nomedb, da aggiungere alla lista degli elementi gestiti dalla classe MysqlClass, per cui dopo i dati necessari per la connessione a MySQL si dovrà elencare anche:

```
// nome del database da selezionare
private $nomedb = "mioblog";
```

Anche in questo caso, l'argomento è stato associato al modificatore private in modo che sia visibile soltanto all'interno della classe di appartenenza.

A questo punto sarà possibile implementare la funzione connetti() in questo modo:

```
// funzione per la connessione a MySQL
public function connetti()
 {
  if(!$this->attiva)
   {
    if($connessione = mysql_connect($this->nomehost,$this->nomeuser,$this->password) or die
(mysql_error()))
     {
      // selezione del database
      $selezione = mysql_select_db($this->nomedb,$connessione) or die (mysql_error());
     }
    }else{
     return true;
    }
   }
```

Anche in questo caso viene prima effettuato un controllo per rilevare la presenza di una connessione già attiva, nel caso in cui questa non dovesse essere presente si procederà con la fase relativa alla connessione al DBMS a cui, posto che la connessione venga effettivamente aperta, seguirà la selezione del database il cui nome è contenuto come informazione all'interno del parametro $nomedb.

Si noti come sia nell'istruzione per la connessione a MySQL che in quella per la selezione della base di dati da utilizzare sia stata introdotta la funzione mysql_error(); come anticipato, essa permetterà di ricevere eventuali notifiche nel caso in cui dovessero rilevarsi dei malfunzionamenti in una o entrambe le due procedure.

Infine, è bene ricordare che a parte l'utente di root, tutti gli altri utenti creati tramite il DBMS potrebbero avere dei privilegi limitati o addirittura potrebbero non aver accesso ad un determinato database; prima di selezionare uno specifico database è quindi buona norma accertarsi che lo username utilizzato in connessione sia associato ad un utente che abbia i privilegi necessari per interagire con esso.

Creazione delle tabelle

Il blog engine d'esempio prevede la creazione di tre tabelle destinate alla memorizzazione dei dati relativi al login per l'amministrazione, ai post e ai commenti degli utenti a corredo degli articoli.

La creazione delle tabelle è possibile passando alla funzione mysql_query() l'istruzione SQL "CREATE TABLE" a cui far seguire il nome della tabella da creare e quelli dei diversi campi che andranno a comporla completi di caratteristiche (tipo di dato associato, dimensione massima consentita se prevista per il tipo di dato scelto, possibilità o meno di ospitare valori nulli e così via).

Avendo già a disposizione la funzione personalizzata query(), non sarà necessario crearne una ex novo per la gestione di questa fase, basterà invece passare ad essa le istruzioni necessarie per la creazione delle tre tabelle desiderate:

```
// inclusione del file contenente la classe
include "funzioni_mysql.php"
// istanza della classe
$data = new MysqlClass();
// connessione a MySQL
$data->connetti();

// creazione della tabella per il login
$data->query("CREATE TABLE `login` (
`id_login` INT( 1 ) NOT NULL AUTO_INCREMENT ,
`username_login` VARCHAR( 10 ) NOT NULL ,
`password_login` VARCHAR( 40 ) NOT NULL ,
PRIMARY KEY ( `id_login` ))");

// creazione della tabella per i post
$data->query("CREATE TABLE `post` (
`id_post` INT( 5 ) NOT NULL AUTO_INCREMENT ,
`titolo_post` VARCHAR( 255 ) NOT NULL ,
`testo_post` TEXT NOT NULL ,
`autore_post` VARCHAR( 30 ) NOT NULL ,
`data_post` DATE NOT NULL ,
PRIMARY KEY ( `id_post` ) )");

// creazione della tabella per i commenti
$data->query("CREATE TABLE `commenti` (
`id_commento` INT( 6 ) NOT NULL AUTO_INCREMENT ,
`id_post` INT( 5 ) NOT NULL ,
`autore_commento` VARCHAR( 30 ) NOT NULL ,
`testo_commento` TEXT NOT NULL ,
`data_commento` DATE NOT NULL ,
`approvato` ENUM( '0', '1' ) NOT NULL ,
PRIMARY KEY ( `id_commento` ) )");
```

```
// disconnessione
$data->disconnetti();
```

Il codice è anche disponibile in formato Sql facendo clic sul link Visualizza il codice sorgente qui in basso.

```
-- Struttura della tabella `commenti`

CREATE TABLE `commenti` (
  `id_commento` int(6) NOT NULL auto_increment,
  `id_post` varchar(5) NOT NULL default '',
  `autore_commento` varchar(30) NOT NULL default '',
  `testo_commento` text NOT NULL,
  `data_commento` date NOT NULL default '0000-00-00',
  `approvato` enum('0','1') NOT NULL default '0',
  PRIMARY KEY  (`id_commento`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

-- Struttura della tabella `login`

CREATE TABLE `login` (
  `id_login` int(1) NOT NULL auto_increment,
  `username_login` varchar(10) NOT NULL default '',
  `password_login` varchar(40) NOT NULL default '',
  PRIMARY KEY  (`id_login`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=21 ;

-- Struttura della tabella `post`

CREATE TABLE `post` (
  `id_post` int(5) NOT NULL auto_increment,
  `titolo_post` varchar(255) NOT NULL default '',
  `testo_post` text NOT NULL,
  `autore_post` varchar(30) NOT NULL default '',
  `data_post` date NOT NULL default '0000-00-00',
  PRIMARY KEY  (`id_post`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Nelle tabelle create sono stati inseriti numerosi campi che permettono di porre in relazione i dati all'interno delle tabelle stesse e tra i campi delle varie tabelle; un'idea più chiara delle relazioni create si otterrà con un'analisi attenta della struttura delle tabelle che vedremo nella prossima lezione.

La struttura delle tabelle

Si analizzi con attenzione la struttura delle tre tabelle create:

Tabella "login":

id_login: campo numerico intero della lunghezza di massimo 1 cifra ("INT(1)"); non può essere vuoto ("NOT NULL"), verrà incrementato automaticamente ad ogni inserimento di un nuovo record ("AUTO_INCREMENT") e verrà utilizzato nel database come chiave primaria della tabella ("PRIMARY KEY");
username_login: campo alfanumerico della lunghezza di massimo 10 caratteri ("VARCHAR(10)"); non potrà essere vuoto e verrà utilizzato per contenere nome associato all'utente autorizzato ad accedere all'amministrazione;
password_login: campo alfanumerico della lunghezza di massimo 40 caratteri; non potrà essere vuoto e conterrà la password per consentire l'accesso all'utente dotato dello username ad essa associato.

Campo	Tipo	Null	Extra
id_login	int(1)	No	auto_increment
username_login	varchar(10)	No	
password_login	varchar(40)	No	

Tabella "post":

id_post: campo numerico intero della lunghezza di massimo 5 cifre; non può essere vuoto, verrà incrementato automaticamente ad ogni inserimento di un nuovo record e sarà utilizzato come chiave primaria della tabella;
titolo_post: campo alfanumerico della lunghezza di massimo 255 caratteri; non potrà essere vuoto e conterrà il titolo associato ad ogni post;
testo_post: campo testuale di lunghezza non definibile ("TEXT"); non potrà essere vuoto e conterrà il testo di ogni singolo post.
autore_post: campo alfanumerico della lunghezza di massimo 30 caratteri; non potrà essere vuoto e conterrà i nomi degli autori dei post;
data_post: campo destinato ad ospitare la data di pubblicazione dei post, per esso è stato scelto un tipo di dato utilizzato appositamente per le date, cioè "DATE", che registrerà l'informazione nel formato "aaaa - mm - gg" (anno/mese/giorno); non può essere vuoto.

Campo	Tipo	Null	Extra
id_post	int(5)	No	auto_increment
titolo_post	varchar(255)	No	
testo_post	text	No	
autore_post	varchar(30)	No	
data_post	date	No	

Tabella "commenti"

id_commento: campo numerico intero della lunghezza di massimo 6 cifre; non può essere vuoto, verrà incrementato automaticamente ad ogni inserimento di un nuovo record e sarà utilizzato come chiave primaria della tabella;
id_post: campo numerico intero della lunghezza di massimo 5 cifre; non può essere vuoto e sarà utilizzato per ospitare l'identificativo univoco del post a cui è riferito il commento memorizzato nel record;
autore_commento: campo alfanumerico della lunghezza di massimo 30 caratteri; non potrà essere vuoto e conterrà i nomi degli autori dei commenti;
testo_commento: campo testuale di lunghezza non definibile; non potrà essere vuoto e conterrà il testo di ogni singolo commento;
data_commento: campo destinato ad ospitare la data di inserimento del commento, per esso è stato scelto il tipo di dato DATE; non può essere vuoto;
approvato: si tratta di un campo a cui è associato un tipo di dato ENUM che consente soltanto l'inserimento di due valori definiti in fase di creazione del campo, nel caso dell'esempio proposto i valori possibili saranno '0' (da associare ai commenti non approvati o non ancora approvati in fase di moderazione dall'amministratore) e '1' (da associare ai commenti approvati).

Campo	Tipo	Null	Extra
id_commento	int(6)	No	auto_increment
id_post	varchar(5)	No	
autore_commento	varchar(30)	No	
testo_commento	text	No	
data_commento	date	No	
approvato	enum('0', '1')	No	

Anche dal semplice elenco proposto saltano immediatamente all'occhio le diverse relazioni esistenti: i post sono per esempio in relazione tra loro se si considera un intervallo di date, i commenti sono in relazione con i post e anche per essi vale il discorso relativo alla data di inserimento e così via.

L'utilizzo di queste relazioni per lo sviluppo dell'applicazione destinata alla gestione di un blog sarà un argomento fondamentale per il resto di questa trattazione.

La tabella per il login

Un'applicazione per la realizzazione di un blog si compone in genere di due parti:

un back-end o pannello di controllo per l'amministrazione di post, commenti ed eventualmente altre funzionalità;
un front-end per la lettura dei post e dei commenti, cioè il blog vero e proprio.

Se l'accesso al front-end è generalmente consentito a tutti gli utenti, lo stesso discorso non vale invece per l'area di amministrazione a cui, per questioni di sicurezza è necessario accedere soltanto dopo aver superato positivamente una fase di autenticazione.

Lo schema del database mioblog prevede a questo scopo una tabella denominata login all'interno

della quale sono disponibili tre campi da popolare, i dati in essa inseriti permetteranno all'utilizzatore di disporre dei parametri necessari per l'autenticazione all'interno dell'area di amministrazione del blog.

In PHP l'inserimento dei dati all'interno di una tabella è possibile passando come parametro alla funzione mysql_query() un'istruzione SQL basata sul comando INSERT INTO a cui devono seguire tra parentesi tonde i nomi dei campi interessati dall'inserimento e i relativi valori da inserire in essi, introdotti dalla chiave VALUES e sempre tra parentesi tonde.

Di seguito verrà mostrata una funzione con modificatore public che sarà inserita tra le funzioni messe a disposizione dalla classe MysqlClass e che permetterà di effettuare una query di INSERT tramite il passaggio di tre argomenti:

$t: il nome della tabella in cui effettuare l'inserimento;
$v: i valori da inserire;
$r: i campi da popolare tramite i valori specificati dall'argomento precedente.

Di seguito verrà proposto il codice della funzione, successivamente si passerà all'analisi dei diversi passaggi:

```
//funzione per l'inserimento dei dati in tabella
   public function inserisci($t,$v,$r = null)
   {
       if(isset($this->attiva))
        {
                    $istruzione = 'INSERT INTO '.$t;
          if($r != null)
          {
             $istruzione .= ' ('.$r.')';
          }

          for($i = 0; $i < count($v); $i++)
          {
             if(is_string($v[$i]))
                $v[$i] = """.$v[$i]."";
          }
          $v = implode(',',$v);
          $istruzione .= ' VALUES ('.$v.')';

          $query = mysql_query($istruzione) or die (mysql_error());

          }else{
             return false;
          }
       }
}
```

La funzione inserisci() funziona sulla base di un semplice meccanismo:

il parametro "$t" rappresenta il nome della tabella in cui si desidera effettuare l'inserimento,

questa variabile verrà quindi passata come argomento al comando INSERT INTO;
"$r" è un array destinato a contenere i nomi dei campi da popolare, questo array non potrà essere vuoto, quindi la funzione ne controllerà il contenuto prima di eseguire la query;
"$v" è un array che rappresenta i valori che andranno a popolare i diversi campi i cui nomi sono anche i valori del vettore "$r";
grazie ad un ciclo for (rigo 12) verranno eseguite tante iterazioni quante sono i valori compresi nell'array "$v", dato che il linguaggio SQL prevede che i valori stringa debbano essere delimitati da apici singoli all'interno delle interrogazioni, ad ogni iterazione del ciclo la funzione "is_string()" intercetterà tutti i valori stringa presenti nell'array "$v" sostituendoli con il medesimo valore delimitato da apici.
l'array "$v" modificato verrà "imploso" (rigo 17) in modo da ottenere un'unica sequenza in cui suddividere ogni valore presente in esso tramite una virgola (sulla base della sintassi prevista dall'istruzione INSERT INTO nel linguaggio SQL);
ora la variabile $istruzione conterrà come valore l'istruzione SQL (rigo 18) da passare alla funzione mysql_query() (rigo 20) che potrà quindi essere eseguita;
tutti i passaggi previsti avranno luogo soltanto nel caso in cui sia stata aperta una connessione al DBMS, diversamente la funzione restituirà immediatamente FALSE ignorando qualsiasi altro passaggio.

Una volta creata l'apposita funzione, l'inserimento dei dati in tabella richiederà poche righe di codice:

```
// inclusione del file contenente la classe
include "funzioni_mysql.php"
// istanza della classe
$data = new MysqlClass();
// connessione a MySQL
$data->connetti();

// definizione delle variabili da passare alla funzione per l'inserimento dei dati
$t = "login"; // nome della tabella
$v = array ("admin",sha1("password")); // valori da inserire
$r =  "username_login,password_login"; // campi da popolare
// chiamata alla funzione per l'inserimento dei dati
$data->inserisci($t,$v,$r);
// disconnessione
$data->disconnetti();
```

La tabella login sarà quindi coinvolta da un'istruzione INSERT INTO in cui i campi username_login e password_login verranno popolati con i valori admin e password (modificabili arbitrariamente) e saranno questi che dovranno essere utilizzati per la procedura di autenticazione.

Si noti come, per ragioni di sicurezza, il valore relativo alla password non venga inserito nella tabella "in chiaro", cioè letteralmente, ma prima criptato tramite la funzione sha1() che produce una stringa di 40 caratteri da una qualsiasi stringa passata come parametro, per questo motivo è stata associata una lunghezza di 40 caratteri al campo password della tabella.

Funzioni PHP per l'estrazione dei dati

Prima di passare al codice necessario per l'autenticazione, verrà creata una piccola funzione personalizzata per l'estrazione dei dati dalla tabella deputata, essa si baserà su una specifica funzione nativa che PHP mette a disposizione per questo scopo. In realtà il linguaggio fornisce più soluzioni per la stessa procedura:

mysql_fetch_row(): restituisce l'array corrispondente ad una riga caricata, diversamente restituisce FALSE nel caso in cui non siano disponibili sono delle righe; questa funzione carica una riga di dati a partire dal risultato associato ad un determinato identificativo, essa viene restituita sotto forma di array e ciascuna colonna del risultato viene archiviata all'interno di un indice appartenente al vettore, a partire dall'indice "0".
mysql_fetch_array(): restituisce un array corrispondente alla riga caricata, diversamente restituisce FALSE nel caso in cui non ci siano più righe; questa funzione si configura come una versione in forma estesa della funzione mysql_fetch_row() precedentemente descritta; mysql_fetch_array(), oltre ad archiviare i risultati di un'interrogazione all'interno di un vettore dotato di indice numerico, li associa a degli indici associativi utilizzando i nomi dei campi quali chiavi; nel caso in cui due o più colonne di uno specifico risultato presentino gli stessi nomi per i campi, avrà la precedenza l'ultima colonna coinvolta dalla query.
mysql_fetch_assoc(): restituisce un array associativo corrispondente alla riga caricata, diversamente restituisce FALSE nel caso in cui non ci siano più righe. L'utilizzo di mysql_fetch_assoc() equivale ad una chiamata per mysql_fetch_array() introducendo MYSQL_ASSOC come secondo parametro dove il primo è l'interrogazione al DBMS passata a mysql_query(). La funzione ha unicamente il compito di restituire un array associativo, quindi, quando è necessario avvalersi di un indice numerico invece che di quello associativo, è necessario utilizzare mysql_fetch_array(); nel caso in cui due o più colonne di uno specifico risultato presentino gli stessi nomi per i campi, avrà la precedenza l'ultima colonna coinvolta dalla query.
mysql_fetch_object(): restituisce un oggetto dotato di proprietà corrispondenti alla riga caricata, diversamente restituisce FALSE nel caso in cui non vi siano più righe. La funzione utilizza un meccanismo simile a quello di mysql_fetch_array(), ma in questo caso viene restituito in output un oggetto e non un vettore; quindi, mysql_fetch_object() permette l'accesso ai dati tramite i nomi dei campi e non utilizzando il loro indice, in quanto i numeri non sono accettabili come nomi per le proprietà.

Tutte le funzioni elencate accettano come parametro una query di selezione passata alla funzione mysql_query() e potranno essere scelte dall'utilizzatore sulla base delle necessità dell'applicazione corrente; una query di selezione si basa sul comando Sql SELECT a cui devono seguire i nomi dei campi coinvolti nell'estrazione separati da una virgola, o in alternativa il carattere jolly asterisco (*) che indica tutti i campi presenti nella tabella il cui nome è introdotto dalla clausola FROM; una query SELECT può coinvolgere tutti i record presenti in una tabella così come soltanto alcuni o uno solo di essi sulla base di una specifica condizione, questa condizione può essere per esempio un valore relativo ad un campo.

Nel caso del piccolo blog engine descritto in questa trattazione verrà utilizzata mysql_fetch_object() all'interno di una funzione personalizzata chiamata estrai() interna alla classe MysqlClass;

```
// funzione per l'estrazione dei record
public function estrai($risultato)
 {
  if(isset($this->attiva))
  {
  $r = mysql_fetch_object($risultato);
  return $r;
  }else{
  return false;
  }
 }
```

Anche in questo caso la funzione controlla innanzitutto che esista una connessione attiva a MySQL, se questa è presente, il risultato della query SELECT passata alla funzione mysql_query() sarà utilizzato come argomento per mysql_fetch_object() producendo il valore di ritorno; se invece non dovesse essere attiva alcuna connessione, la funzione si limiterà a restituire FALSE.

L'Autenticazione

La procedura di autenticazione per il back-end del piccolo blog engine d'esempio si articolerà in tre fasi:

- inserimento in un modulo dei dati richiesti per l'autenticazione (username e password);
- confronto tra i dati inviati dal modulo e il contenuto della tabella richiamata in query;
- esito del confronto: in caso positivo (i dati inviati tramite il modulo corrispondono a quelli memorizzati in tabella) si avrà accesso all'amministrazione, diversamente sarà eseguita una procedura di rindirizzamento verso la homepage del sito.

Di seguito viene presentato il codice necessario per il login, compreso di form per l'invio dei dati di autenticazione, i commenti aiuteranno nella comprensione dei diversi passaggi ma verranno fornite successivamente ulteriori informazioni sul suo funzionamento:

```
<?php
// inizializzazione della sessione
session_start();
// se la sessione di autenticazione
// è già impostata non sarà necessario effettuare il login
// e il browser verrà reindirizzato alla pagina di scrittura dei post
if (isset($_SESSION['login']))
{
 // reindirizzamento alla homepage in caso di login mancato
 header("Location: gestisci.php");
}
// controllo sul parametro d'invio
if(isset($_POST['submit']) && (trim($_POST['submit']) == "Login"))
```

```
{
  // controllo sui parametri di autenticazione inviati
  if( !isset($_POST['username']) || $_POST['username']=="" )
  {
    echo "Attenzione, inserire la username.";
  }
  elseif( !isset($_POST['password']) || $_POST['password'] =="")
  {
    echo "Attenzione, inserire la password.";
  }else{
    // validazione dei parametri tramite filtro per le stringhe
    $username = trim(filter_var($_POST['username'], FILTER_SANITIZE_STRING));
    $password = trim(filter_var($_POST['password'], FILTER_SANITIZE_STRING));
    $password = sha1($password);
    // inclusione del file della classe
    include "funzioni_mysql.php";
    // istanza della classe
    $data = new MysqlClass();
    // chiamata alla funzione di connessione
    $data->connetti();
    // interrogazione della tabella
    $auth = $data->query("SELECT id_login FROM login WHERE username_login = '$username'
AND password_login = '$password'");
    // controllo sul risultato dell'interrogazione
        if(mysql_num_rows($auth)==0)
    {
        // reindirizzamento alla homepage in caso di insuccesso
          header("Location: index.php");
    }else{
          // chiamata alla funzione per l'estrazione dei dati
      $res =  $data->estrai($auth);
          // creazione del valore di sessione
      $_SESSION['login'] = $res-> id_login;
          // disconnessione da MySQL
          $data->disconnetti();
        // reindirizzamento alla pagina di amministrazione in caso di successo
          header("Location: gestisci.php");
    }
  }
}else{
  // form per l'autenticazione
  ?>
<h1>Accesso all'amministrazione:</h1>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
Username:<br />
<input name="username" type="text"><br />
Password:<br />
<input name="password" type="password" size="20"><br />
<input name="submit" type="submit" value="Login">
</form>
```

```
  <?
}
?>
```

Commentando il funzionamento del codice proposto è possibile fornire la seguente descrizione:

viene lanciata una sessione che sarà aperta solo nel caso in cui il processo di autenticazione dovesse avere successo;
viene effettuato un controllo sull'invio dei dati tramite modulo, se uno dei due campi di input non dovesse essere formulato si riceverà una notifica relativa all'omissione;
nel caso siano stati inviati tutti e due i parametri richiesti, questi verranno passati alla funzione filter_var() che grazie all'argomento FILTER_SANITIZE_STRING permetterà di filtrare gli input evitando stringhe potenzialmente pericolose per l'applicazione e i dati da essa gestiti;
la query di selezione consiste nell'estrazione del valore relativo al campo denominato "username_login" (SELECT username_login) dove i parametri di input sono identici ai valori memorizzati nei campi "username_login" e "password_login" (WHERE username_login = '$username' AND password_login = '$password'); l'operatore di confronto AND consente di introdurre un nuovo criterio di selezione oltre quello già proposto tramite WHERE, essa restituisce TRUE soltanto nel caso in cui entrambi i confronti effettuati producano un risultato positivo.
mysql_num_rows() è una funzione nativa di PHP che ha il compito di restituire il numero di righe coinvolte da un risultato per le istruzioni SELECT, accetta come parametro l'esito della query passata a mysql_query() e può essere utilizzata per procedure di controllo; nel caso del codice proposto, per esempio, il numero di righe coinvolte dall'interrogazione non potrà essere uguale a "0", diversamente l'esito del confronto tra gli input inviati e il contenuto della tabella sarebbe negativo e l'autenticazione non avrebbe luogo.
Se il controllo effettuato da mysql_num_rows() dovesse restituire "0" allora il login fallirebbe e il browser dell'utente subirà una rindirizzamento alla homepage del blog.
Nel caso di esito positivo verrà invece richiamata la funzione estrai() che estrarrà dalla tabella il valore relativo al campo "id_login" e lo utilizzerà per creare un valore di sessione che permetterà di accedere alle altre pagine dell'amministrazione.
Il successo della procedura di login determinerà una rindirizzamento alla pagina gestisci.php in cui sarà presente il modulo per l'inserimento dei post.

Inserimento dei post

La pagina destinata all'inserimento dei post è il cuore dell'applicazione blog engine, consente infatti di scrivere gli articoli che verranno pubblicati sul sito Web e che eventualmente verranno commentati dagli utenti che frequentano il blog.

Si tratta di una pagina riservata, infatti ad essa potranno accedere soltanto gli utenti autenticati tramite la procedura di login descritta in precedenza, traducendo tecnicamente quanto appena detto, per poter accedere alla pagina di scrittura dei post sarà necessario che esista una sessione attiva, altrimenti il browser verrà reindirizzato verso la prima pagina del front-end.

Di seguito viene presentato il codice necessario per l'inserimenti degli articoli:

```
<?php
// inizializzazione della sessione
```

```
session_start();
// controllo sul valore di sessione
if (!isset($_SESSION['login']))
{
 // reindirizzamento alla homepage in caso di login mancato
 header("Location: index.php");
}

// valorizzazione delle variabili con i parametri dal form
if(isset($_POST['submit'])&&($_POST['submit']=="Scrivi")){

  if(isset($_POST['autore'])){
    $autore = addslashes(filter_var($_POST['autore'], FILTER_SANITIZE_STRING));
  }
  if(isset($_POST['titolo'])){
    $titolo = addslashes(filter_var($_POST['titolo'], FILTER_SANITIZE_STRING));
  }
  if(isset($_POST['testo'])){
    $testo = addslashes(filter_var($_POST['testo'], FILTER_SANITIZE_STRING));
  }

  // inclusione del file della classe
include "funzioni_mysql.php";
// istanza della classe
$data = new MysqlClass();
// chiamata alla funzione di connessione
$data->connetti();

    $t = "post"; # nome della tabella
    $v = array ($titolo,$testo,$autore,date("Y-m-d")); # valori da inserire
    $r =  "titolo_post,testo_post,autore_post,data_post"; # campi da popolare

   // chiamata alla funzione per l'inserimento dei dati
    $data->inserisci($t,$v,$r);
    echo "Articolo inserito con successo.";
    // disconnessione
    $data->disconnetti();
   }else{
  // form per l'inserimento
  ?>
<h1>Inserimento post:</h1>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
Titolo:<br>
<input name="titolo" type="text"><br />
Testo:<br>
<textarea name="testo" cols="30" rows="10"></textarea><br />
Autore:<br>
<input name="autore" type="text"><br />
<input name="submit" type="submit" value="Scrivi">
</form>
```

```
  <?
}
?>
```

Il meccanismo che consente la pubblicazione dei post è molto semplice:

se sono stati inviati i parametri di input (titolo, testo e autore del post) viene lanciata la fase di inserimento dei valori nella tabella, diversamente verrà visualizzato il form necessario per la scrittura dell'articolo;
i dati inviati vengono trasformati in variabili che contengono i valori da inserire in tabella;
oltre ai campi titolo, testo e autore è necessario popolare anche il campo relativo alla data di stesura, quest'ultimo è caratterizzato dal tipo di dato DATE che prevede un formato per le date "anno-mese-giorno", quindi per creare il valore da inserire viene richiamata la funzione date() di PHP che registra la data corrente sulla base dei parametri passati come argomento, nel caso specifico vengono passati come parametri "Y" (anno in quattro cifre), "m" (numero del mese preceduto da "0" se formato da una sola cifra, ad esempio "09" per settembre) e "d" (giorno del mese preceduto da "0" se formato da una sola cifra, ad esempio "01" per il primo giorno del mese).
Le variabili relative ai parametri inviati vengono passate alla funzione inserisci() per la popolazione della tabella, una volta eseguita la query di INSERT viene chiusa la connessione al DBMS in modo da liberare risorse per il sistema

La formattazione dei post

Una volta inseriti tramite l'area di amministrazione gli articoli che si desidera mostrare nel front-end, è possibile procedere con il codice necessario per la loro visualizzazione in homepage; tipicamente nella prima pagina di un blog non vengono mostrati gli articoli per esteso ma soltanto delle anteprime composte dalle prime parole dei post.

A questo scopo verrà proposta una piccola funzione che permetterà di "tagliare" il testo degli articoli ad un'altezza definita attraverso una cifra che rappresenterà il numero di parole che verranno mostrate per ogni testo.

Il suo funzionamento è abbastanza semplice, essa accetta tre parametri:

il testo da tagliare;
il numero di parole da presentare in anteprima;
un elemento da mostrare in coda all'anteprima (nel caso specifico sarà un collegamento al testo completo del post).

La funzione "esplode" una stringa e quindi anche un testo sulla base degli spazi in esso presenti, utilizzati come delimitatori per identificare le diverse parole; una volta contato il numero delle parole che compongono il testo, questo viene tagliato subito dopo la parola che corrisponde alla cifra indicata come parametro della funzione preview(), se per esempio $offset = 50, allora il testo verrà tagliato dopo la cinquantesima parola.

L'ultimo parametro ($collegamento) potrà essere definito arbitrariamente a seconda dell'elemento che si desidera visualizzare in code all'anteprima.

Di seguito è possibile analizzare il codice della funzione per la creazione delle anteprime:

```
// funzione per la creazione di anteprime dei testi
public function preview($post, $offset, $collegamento) {
 return (count($anteprima = explode(" ", $post)) > $offset) ? implode(" ", array_slice($anteprima, 0, $offset)) . $collegamento : $post;
}
```

Prima di passare al codice per la visualizzazione dei post in homepage, è utile proporre anche una seconda funzione, si ricordi infatti che ad ogni post è associato un valore relativo alla data che utilizza il formato "aaaa-mm-dd", se si desidera riformattare la data in modo da utilizzare la disposizione consueta nei paesi mediterranei, "gg-mm-aaaa", sarà possibile creare una piccola funzione che suddivida la data nei tre diversi componenti e li riunisca nell'ordine desiderato:

```
// funzione per la formattazione della data
public function format_data($d)
{
  $vet = explode("-", $d);
  $df = $vet[2]."-".$vet[1]."-".$vet[0];
  return $df;
}
```

La funzione "vede" la data registrata nel campo "data_post" della tabella "post" come una stringa che "esplode" sulla base del carattere di delimitazione "-", fatto questo la suddivide in tre elementi che verranno risistemati in ordine inverso rispetto a quello originale per ottenere il valore desiderato.

In alternativa possiamo utilizzare le funzioni strtotime e strftime per la formattazione della data. Vediamo come:

```
// funzione per la formattazione della data
public function format_data($d)
{
  // converte la data in timestamp
  $vet = strtotime($d);
  // converte il timestamp della variabile $vet
  // in data formattata
  $df = strftime('%d-%m-%Y', $vet);
  return $df;
}
```

Visualizzazione dei post nella homepage

Per la visualizzazione dei post in homepage non sarà necessario creare funzioni aggiuntive; nella prima pagina del blog verranno visualizzati i dati dei post scritti fino al momento corrente compresa un'anteprima del testo completo relativo ad ogni articolo, per comodità del lettore ogni anteprima sarà seguita dal classico collegamento alla pagina che permetterà di leggere il testo integrale del post, in questo modo si eviterà di generare un'homepage troppo lunga da scorrere e gli utenti

potranno avere informazioni sufficienti per decidere se proseguire o meno con la lettura degli articoli.

Di seguito viene proposto il codice necessario per la visualizzazione dei post in homepage:

```
<html>
<head>
<title>MioBlog</title>
</head>
<body>
<h1>MioBlog: realizzato in PHP e MySQL</h1>
<?php
// inclusione del file di classe
include "funzioni_mysql.php";
// istanza della classe
$data = new MysqlClass();
// chiamata alla funzione di connessione
$data->connetti();
// query per l'estrazione dei record
$post_sql = $data->query("SELECT * FROM post ORDER BY data_post DESC");
// controllo sul numero di records presenti in tabella
if(mysql_num_rows($post_sql) > 0){
  // estrazione dei record tramite ciclo
  while($post_obj = $data->estrai($post_sql)){
    $id_post = $post_obj->id_post;
    $titolo_post = stripslashes($post_obj->titolo_post);
    $testo_post = stripslashes($post_obj->testo_post);
    $autore_post = stripslashes($post_obj->autore_post);
    $data_post = $post_obj->data_post;

    // visualizzazione dei dati
    echo "<h2>".$titolo_post."</h2>\n";
    echo  "Autore <b>". $autore_post . "</b>\n";
    echo  "<br />\n";
    echo  "Pubblicato il <b>" . $data->format_data($data_post) . "</b>\n";
    echo  "<br />\n";
       // collegamento al testo completo del post
    $leggi_tutto = "<br /><a href=\"post.php?id_post=$id_post\">Articolo completo</a>\n";
       // anteprima del testo

       echo "<p>".$data->preview($testo_post, 50, $leggi_tutto)."</p>\n";
    echo "<hr>\n";
  }
}else{
  // notifica in assenza di record in tabella
  echo "Per il momento non sono disponibili post.";
}
// chiusura della connessione a MySQL
$data->disconnetti();
?>
```

```
</body>
</html>
```

Per ottenere l'output desiderato, è stata passata come argomento alla funzione personalizzata query() l'istruzione Sql SELECT * FROM post ORDER BY data_post DESC, che "tradotta" in linguaggio umano significa: "seleziona tutti i record presenti nella tabella post e ordinali sulla base dei valori contenuti nel campo data_post disposti in ordine discendente, dall'ultimo al primo per data di scrittura". DESC infatti è una delle due clausole che è possibile associare al comando ORDER BY di SQL e consente di visualizzare dei valori partendo dal maggiore (o nel caso specifico, dal più recente) fino ad estrarre come ultimo record quello corrispondente al valore minore; l'ordinamento contrario, cioè a partire dal valore inferiore, è ottenibile sostituendo DESC con ASC.

La stessa query avrebbe potuto coinvolgere soltanto uno specifico numero di record, ad esempio soltanto quelli corrispondenti agli ultimi dieci post pubblicati, utilizzando una particolare clausola chiamata LIMIT, in questo caso l'istruzione sarebbe stata la seguente: SELECT * FROM post ORDER BY data_post DESC LIMIT 10; ma sarebbe stato anche possibile estrarre soltanto i record validi per un determinato intervallo di valori, ad esempio, l'istruzione SELECT * FROM post ORDER BY data_post DESC LIMIT 5,10; avrebbe consentito la visualizzazione dei record che disposti in senso cronologico crescente avessero occupato le posizioni ricompresse tra la quinta e la decima.

Si noti inoltre, come il risultato dell'applicazione della funzione query() sull'istruzione SQL sia stato si passato alla funzione estrai() quale condizione di un ciclo while, ciò è stato necessario perché i record contenuti all'interno della tabella possono essere più di uno, quindi ognuno di essi dovrà essere visualizzato tramite una diversa iterazione del ciclo fino a quando non venga più soddisfatta la condizione prevista per esso e tutti i risultati non siano stati mostrati a video.

Visualizzazione di un singolo post

L'output della homepage del blog d'esempio prevede la creazione di una lista di anteprime dei testi relativi ai post; in coda ad ogni anteprima vi è un link associato alla descrizione "Articolo completo" che funge da collegamento alla pagina che presenta il testo integrale del post selezionato; ciò è possibile perché il link prevede una querystring all'interno della quale è stato inserito l'identificativo univoco del post in anteprima (post.php?id_post=$id_post), ad esempio: post.php?id_post=1 permetterà di visualizzare nella pagina post.php il testo integrale del post che in tabella ha come identificativo univoco il valore "10", per cui nel record il valore inserito nel campo id_post sarà appunto "10".

Di seguito viene proposto il codice necessario per la visualizzazione dei singoli post con tanto di testo dell'articolo riprodotto integralmente:

```
<?php
// controllo sulla variabile inviata per querystring
if( (!isset($_GET['id_post'])) || (!is_numeric($_GET['id_post'])) )
{
// reindirizzamento del browser nel caso in cui la variabile non venga validata
header("Location: index.php");
}else{
```

```
$id_post = $_GET['id_post'];
}
?>
<html>
<head>
<title>MioBlog</title>
</head>
<body>
<?php
// inclusione del file di classe
include "funzioni_mysql.php";
// istanza della classe
$data = new MysqlClass();
// chiamata alla funzione di connessione
$data->connetti();
// query per l'estrazione dei record
$post_sql = $data->query("SELECT * FROM post WHERE id_post = $id_post");
// controllo sulla presenza in tabella del record corrispondente dell'id richiesto
if(mysql_num_rows($post_sql) > 0){
  // estrazione dei record
    $post_obj = $data->estrai($post_sql);
    $id_post = $post_obj->id_post;
    $titolo_post = stripslashes($post_obj->titolo_post);
        $testo_post = stripslashes($post_obj->testo_post);
    $autore_post = stripslashes($post_obj->autore_post);
    $data_post = $post_obj->data_post;

    // visualizzazione dei dati
    echo "<h1>".$titolo_post."</h1>\n";
    echo  "Autore <b>". $autore_post . "</b>\n";
        echo  "<br />\n";
    echo  "Pubblicato il <b>" . $data->format_data($data_post) . "</b>\n";
        echo  "<br />\n";
        echo "<p>".$testo_post."</p>\n";
        echo " :: <a href=\"commenti.php?id_post=$id_post\">Inserisci un commento</a>\n";
        echo  "<br />\n";
}else{
  // notifica in assenza di record
  echo "Non esiste alcun post per questo id.";
}
// chiusura della connessione a MySQL
$data->disconnetti();
?>
</body>
</html>
```

La pagina controlla che il parametro di input inviato in querystring sia valido, esso infatti deve essere impostato (cioè deve essere definito e quindi "esistere") e deve essere numerico, altrimenti non verrà ritenuto valido e il browser subirà un rindirizzamento alla homepage; naturalmente PHP mette a disposizione strumenti più raffinati per la validazione delle querystring e dei loro contenuti,

il discorso però allontanerebbe dal tema centrale di questa trattazione e nella sezione PHP di HTML.it sono già presenti numerosi articoli e guide in materia.

Il parametro inviato per querystring verrà associato come valore ad una variabile che né conserverà l'informazione e verrà utilizzata all'interno dell'istruzione necessaria per l'interrogazione Sql SELECT * FROM post WHERE id_post = $id_post; $id_post rappresenterà qualsiasi valore numerico presente in tabella, se per esempio alla variabile dovesse corrispondere il valore "10", allora l'istruzione che verrà lanciata al DBMS sarà la seguente SELECT * FROM post WHERE id_post = 10.

La funzione mysql_num_row() permetterà di controllare che nella tabella sia presente un record con identificativo univoco di valore pari a quello della variabile $id_post e, solo in caso di esito positivo, verrà richiamata la funzione estrai() per la visualizzazione dei valori contenuti nel record selezionato.

In questo caso il risultato di estrai() non dovrà essere utilizzato come condizione di un ciclo while in quanto l'identificativo è, appunto, univoco, quindi ad esso può corrispondere soltanto un unico record.

Il modulo per l'inserimento dei commenti

I post nei blog non sono soltanto degli articoli informativi, ma anche degli spunti per i commenti e le discussioni degli utenti che in questo modo possono partecipare alla vita del sito Web; il meccanismo che sta alla base della parte di applicazione che presiede alla scrittura dei commenti è molto simile a quello già analizzato per la stesura dei post, con una importante differenza, infatti i commenti saranno visibili nel front-end in corrispondenza dei diversi articoli soltanto dopo essere stati "moderati" e quindi approvati dall'amministratore; il campo "approvato" della tabella "commenti", associato ad un tipo di dato ENUM, è stato creato appositamente per questo scopo in quanto accetta soltanto due valori: "0", che è poi il valore predefinito e "1".

Quando un utente posta un commento, il campo "approvato" viene automaticamente impostato su "0", a questo punto il commento sarà stato memorizzato nel database ma non sarà visibile in corrispondenza del relativo articolo; come si vedrà in seguito, infatti, la query SELECT dei commenti prevede di estrarre soltanto i commenti che hanno approvato = 1.

Per modificare il valore di "approvato" da "0" a "1" sarà necessario implementare l'applicazione in amministrazione aggiungendo una parte dedicata alla "moderazione dei commenti" che verrà analizzata a breve, per ora invece verrà mostrato il codice necessario al loro inserimento:

```
<?php
// inclusione del file di classe
include "funzioni_mysql.php";
// istanza della classe
$data = new MysqlClass();
// chiamata alla funzione di connessione
$data->connetti();

// valorizzazione delle variabili con i parametri dal form
```

```
if(isset($_POST['submit'])){
  if(!isset($_POST['autore']) || !isset($_POST['commento']) || !isset($_POST['post_id']) || !
is_numeric($_POST['post_id']))
  {
   echo "Tutti i campi sono obbligatori";
  }else{
    $autore = htmlentities(addslashes($_POST['autore']));
    $post_id = $_POST['post_id'];
    $commento = htmlentities(addslashes($_POST['commento']));

    $t = "commenti"; # nome della tabella
    $v = array ($post_id,$autore,$commento,date("Y-m-d")); # valori da inserire
    $r =  "id_post,autore_commento,testo_commento,data_commento"; # campi da popolare

   // chiamata alla funzione per l'inserimento dei dati
    $data->inserisci($t,$v,$r);
    header("Location: post.php?id_post=$post_id");
 }
}else{
  // controllo sull'id del post inviato per querystring
  if( isset($_GET['id_post']) && is_numeric($_GET['id_post']) ){
    $id_post = $_GET['id_post'];
    $sql_commenti = $data->query("SELECT id_post FROM post WHERE id_post='$id_post'");
    if(mysql_num_rows($sql_commenti) > 0){
      // viene visualizzato il form solo nel caso in cui l'unico dato inviato sia l'id del post
      ?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
Autore:<br />
<input name="autore" type="text"><br />
Commento:<br />
<textarea name="commento" cols="30" rows="10"></textarea><br />
<input name="post_id" type="hidden" value="<?php echo $id_post; ?>">
<input name="submit" type="submit" value="Invia">
</form>
      <?php
      // notifiche in caso di querystring vuota o non valida
    }else{
      echo "Non è possibile accedere alla pagina da questo percorso.";
    }
  }else{
    echo "Commenti non consentiti, articolo inesistente.";
  }
}
// disconnessione
$data->disconnetti();
?>
</body>
</html>
```

È possibile postare un commento a partire dalla homepage o da un singolo articolo, facendo clic sul

collegamento chiamato Inserisci un commento; il click sul link produrrà una querystring che conserva al suo interno l'informazione relativa all'identificativo univoco dell'articolo che si desidera commentare, questo dato è fondamentale perché permette di popolare il campo post_id della tabella commenti per mettere in relazione post e commenti.

L'inserimento dei dati in tabella avviene grazie all'invio di due parametri (autore e commento) da valorizzare tramite un modulo che conserva come dato nascosto (hidden) anche l'identificativo univoco del post di riferimento; una volta compilato il modulo, l'effettivo invio dei parametri verrà controllato e, se l'esito dovesse essere positivo, questi verranno filtrati e utilizzati per definire delle variabili che andranno a popolare i campi della tabella commenti creando un nuovo record grazie alla funzione inserisci(); il campo denominato approvato relativo al nuovo record avrà associato il valore predefinito "0" e il commento non sarà visibile prima che questo non sia modificato in "1".

Si noti come il valore relativo al campo ENUM approvato sia posto tra singoli apici, apparentemente infatti sembrerebbe associato ad un dato numerico e invece si tratta di un valore che il DBMS vede come un carattere, quindi, rispettando la sintassi SQL prevista per MySQL esso non dovrà essere inserito in istruzione senza apici, come avverrebbe per i valori numerici, ma delimitato da essi.

Moderazione dei commenti

Per moderare i commenti postati dagli utenti verranno creati due file, uno destinato alla visualizzazione dei commenti non ancora moderati (lista_commenti.php), un altro con il compito di approvare tramite una query di UPDATE sul campo approvato il commento selezionato (moderazione.php). Entrambi i file saranno inseriti all'interno dell'area di amministrazione e il loro accesso sarà riservato soltanto agli utenti autenticati.

Il file che produce la lista dei commenti si basa su una semplice query SELECT che permetterà di estrarre tutti i record dalla tabella commenti in cui approvato = '0' (SELECT id_commento, testo_commento, autore_commento, data_commento FROM commenti WHERE approvato = '0') e ordinandoli in senso cronologico decrescente (ORDER BY data_commento DESC), cioè a partire dall'ultimo commento postato:

```
<?php
// inizializzazione della sessione
session_start();
// controllo sul valore di sessione
if (!isset($_SESSION['login']))
{
 // reindirizzamento alla homepage in caso di login mancato
 header("Location: index.php");
}
// inclusione del file di classe
include "funzioni_mysql.php";
// istanza della classe
$data = new MysqlClass();
// chiamata alla funzione di connessione
$data->connetti();
// query per l'estrazione dei record
```

```
$commento_sql=$data->query("SELECT
id_commento,testo_commento,autore_commento,data_commento FROM commenti WHERE
approvato='0' ORDER BY data_commento DESC");

echo "<h1>Elenco dei commenti da approvare</h1>\n";
// controllo sul numero di record presenti in tabella
if(mysql_num_rows($commento_sql) > 0){
  echo "<ul>\n";
  // estrazione dei record tramite ciclo
  while($commento_obj = $data->estrai($commento_sql)){
    $id_commento = $commento_obj->id_commento;
       $testo_commento = stripslashes($commento_obj->testo_commento);
    $autore_commento = stripslashes($commento_obj->autore_commento);
    $data_commento = $commento_obj->data_commento;

    // visualizzazione dei dati
        echo "<li>\n";
        echo "Autore: " . $autore_commento . " Scritto il ". $data->format_data($data_commento) .
"\n";
     echo  "<br />\n";
        echo "Commento: " . $testo_commento;
     echo  "<br />\n";
        echo " :: <a href=\"moderazione.php?id_commento=$id_commento\">approva</a>\n";
        echo "</li>\n";
  }
  echo "</ul>\n";
}else{
  // notifica in assenza di record che soddisfino le caratteristiche richieste
  echo "Per il momento non sono disponibili commenti da approvare.";
}
// chiusura della connessione a MySQL
$data->disconnetti();
?>
```

Ad ogni commento elencato corrisponderà un link chiamato approva e destinato a produrre una querystring che trasmetta alla pagina moderazione.php l'identificativo univoco del commento selezionato:

```
<?php
// inizializzazione della sessione
session_start();
// controllo sul valore di sessione
if (!isset($_SESSION['login']))
{
 // reindirizzamento alla homepage in caso di login mancato
 header("Location: index.php");
}

// controllo sull'id del commento inviato per querystring
if (isset($_GET['id_commento']) && is_numeric($_GET['id_commento']))
```

```
{
$id_commento = $_GET['id_commento'];
?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
  <h1>Attenzione!</h1>
  Si sta per approvare il commento selezionato.<br />
  Premere il pulsante per eseguire l'operazione richiesta.<br />
  <br>
  <input name="commento_id" type="hidden" value="<?php echo $id_commento; ?>">
  <input name="submit" type="submit" value="Modera">
</form>
<?php
}
// controllo sull'id del commento inviato per form
elseif(isset($_POST['commento_id']) && is_numeric($_POST['commento_id']))
{
 $commento_id = $_POST['commento_id'];
 // inclusione del file di classe
 include "funzioni_mysql.php";
 // istanza della classe
 $data = new MysqlClass();
 // chiamata alla funzione di connessione
 $data->connetti();
 $data->query("UPDATE commenti SET approvato='1' WHERE id_commento = $commento_id");
 // reindirizzamento alla pagina di gestione dei commenti
 header("Location: lista_commenti.php");
 // chiusura della connessione a MySQL
 $data->disconnetti();
}
?>
```

L'identificativo univoco inviato alla pagina per la moderazione è indispensabile per identificare il record da aggiornare all'interno della query per la modifica; come anticipato, questa si basa sul comando SQL UPDATE a cui deve seguire il nome della tabella che contiene il record e il nome del campo da aggiornare con il nuovo valore introdotti dalla chiave SET (UPDATE commenti SET approvato = '1'); per sapere a carico di quale record dovrà essere effettuato l'update, l'istruzione dovrà prevedere anche la clausola WHERE che specificherà il valore dell'identificativo univoco corrispondente al record da modificare (WHERE id_commento = $commento_id).

Il codice presente nel file per la moderazione potrà essere modificato facilmente anche per effettuare l'eliminazione di un commento, infatti l'unica operazione da fare sarà quella di sostituire la query basata sul comando UPDATE con un'altra basata sul comando DELETE; in questo caso infatti si dovrà utilizzare l'istruzione:

```
$data->query("DELETE FROM commenti WHERE id_commento = $commento_id");
```

In una query per la cancellazione di un record è sufficiente passare in istruzione l'identificativo di una riga introdotto dalla clausola WHERE, in questo modo il DBMS potrà identificare con precisione il record da cancellare senza coinvolgere altri dati in questo processo potenzialmente pericoloso.

Conteggio e visualizzazione dei commenti

Nella homepage del blog, oltre alle anteprime dei post, sarà disponibile anche il risultato del conteggio dei commenti relativi ad ogni articolo; questo sarà possibile grazie ad una piccola funzione basata sull'istruzione Sql SELECT COUNT:

```
// funzione per il conteggio dei commenti
public function conta_commenti($id_c, $tbl, $campo, $id_post,$enum, $valore_enum)
 {
  if(isset($this->attiva))
  {
  $query_n_com = mysql_query("SELECT COUNT($id_c) AS n_commenti from $tbl WHERE
$campo = $id_post AND $enum = '$valore_enum'") or die (mysql_error());
  $obj_n_com = mysql_fetch_object($query_n_com) or die (mysql_error());
  return $obj_n_com->n_commenti;
  }else{
  return false;
  }
 }
```

Si analizzi il funzionamento del codice appena proposto: la funzione accetta come parametri 6 argomenti simboleggiati da altrettante variabili:

$id_c: il nome del campo utilizzato per il conteggio dei valori;
$tbl: il nome della tabella coinvolta nell'interrogazione;
$campo: il nome del campo che memorizza il valore che verrà utilizzato come condizione per la clausola WHERE;
$id_post: il valore del campo descritto in precedenza;
$enum: il nome del campo utilizzato l'operatore di confronto AND per specificare ulteriormente le caratteristiche dei record da contare;
$valore_enum: il valore che dovrà essere associato al campo descritto in precedenza.

Sulla base di questi parametri sarà possibile eseguire una chiamata alla funzione conta_commenti() direttamente dalla homepage del sito Web:

```
// parte relativa al conteggio dei commenti
echo "Commenti: " . $data->conta_commenti("id_commento", "commenti", "id_post", $id_post,
"approvato",'1');
echo " :: <a href=\"commenti.php?id_post=$id_post\">Inserisci un commento</a>\n";
```

Sarà quindi eseguita una query che conterà tutti i valori relativi al campo id_commento presenti nella tabella commenti in cui il valore di id_post sia uguale a quello dell'identificativo univoco del post mostrato in anteprima e in cui il campo ENUM approvato sia uguale a "1"; in questo modo saranno conteggiati soltanto i commenti approvati per ogni singolo post, la query che per esempio conterà i commenti approvati per il post che avrà associato l'identificativo univoco "2" sarà la seguente:

```
SELECT COUNT(id_commento) AS n_commenti from commenti WHERE id_post = 2 AND
```

approvato = '1';

n_commenti è un "alias", cioè un campo in realtà non presente all'interno di una tabella ma che viene creato temporaneamente per la generazione di un risultato desiderato; in questo caso l'alias conterrà il risultato relativo al conteggio dei record coinvolti dall'istruzione.

A questo punto, il discorso relativo alla gestione dei commenti può ritenersi concluso, manca soltanto la parte che concerne la visualizzazione dei commenti per ogni articolo il cui codice dovrà essere inserito nella pagina destinata a mostrare il testo integrale dei singoli post; ciò sarà possibile attraverso una semplice query di selezione:

```
// estrazione dei commenti
$post_commenti=$data->query("SELECT autore_commento,testo_commento,data_commento
FROM commenti WHERE id_post = $id_post AND approvato='1' ORDER BY data_commento
DESC");
   if(mysql_num_rows($post_commenti) > 0){
       echo "<ul>\n";
    while($commenti_obj = $data->estrai($post_commenti))
       {
       $autore_commento = stripslashes($commenti_obj->autore_commento);
       $testo_commento = stripslashes($commenti_obj->testo_commento);
       $data_commento = stripslashes($commenti_obj->data_commento);
       echo "<li>\n ";
       echo "Autore: " . $autore_commento . " Scritto il ". $data->format_data($data_commento) .
"\n";
    echo  "<br />\n";
       echo "Commento: " . $testo_commento;
       echo "</li>\n ";
       }
       echo "</ul>\n";
      }else{
      echo "Nessun commento per questo post";
      }
```

Il listato esposto esegue un'interrogazione sulla base dell'identificativo univoco relativo al post visualizzato: (SELECT … FROM commenti WHERE id_post = $id_post), essa prevede che siano mostrati soltanto i commenti approvati (AND approvato = '1') relativi al post corrente e che l'ordine per la loro visualizzazione sia impostato a partire dal commento più recente (ORDER BY data_commento DESC).

Un semplice motore di ricerca per il blog

In una buona applicazione per la gestione di un blog non può mancare un motore di ricerca interno per trovare velocemente i post sulla base di parole chiavi da confrontare con il contenuto della tabella dedicata ai post; il meccanismo che permetterà il funzionamento del semplicissimo motore di ricerca proposto in questa trattazione si basa su un una query SELECT che presenta qualche differenza rispetto a quelle analizzate in precedenza.

Si immagini per esempio di voler ricercare il termine "data" all'interno del contenuto dei post:

la chiave verrà passata all'applicazione per la ricerca sotto forma di parametro di input inviato tramite form;
l'applicazione effettuerà un controllo sulla validità del parametro di input e lo utilizzerà come valore da associare ad una variabile ($key nell'esempio);
a questo punto verrà effettuata una query SELECT sulla tabella dei post (SELECT * FROM post);
la query verrà effettuata confrontando la parola ricercata con il contenuto dei record relativi ai titoli e al testo dei post (WHERE (titolo_post LIKE '%" . $key . "%') OR (testo_post LIKE '%" . $key . "%') ;
sarà possibile ottenere un risultato valido non soltanto se la chiave sarà rilevata all'interno dei titoli e dei testi ma anche se verrà trovata in uno solo di questi termini di confronto, ciò è possibile grazie all'utilizzo dell'operatore OR al posto di AND, il primo infatti, a differenza del secondo, restituisce TRUE anche se soltanto uno degli argomenti di un confronto risulta vero;
i risultati della query verranno ordinati in senso decrescente sulla base dei valori contenuti nel campo data_post (ORDER BY data_post).

Di seguito viene proposto il codice necessario per il motore di ricerca:

```
<html>
<head>
<title>MioBlog</title>
</head>
<body>
<h1>Motore di ricerca</h1>
<?php
// inclusione del file di classe
include "funzioni_mysql.php";
// istanza della classe
$data = new MysqlClass();
// chiamata alla funzione di connessione
$data->connetti();
?>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
<input type="text" name="key" value="" />
<input type="submit" value="cerca" class="submit" />
</form>
<?php
if(isset($_POST['key'])&&($_POST['key']!="")&&(preg_match("/^[a-z0-9]+$/i", $_POST['key'])))
 {
$key = $_POST['key'];

$sql_cerca = $data->query("SELECT * FROM post WHERE (titolo_post LIKE '%" . $key . "%')
OR (testo_post LIKE '%" . $key . "%') ORDER BY data_post");
$trovati = mysql_num_rows($sql_cerca);
if($trovati > 0)
{
 echo "<p>Trovate $trovati voci per il termine <b>".stripslashes($key)."</b></p>\n";
 while($cerca_obj = $data->estrai($sql_cerca))
```

```
  {
   $id_post = $cerca_obj->id_post;
   $titolo_post = stripslashes($cerca_obj->titolo_post);
      $testo_post = stripslashes($cerca_obj->testo_post);
   $autore_post = stripslashes($cerca_obj->autore_post);
   $data_post = $cerca_obj->data_post;

   // visualizzazione dei dati
   echo "<h2>".$titolo_post."</h2>\n";
   echo  "Autore <b>". $autore_post . "</b>\n";
      echo  "<br />\n";
   echo  "Pubblicato il <b>" . $data->format_data($data_post) . "</b>\n";
      echo  "<br />\n";
      // link al testo completo del post
   $leggi_tutto = "<br /><a href=\"post.php?id_post=$id_post\">Articolo completo</a>\n";
      // anteprima del testo
      echo "<p>".$data->preview($testo_post, 50, $leggi_tutto)."</p>\n";
  }
  }else{
  // notifica in caso di mancanza di risultati
  echo "Al momento non sono stati pubblicati post che contengano questo termine.";
 }
}
// disconnessione
$data->disconnetti();
?>
```

LIKE è un operatore di confronto che permette di effettuare una comparazione tra campi simili anche se non uguali;

I due simboli percentuali (%), vengono utilizzati dal DBMS per rilevare se prima e dopo il termine ricercato vi siano altre parole; se invece di LIKE si fosse utilizzato l'operatore di identità (ad esempio: WHERE "titolo_post =' . $key . ' OR testo_post =' . $key . ') l'istruzione avrebbe cercato soltanto i titoli o i testi contenenti esclusivamente la chiave inviata in input.

Si noti come la chiave di ricerca, prima di partecipare alla query, sia stata validata tramite un'espressione regolare (preg_match("/^[a-z0-9]+$/i", $_POST['key']")), questo perché per ragioni di sicurezza nella parola ricercata non devono essere presenti caratteri speciali.