PHP/Versione stampabile

Wikibooks, manuali e libri di testo liberi.
< PHP

Indice

Copertina

Benvenuto nel wikibook:

PHP

Vai ai contenuti >>

Fase di sviluppo: (Sviluppo (http://it.wikibooks.org/wiki/Speciale:EspandiTemplate?wpInput=%7B%7BTemplate:Bollettino%7C1=PHP%7D%7D#Bollettino))

Introduzione

Molti si saranno chiesti cosa avesse realmente il PHP di diverso dagli altri linguaggi quali HTML, ASP ecc... La risposta non è così semplice come potrebbe sembrare...

Alcune caratteristiche che saltano all'occhio sono:

- **La dinamicità**: le pagine in HTML sono statiche e la possibilità di renderle dinamiche si riduce alle operazioni eseguibili in Javascript e al dhtml. Con il PHP, il Javascript serve molto meno in quanto il codice viene eseguito dal browser al caricamento della pagina, eventuali script Javascript servirebbero solo per cambiamenti in *real time* come le dissolvenze e le etichette. Per altre informazioni su questa innovazione vai alla sezione Come funziona
- **La semplicità**: il linguaggio PHP, a differenza dell'ASP, è molto più semplice da sviluppare, gestire e in seguito modificare. Non occorre infatti essere laureati per poter scrivere del codice complesso grazie a funzioni molto semplici e intuitive. Ve ne sarà data una dimostrazione in seguito.
- **La grandissima varietà di funzioni**: nel sito www.php.net (http://www.php.net) è visualizzabile la

documentazione completa del PHP con tutte le funzioni disponibili all'uso. Ce ne sono veramente tante, per appunto soddisfare ogni scopo si abbia in mente.
- **Il costo**: dato da non sottovalutare è il costo. Il php è Open Source, non ha limitazioni di sorta e permette ai sysadmin di compilare solo le parti realmente utili. Ha costi di manutenzione molto bassi e, ovviamente, non ha costi di licenza al contrario dell'asp.
- **La comunità**: PEAR è il più eclatante esempio di comunità di sviluppatori. Chiunque può scrivere una funzione o una classe php specifica e renderla disponibile a tutta la comunità grazie a PEAR che permette allo sviluppatore di ridurre il tempo utilizzando classi e/o funzioni già scritte.

Sommario della sezione

Introduzione/Storia

L'8 giugno 1995 il danese Rasmus Lerdorf invia un messaggio in un newsgroup (leggi il messaggio (http://groups.google.com/groups?selm=3r7pgp%24aa1%40ionews.io.org)) annunciando il rilascio di "un set di piccoli binari scritti in C", PHP 1.0 (che, all'inizio, significava "Personal Home Page", "Pagina Principale Personale"). Le funzioni di PHP 1.0 sono limitate: registra gli accessi ad un sito tracciando anche i referrers, può fare inclusioni server-side, mostra gli utenti connessi, protegge con password una pagina...

Nelle successive versioni vengono aggiunti il supporto per le query SQL in mSQL (predecessore di MySQL) e la disponibilità di un wrapper cgi (FI, "Form Interpreter", "Interprete di Form"). Intanto, verso la fine del '95, PHP comincia a diventare famoso e viene rinominato in PHP/FI, anche grazie alla possibilità di integrare PHP nelle pagine HTML

Il 12 novembre 1997 arriva PHP/FI 2.0 che, secondo il sito php.net (http://www.php.net), è usato da circa 50 000 domini.

PHP 3.0, rilasciato il 6 giugno 1998, segna un punto di svolta, in quanto appare lo "Zend Engine", creato dagli israeliani Zeev Suraski and Andi Gutmans. Oltre a questo vengono aggiunti il supporto per altri database e la compatibilità con Windows ed altri sistemi operativi. Cambia anche il nome che da "Personal Home Page" diventa l'attuale "PHP: Hypertext Preprocessor".

PHP 4.0 porta, il 22 maggio 2000, molte ottimizzazioni. Viene cambiata anche la licenza, che dalla GPL (addottata fin da PHP 1.0) passò alla PHP License (http://www.php.net/license/3_01.txt), più restrittiva ma sempre Open source.

Circa quattro anni dopo, il 13 luglio 2004, viene rilasciato PHP 5.0. Molti sono i miglioramenti proposti da questa versione; il principale è l'introduzione dello Zend Engine 2 e il supporto nativo alla programmazione a oggetti.

Nel 2005 la configurazione LAMP (Linux, Apache, MySql, PHP) supera il 50% del totale dei server sulla rete mondiale.

Introduzione/Come funziona

Fin dalla sua prima uscita, PHP è stato un linguaggio fortemente orientato al web. Tuttavia, i nuovi e più recenti miglioramenti lo rendono adatto ai più svariati scopi. Le tre principali aree di utilizzo di PHP sono:

- ***Server side scripting*** - *scripting* lato server per il web
- ***Shell scripting*** - *scripting* a riga di comando
- **Applicazioni desktop**

Server side scripting

Questo ambito di utilizzo è il più tradizionale ed il più diffuso. PHP consente di generare in maniera dinamica le pagine web. Per utilizzare PHP in questo ambito occorrono:

- un server web
- l'interprete PHP
- un browser web

Durante il caricamento di una pagina Web, il browser del client (cioè dell'utente) invia una richiesta HTTP al web server, il quale si incarica di restituirgli un file, normalmente "pagina" contenente codice HTML, oppure anche ad esempio un'immagine.
Nel caso sia una pagina scritta in HTML (solitamente indicata dall'estensione .htm o .html"), una volta ricevuta il browser è in grado di disegnarne il contenuto sullo schermo interpretando il linguaggio di markup.

Le pagine nelle quali è presente codice PHP, che sono memorizzate sul server, non sono direttamente lette ed interpretate dal browser ma vengono interpretate da un modulo aggiuntivo del web server che è appunto il modulo PHP.

Normalmente le pagine contenente codice PHP devono avere una estensione di tipo ".php" ma, configurando opportunamente il server, è possibile utilizzare anche estensione ".html" o altro.

Tutte le volte che al web server viene fatta la richiesta di una pagina, questa viene analizzata da esso. Se all'interno della pagina viene riconosciuta la presenza di codice PHP (delimitato da tags — marcatori — appositi) questa viene passata al modulo PHP che si preoccuperà di restituirla (in un certo senso di riscriverla) nel formato HTML, direttamente interpretabile dal browser richiedente.

Il susseguirsi logico delle varie fasi è il seguente:

1. l'utente richiama la pagina (inserendo l'URL o cliccando un link)
2. il browser inoltra la richiesta al web server
3. il web server cerca la pagina (il file) richiesto
4. se la pagina contiene codice PHP viene passata al modulo PHP, altrimenti si va al punto 6
5. il modulo PHP interpreta la pagina PHP e restituisce la corrispondente pagina HTML
6. la pagina "HTML" viene spedita al browser richiedente
7. il browser, una volta ricevuta la pagina, la legge e la disegna a monitor

Una pagina web è da considerare composta da due componenti fondamentali: la struttura o layout e il contenuto. Per layout intendiamo tutto ciò che descrive come la pagina deve essere disegnata, tabelle, colori, fonts, frames... in generale tutto ciò che può essere definito mediante il linguaggio HTML. Per contenuto consideriamo, per semplicità, tutto ciò che non è struttura ma informazione che la pagina ci offre. In un sito web di solito la struttura resta all'incirca la stessa per tutte le pagine. Quel che cambia è il contenuto.

Per facilitare il lavoro ai webmaster la soluzione sarebbe quella di poter separare "fisicamente" il contenuto delle pagine dalla loro struttura. Il PHP viene in aiuto soprattutto in tali situazioni: generalmente un pagina viene costruita memorizzando in un file la struttura (della generica pagina) e in un database il contenuto. In

questo modo quello che è il compito dell'interprete PHP è quello di assemblare la pagina inserendo il contenuto caricato dal database nella struttura.

Il funzionamento a questo punto differisce leggermente da quello sopra riportato in quanto il punto 5 si modifica in questo modo:

5. il modulo PHP interpreta la pagina PHP, richiede al database il contenuto da inserire, genera e restituisce la corrispondente pagina HTML.

In realtà, esistono anche altre possibilità: si può modificare il mime type con l'istruzione

```
<?php Header("Content-type: Linguaggio contenuto"); ?>
```

oppure tramite particolari estensioni di PHP come le librerie GD è possibile creare delle immagini e restituire quindi non una pagina HTML bensì un'immagine vera e propria.

Shell scripting

Per ***shell scripting*** si intende la scrittura ed esecuzione di script a riga di comando (in Windows, è il programma cmd.exe; in linux è chiamato anche Terminale o Console). Quando PHP lavora in questo ambito, non sono necessari né web server, né browser: basta lanciare dalla riga di comando l'interprete PHP.

Questo tipo di utilizzo è ideale, per esempio, per gli scripts che devono essere eseguiti periodicamente in maniera trasparente (con cron sui sistemi *nix oppure con *Task Scheduler* su Windows) oppure per quegli script che devono processare dei testi in automatico, senza generare particolari output.

*Per approfondire, vedi **PHP/Cli**.*

Applicazioni desktop

Grazie all'estensione **php-gtk2**, reperibile sul sito ufficiale [1] (http://gtk.php.net), adesso è possibile utilizzare PHP anche per la generazione di applicazioni desktop client side.

Anche se questo ambito di utilizzo non è il più diffuso ed è ancora abbastanza "instabile", può essere utile e divertente sviluppare applicazioni desktop con PHP e le librerie GTK, in quanto si tratta di strumenti di rapido utilizzo che generano programmi portabili fra più sistemi operativi.

*Per approfondire, vedi **PHP/Gtk**.*

Installazione e configurazione

Questa sezione spiega brevemente come installare e configurare PHP.

PHP è in grado di girare su una grande varietà di sistemi operativi, in congiunzione con molti web server. In questa sede, ci limiteremo a trattare l'installazione sui sistemi operativi Windows, Linux, BSD* e Mac OS X, considerando i due più diffusi web server: Apache http server, della Apache Software Foundation, e Internet Information Services della Microsoft.

Per installazioni su altri sistemi operativi ed altri web server, si rimanda alla documentazione ufficiale

(http://www.php.net/manual/en/) di PHP.

Installazione

Questa sezione spiega come si installa PHP nei vari sistemi operativi tra cui Windows, Mac OS e Linux.

Installazione/Linux

Sistemi Debian-Like

Nel caso di distribuzioni basate su Debian (come Debian stesso, Ubuntu, ecc) l'installazione di PHP può essere fatta tramite la console usando il comando `apt-get` oppure, tramite l'interfaccia grafica del gestore di pacchetti Synaptic.

Tramite apt-get

Per prima cosa è necessario avere un server web attivo, solitamente si usa Apache. Per installarlo digitiamo:

```
sudo apt-get install apache2
```

Ora è possibile procedere con l'installazione di PHP (l'asterisco deve essere sostituito con il numero della versione di PHP che si desidera installare sul proprio sistema, solitamente si sceglie l'ultima disponibile):

```
sudo apt-get install php*
sudo apt-get install libapache2-mod-php*
sudo /etc/init.d/apache2 restart
```

Per verificare l'avvenuta installazione di PHP si procede con:

```
sudo echo "<?php phpinfo() ?>" > /var/www/test.php
sudo chown www-data:www-data /var/www/test.php
```

Il primo comando crea una pagina di test contenente un piccolo script di prova (la cartella potrebbe essere diversa, per esempio `/srv/www`) e il secondo comando assegna la pagina di test al proprietario e al gruppo di default di Apache chiamato `www-data`. Se tutto funziona correttamente, visualizzando la pagina:

```
http://localhost/test.php
```

Sarà possibile vedere tutte le informazioni relative a PHP: Ciò significa che Apache e PHP funzionano correttamente.

Tramite Synaptic

Nel caso di distribuzioni come Ubuntu è possibile installare PHP utilizzando il Gestore di pacchetti Synaptic.

Non è consigliabile marcare i singoli pacchetti da installare poiché anche un utente esperto potrebbe dimenticarne qualcuno.

È più facile tenere a mente che si vuole realizzare un ambiente LAMP (Linux/Apache/MySQL/PHP) e procedere come segue:

1. Selezionare il menù **Modifica**;
2. Portarsi alla voce **Marca pacchetti per attività**;
3. Selezionare **LAMP Server**;
4. Premere **OK**.

Questa modalità consente di sfruttare gruppi di pacchetti preselezionati per attività specifiche. Quando viene selezionata un'attività, i pacchetti corrispondenti vengono marcati automaticamente per l'installazione. Premendo **Applica** si installeranno tutti i pacchetti selezionati.

Altri sistemi GNU/Linux

Da sorgente

L'installazione da sorgenti per GNU/Linux è del tutto equivalente alle operazioni effettuate per ogni altro pacchetto libero su un sistema Unix like qualsiasi, come per esempio i sistemi *BSD.

Per prima cosa ci si procura il sorgente che avrà un nome tipo: `php-x.y.z.tar.gz` dove x.y.z indicano i valori di versione e sottoversione. Una volta scaricati si scompattano con il comando:

```
tar -xvzf php-x.y.z.tar.gz
```

Poi si dovrà entrare nella cartella decompressa con:

```
cd php-x.y.z
```

Nella cartella dovrebbe essere presente uno script eseguibile di nome `configure`. Eseguirlo con:

```
./configure
```

Il comando dovrebbe accertarsi della presenza di tutte le librerie necessarie per l'esecuzione di PHP. Eventualmente è possibile abilitare a mano l'uso di alcune librerie. L'elenco delle opzioni è disponibile facendo:

```
./configure --help
```

Per esempio per abilitare l'uso della libreria `gd` e per l'installazione nella posizione standard `/usr` si effettuerà il comando:

```
./configure --with-gd --prefix=/usr
```

Altre opzioni comuni sono per esempio: `--with-mysql --enable-track-vars --with-pgsql`

Al termine del comando, se questo non ha rilevato errori, effettuare i classici comandi:

```
make
```

Seguito da:

```
make install
```

Quest'ultimo comando deve essere eseguito come amminitratore della macchina. Per diventarlo è necessario usare il comando:

```
su
```

o (per i sistemi *ubuntu):

```
sudo -s
```

Seguiti rispettivamente dalla password dell'utente root o dalla password dell'utente corrente che deve far parte del gruppo degli amministratori.

Installazione/Windows

L'installazione di PHP 5 su **Windows** è più complicata di quanto ci si aspetti su questa piattaforma. Il file di installazione fornito da php.net, infatti, potrebbe non funzionare oppure non configurare correttamente il sistema, quindi consigliamo di effettuare una sorta di installazione "manuale".

Per prima cosa dobbiamo avere, sul nostro sistema, un web server installato e funzionante. Php dà il meglio di sé in abbinamento con Apache web server, per la cui installazione rimandiamo al sito ufficiale http://httpd.apache.org/, ma funziona egregiamente anche con Internet information services, visto che stiamo parlando di ambiente Windows, e con altri web server.

Scarichiamo dal sito php.net l'ultima versione dei due archivi in formato .zip, contenenti i file binari per Windows, e li salviamo sul nostro disco rigido. Il file "php-5.x.x-Win32.zip" contiene i file e le dll necessari al funzionamento di php sul nostro sistema, il file "pecl-5.x.x-Win32.zip" contiene tutte le estensioni disponibili. Successivamente, decomprimiamo il primo archivio in una cartella sul nostro disco rigido(ad esempio C:\PHP) ed il secondo archivio nella sottocartella \ext (nel nostro esempio C:\PHP\ext).

A questo punto dobbiamo indicare a Windows che i file contenuti nella cartella C:\PHP devono diventare file di sistema, apriamo dal Pannello di Controllo l'opzione Sistema e scegliamo Avanzate/Variabili d'ambiente/Variabili di sistema. Facciamo doppio click sulla variabile PATH per modificarla e aggiungiamo a Valore variabile la stringa `;C:\PHP` (o il nome vostra cartella di decompressione), senza lasciare spazi tra il punto virgola e il testo già presente nella casella di testo. In questo modo Windows riconoscerà la cartella come di sistema.
In alternativa, possiamo copiare i file "php5ts.dll" e "php.ini" nella cartella C:\WINDOWS

Usando WAMP

Una buona soluzione per risolvere l'installazione di PHP 5 su Windows è quella di ricorrere al software "**WAMP**", acronimo per **W**indows **A**pache **M**ysql **P**HP.
La comodità di questo software consiste nel fatto che utilizzando l'*installer* scaricabile dal sito www.wampserver.com (http://www.wampserver.com/) si installano in una sola volta il web server Apache, il server e il client SQL MySQL ed infine l'ultima versione stabile di PHP.
Il tutto corredato di un programma che rimarrà attivo nella system tray di Windows e vi permetterà di gestire molto facilmente il vostro server.
Ovviamente è una scelta consigliata soprattutto ai principianti.

Installazione/Mac OS X

Leopard

Php 5 è già presente in Leopard ma non è abilitato per default. È sufficiente decommentare la seguente riga del file /etc/apache2/httpd.conf

#LoadModule php5_module libexec/apache2/libphp5.so

Per modificare un file di sistema dovremo utilizzare un programma da Terminale come *nano*; quindi riavviare il server apache da Preferenze di sistema oppure con il comando da Terminale

```
sudo /usr/sbin/apachectl graceful
```

Configurazione

La **configurazione di PHP** si può dividere in due passi:

- configurazione del web server per il parsing degli script
- configurazione del parser

Configurazione del web server

La configurazione del web server viene eseguita solo quando si usa PHP in ambito di programmazione web di script server side. Possiamo configurare il web server per eseguire i nostri script tramite un modulo SAPI, oppure tramite l'eseguibile CGI.

Configurazione di Apache

Apache è un noto web server. Solitamente Apache, così com'è, non è configurato affinché possa proibire o permettere l'accesso alle directory: avremo quindi bisogno di tre strumenti e di due modifiche ad altrettanti file di configurazione ed un file .htaccess.

La prima modifica

La prima modifica che dobbiamo fare è nel file "srm.conf": scorriamolo fino ad arrivare alla linea che inizia

con "AccessFileName": questa direttiva indica il nome del file da utilizzare per leggere gli attributi che dobbiamo dare alle directory nelle quali tale file è presente. Teoricamente, la direttiva dovrebbe essere seguita da ".htaccess". Una volta che avrete deciso il nome del file che conterrà le istruzioni e che avrete riavviato Apache per rendere efettive le modifiche, Apache andrà a cercare per ogni directory richiamata da un browser questo file, dal quale leggerà le istruzioni. Ovviamente, se questo non è presente, Apache agirà come di norma; se invece il file esiste, Apache lo leggerà e agirà di conseguenza: se abbiamo impostato delle protezioni tramite password, Apache farà in modo che il browser visualizzi una maschera nella quale vengono richiesti username e password, tramite i quali Apache può verificare o meno l'autenticità del richiedente.

La seconda modifica

La seconda modifica da attuare è nel file access.conf: scorrete anche questo file fino ad arrivare alla riga "AllowOverride": inserendo un "AuthConfig" Apache richiederà l'autentificazione. Le possibilità, oltre a questa sono molte, in modo da poter affinare il processo di protezione: se volete saperne di più, leggete il file manual/mod/core.html#allowoverride, presente nella directory locale della documentazione di Apache.

Riavviato Apache, saremo pronti a preparare il file che servirà a proteggere le directory che ci interessano, che supporremo essere ".htaccess".

La terza modifica

La terza modifica si attua nel file ".htaccess". Prima di tutto, dobbiamo creare un file ".htaccess" dentro questa directory, e scriviamoci:

```
AuthName "prova"
AuthType Basic
AuthUserFile /etc/apache/passwd
require valid-user
```

La prima riga indica il nome della protezione. La seconda riga indica il tipo di autorizzazione da eseguire: al momento, solamente l'autorizzazione del tipo "Basic" è usata, sebbene sia già in lavorazione un'autorizzazione "digest". La terza riga indica il file che Apache andrà a leggere per verificare se l'username e la password inseriti sono corretti. La quarta riga, infine, controlla gli username.

In definitiva, quindi, il file .htaccess completo potrebbe essere:

```
AuthName "prova"
AuthType Basic
AuthUserFile /etc/apache/passwd
AuthGroupFile /etc/apache/group
require valid-user
require group admin
```

Configurazione del parser

La configurazione del parser, che *deve essere eseguita* qualunque sia l'ambito di utilizzo di PHP, avviene tramite il file **php.ini**.

Programmazione

Analizzeremo ora i fondamenti del linguaggio, partendo dalle nozioni di base fino a descrivere tutte le caratteristiche fondamentali (alcune delle quali saranno trattate approfonditamente in seguito).

Ogni file viene interpretato dal motore di PHP scorrendo le istruzioni nell'ordine in cui sono scritte, dal basso verso l'alto. Non è necessario compilarli, come ad esempio in C. Ad ogni richiesta giunta al webserver (*input*) viene chiamato l'interprete PHP, il quale processerà il file oggetto della richiesta e restituirà un *output*. Nella maggior parte dei casi la richiesta proviene da un browser e l'output sarà in formato HTML.

Per poter essere eseguiti dall'interprete, i file PHP devono trovarsi in una cartella a cui il webserver abbia accesso, denominata *root directory*, e devono avere estensione *.php* (a meno di configurazioni particolari, che qui saranno ignorate).

PHP e HTML insieme

Il codice PHP può coesistere all'interno dello stesso file insieme al linguaggio HTML. Sebbene non sia una buona pratica, nella prima parte di questo libro gli esempi potranno contenere entrambi i linguaggi per una più facile comprensione. In seguito si esamineranno soluzioni migliori per scrivere il codice che faccia parte della "logica applicativa" quanto più possibile slegato dal codice "presentazionale".

Delimitare il codice PHP

Anche qualora non vi fosse HTML, ogni codice di PHP deve essere racchiuso dai tag `<?php` e `?>`.

Se l'interprete è configurato appositamente può riconoscere la sintassi abbreviata `<?` e `?>`, del tutto simile alla precedente ma il cui uso è fortemente scoraggiato e da evitare. D'ora in avanti nel testo troverete sempre la versione estesa `<?php`.

File contenenti solo PHP

Nel caso il file contenga soltanto codice PHP (senza testo o HTML) è preferibile omettere il tag di chiusura. In questo modo si eviterà che spazi, tabulazioni o interruzioni di linea inseriti accidentalmente dopo di esso possano causare errori (per esempio quando si gestisce il buffer dell'output o quando si inizializza una sessione). Si ricordi, ovviamente, di porre `<?php` all'inizio del file, prima di ogni altra cosa.

Separazione delle istruzioni

All'interno dei tag delimitatori si possono inserire un qualsiasi numero di **istruzioni**, ciascuna delle quali deve essere seguita dal punto e virgola per separarla dalla successiva. Esso può essere omesso nell'ultima istruzione. Tuttavia questa tecnica può generare confusione (o addirittura errore se si aggiungesse altro codice) pertanto non è una consigliata.

È possibile inserire più istruzioni sulla stessa riga o una singola istruzione su più righe. Sebbene tra ogni istruzione siano ammessi caratteri di spaziatura e tabulazioni, non si deve abusare di questa grande flessibilità; ogni programmatore dovrebbe scrivere codice pulito e leggibile.

```
<?php

  // Esempio di pessimo codice
  echo

    'Testo da mostrare al nostro utente'
 ;
```

```
   // Lo stesso codice più snello e leggibile
   echo 'Testo da mostrare al nostro utente';
```

Le righe che iniziano con i caratteri `//` sono commenti che il parser ignora totalmente. Saranno spiegati dettagliatamente nella prossima sezione.

Il primo esempio: hello world

Procediamo alla creazione di un file semplice che abbia come finalità solamente quello di mostrare all'utente la classica scritta "Hello world!" ("Ciao mondo!"), salvandolo con il nome *prova.php* nella directory riconosciuta dal webserver che abbiamo installato.

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Primo script PHP</title>
  </head>
  <body>
    <p><?php echo 'Hello world!'; ?></p>
  </body>
</html>
```

Se tutto è configurato correttamente, visitando l'indirizzo http://localhost/prova.php vedremo una pagina contenente la scritta "Hello world!". Per farlo abbiamo inserito un'istruzione che fa uso del costrutto `echo`, il quale invia in output una o più stringhe. `print` agisce allo stesso modo, è anch'esso un costrutto del linguaggio PHP (non, quindi, una funzione, che saranno esaminate più avanti in questo libro) ma può inviare in output una sola stringa.

```
<?php

  echo 'Prima stringa', 'Seconda stringa'; // Codice corretto

  print 'Prima stringa', 'Seconda stringa'; // Errore di sintassi
```

Esiste una sintassi abbreviata per inviare in output una stringa ed è la seguente: `<?= ?>`. Può essere utile quando ci si trova di fronte a un codice in cui è presente codice HTML, come visto nell'esempio precedente.

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Primo script PHP</title>
  </head>
  <body>
    <p><?= 'Hello world!' ?></p>
  </body>
</html>
```

Ovviamente usare un linguaggio come PHP per inviare output totalmente statici – come negli esempi visti – non avrebbe utilità. La potenza della programmazione risiede nella dinamicità, alla cui base vi sono le variabili che qui non sono state presentate per motivi di semplicità.

Programmazione/Commenti

All'interno dei tag `<?php` e `?>` è possibile inserire commenti al codice, porzioni di testo opportunamente

marcate che verranno ignorate dal motore PHP durante il *parsing* degli script.

L'uso dei commenti ha vari benefici, tra cui:

1. rendere più leggibile il sorgente da parte di altri utenti che eventualmente debbano variare lo script PHP (utile soprattutto nei casi di collaborazione tra più persone);
2. non eseguire una parte di codice senza doverla spostare o cancellare (potrebbe tornare utile in futuro, ad esempio).

Sintassi

Il codice può essere commentato in tre modi:

```
<?php

  // Commento su singola linea in stile 'C++'

  # Commento su singola linea in stile 'Unix shell'

  /* Commento multilinea (in stile 'C').
     Al suo interno è ammessa qualsiasi cosa,
     eccetto il delimitatore finale */
```

La differenza sostanziale è tra i primi due, che fanno uso di un solo delimitatore ma possono estendersi su una singola riga, e il terzo, che può estendersi su più righe.

Qualsiasi codice PHP presente nei commenti non viene eseguito dal *parser* tranne il tag di chiusura `?>` nei tipi a singola linea. Si veda il seguente esempio:

```
<?php
  echo 'Primo esempio'; // Commento del primo esempio ?>
  echo 'Altro testo';
```

Come detto, pur trovandosi sulla stessa riga del commento, il tag di chiusura **non** sarà ignorato e il secondo `echo` verrebbe interamente inviato in output al browser (in quanto esterno allo script). Anche usando il carattere `#` le cose non sarebbero andate diversamente.

La sintassi multi-linea, invece, non è affetta da questo rischio:

```
<?php

  /*
   * Esempio di commento in cui il parser non
   * elabora il tag di chiusura: ?>
   */
  echo 'Secondo esempio';
?>
```

Si tratta di una tipologia usata spesso all'inizio degli script per descrivere lo scopo del file. Le uniche attenzioni che richiede sono di non dimenticare il delimitatore di chiusura (causerebbe errore durante l'esecuzione) e di non annidare altri commenti scritti con la stessa sintassi. A tal proposito si veda il seguente codice:

```
<?php
  /*
    echo 'prova'; /* Test di esecuzione */
   */
```

Questo potrebbe essere un classico caso di codice scritto per verificare la corretta esecuzione (se appare "prova" nell'output vuol dire che funziona) commentato per eseguire altri test in futuro. Tuttavia il parser interpreterà il primo delimitatore di chiusura come la **fine dell'intero commento**, generando un errore di sintassi appena arriverà al successivo delimitatore. Pertanto non è possibile annidare commenti all'interno di altri.

Programmazione/Variabili

Possiamo pensare ad una variabile come una scatola nella quale immagazzinare le informazioni e da cui possiamo ottenerle quando è necessario.

Lavorare con le variabili

In PHP le variabili sono identificate dal simbolo $, che precede il nome della variabile stessa. È necessario tuttavia che il primo carattere dopo il $ non sia un numero o un carattere speciale, ma sia una lettera o un carattere underscore (_).

Molti linguaggi di programmazione richiedono che le variabili usate nel corso del programma siano dichiarate. Il linguaggio PHP è un linguaggio chiamato a **tipizzazione debole** , che significa invece che non richiede alcuna dichiarazione di variabile: per il motore PHP, infatti, una variabile è tale dalla prima riga nella quale se ne fa uso.

L'istruzione fondamentale che è possibile eseguire con una variabile è l'**assegnazione**, che imposta (assegna) il valore contenuto dalla variabile. La sintassi è

```
$nome_var = valore
```

dove *valore* è un'espressione valida per PHP (per espressione si intende una sequenza di dati, operatori e/o variabili che restituisca un valore). Sono ad esempio espressioni

```
3 //restituisce 3
3 + $var //restituisce il valore di $var sommato di 3.
```

Per fare riferimento ad una variabile e al suo valore sarà necessario semplicemente riferirsi al nome; si noti che PHP è case-sensitive, quindi `$var` e `$Var` sono due variabili differenti. Questo script, ad esempio, stampa il valore di una variabile:

```
<?php
 $variabile = "valore della variabile";
 echo "$variabile";
?>
```

Da notare che l'istruzione echo non stampa *$variabile* ma il valore della variabile `$variabile`; sarebbe equivalente scrivere

```
echo $variabile;
```

Un'istruzione, invece, come

```
echo '$variabile';
```

stamperebbe *$variabile* (si notino gli apici e non le virgolette doppie).

Variabili per valore e per riferimento

Le variabili PHP sono solitamente passate per **valore**: quando una variabile viene assegnata ad un'altra in realtà viene assegnato ad una variabile una copia del valore dell'altra, ma le due variabili identificano comunque due celle di memoria differenti. Ad esempio:

```
$var1 = 3;
$var2 = $var1 //ora $var2 contiene il valore 3
$var2 = 4 //in questo caso non cambia il valore di $var1 ma solo quello di $var2
```

Talvolta, soprattutto per quanto riguarda l'uso di funzioni, è comodo aver due variabili che puntino alla stessa cella di memoria. Per fare ciò si assegnano le **variabili per riferimento** usando la sintassi:

```
$var1 = &$var2
```

Ad esempio

```
$var1 = 3;
$var2 = &$var1 //ora $var2 e $var1 puntano alla stessa cella di memoria
$var2 = 4 //ora $var1 e $var2 contengono entrambe il valore 4
```

Costanti

Può essere comodo durante la programmazione definire valori **costanti** riutilizzabili nel codice. La differenza sostanziale tra costanti e varibili sta nel fatto che le prime, a differenza delle seconde, non possono essere modificate. Per definire una costante si usa la sintassi:

```
define("nome_costante", valore);
```

e per richiamarle si usa semplicemente il loro nome:

```
echo nome_costante;
```

Esistono alcune costanti predefinite, che sono valide cioè in tutti gli script:

- `__FILE__`: restituisce il percorso completo e il nome del file (ad esempio */var/www/html/index.php* su sistemi Linux)
- `__LINE__`: restituisce il numero di riga in cui si trova la costante
- `__FUNCTION__` e `__CLASS__`: restituiscono rispettivamente il nome della funzione e della classe in cui la costante è richiamata.

Tipi di dati

PHP è a tipizzazione debole anche perché converte automaticamente il tipo di dati contenuto nella variabile a seconda del contesto (questo è importante quando si usano gli operatori).

Nonostante ciò, il concetto di tipo di dato esiste in PHP: ogni variabile è di un determinato tipo a seconda del valore che contiene in quel momento. Principalmente i tipi di dato sono:

Nome	Descrizione	Esempio
Numero intero (*int*) o a virgola mobile (*float*)	un numero razionale o intero	`$a = 3; $b = -12.5;`
Stringa (*string*)	sequenza alfanumerica (testo); durante l'assegnazione deve essere delimitata da due virgolette (") o apici (').	`$a = "testo"; $b = '"I promessi sposi" è un romanzo di A. Manzoni';`
Booleano (*boolean*)	può assumere solo i valori *true* (vero) o *false* (falso)	`$a = true; $b = (3 == 5);`
Array	tipo di dato complesso, verrà trattato più avanti	
Null	indica l'assenza di un valore; serve soprattutto ad annullare una variabile	`$a = null;`

Di fronte a diversi tipi di dato, il motore PHP può trovarsi in diverse situazioni e si comporta in maniere differenti:

- se si aspetta un valore *numerico intero* e viene fornito un *numero a virgola mobile* PHP tronca la parte decimale, restituendo solo la parte intera
- se si aspetta un valore *numerico* e viene fornita una *stringa*, PHP elimina spazi e lettere di troppo utilizzando soltanto i numeri contenuti in tale stringa
- se si aspetta un valore *numerico* e viene fornito un valore *booleano* viene restituito 1 se il valore è TRUE, 0 se il valore è FALSE
- se si aspetta un *numero* e viene fornito un *array* restituisce un numero pari al numero di elementi contenuti dall'array
- se si aspetta una *stringa* e viene fornito un *numero* questo viene convertito in una stringa contentente esattamente il numero stesso
- se si aspetta un valore *stringa* e viene fornito un valore *booleano* viene restituito *1* se il valore è TRUE, una stringa vuota se è FALSE
- se si aspetta una *stringa* e viene fornito un *array* restituisce una stringa contenente il valore *array*
- se si aspetta un valore *booleano* e viene fornito un *numero* PHP restituisce FALSE se il numero è uguale a 0, TRUE se è il numero è diverso da 0 (di solito 1)
- se si aspetta un valore *booleano* e viene fornita una *stringa* PHP restituisce FALSE se la stringa è vuota o contiene il valore *0*; restituisce TRUE negli altri casi
- se si aspetta un valore *booleano* e viene fornita un *array* PHP restituisce FALSE se l'array è vuoto , TRUE negli altri casi
- il valore *null* viene trattato come un valore booleano FALSE

Esistono tuttavia numerose funzioni di conversione per trasformare un tipo di dato in un altro, che consistono nell'anteporre all'espressione in questione il nome del tipo di dato che si vuole ottenere tra parentesi. Ad esempio:

```
(int)(3.45 + 7.3)
```

restituisce *10*, in quanto viene convertito un numero *float* in un intero secondo le regole di conversione. Allo stesso modo

```
(boolean)("questa è un'espressione stringa")
```

restituisce TRUE

Calcolo multibase

Oltre al sistema decimale, PHP può lavorare con i sistemi di numerazione in base otto e sedici. Per inizializzare una variabile in base otto, il numero deve iniziare con uno `0` (es `01247`); i numeri in base sedici devono invece iniziare con `0x` (es `0xf56b`).

Stringhe

Meritano particolare attenzione le stringhe, soprattutto nell'analisi dei caratteri di commutazioni.
Una stringa in PHP deve essere delimitata da apici o da apici doppi; bisogna tuttavia chiudere una stringa con la stessa modalità con cui si è aperta:

```
"Questa non è una stringa valida'
'Questa lo è'
```

Può essere necessario in alcuni casi usare carattere particolari; ad esempio può essere necessario inserire un apice in una stringa delimitata da apici singoli. In questo caso si usano i **caratteri di commutazione** (o **sequenze di escape**). I principali sono:

\'	Singolo apice (necessario solo se la stringa è racchiusa da apici singoli)
\"	Doppio apice (necessario solo se la stringa è racchiusa da apici doppi)
\\	Backslash
\n	New line (ritorno a capo)
\r	Ritorno del carrello
\t	Tabulazione orizzontale
\v	Tabulazione verticale (disponibile nelle versioni di PHP superiori alla 5.2.5)
\f	form feed (disponibile nelle versioni di PHP superiori alla 5.2.5)
\$	Segno del dollaro
\x00 - \xFF	Carattere esadecimale

Nota: nel caso di stringhe racchiuse da apici singoli l'unica sequenza di escape ammessa è la prima (\')

Programmazione/Operatori

Nel linguaggio PHP affianco ai classici operatori matematici, booleani e logici sono disponibili anche gli operatori unari di incremento e decremento e un operatore ternario. Gli operatori principali sono:

- **matematici** (restituiscono e richiedono valori numerici)
  - + somma algebrica
  - - sottrazione o negazione del numero

    - * Moltiplicazione
    - / divisione
    - % modulo (resto della divisione intera)

- **stringa** (restituiscono una stringa)
    - . (punto) concatena due stringhe

Gli operatori visti finora hanno inoltre una sintassi particolare nel caso di espressioni come ad esempio

```
$a = $a + 3;
$b = $b.' stringa';
```

Queste espressioni, infatti, nelle quali compare la stessa variabile ambo sia a destra che a sinistra dell'uguale, possono essere riassunte in

```
$a += 3;
$b .= ' stringa';
```

- booleani o di **confronto** (restituiscono un valore boolean)
    - === identicamente uguale (anche del medesimo tipo)
    - == uguale a
    - != diverso da
    - > maggiore di
    - < minore di
    - => maggiore o uguale a
    - <= minore o uguale a

- **logici** (restituiscono e operano su boolean)
    - ! corrisponde alla negazione logica ed è un operatore unario (necessita di un solo operando). Restituisce false se l'operando è true, true se viceversa.
    - and o && corrisponde alla congiunzione logica (et). Restituisce true solo se entrambi gli operandi sono veri.
    - or o || corrisponde disgiunzione inclusiva logica (vel). Restituisce true anche se uno sole degli operandi è vero.
    - xor corrisponde alla disgiunzione esclusiva logica (out). Restituisce true solo se uno dei due valori è true e l'altro è false;

- due operatori molto importanti e comodi in PHP sono gli operatori chiamati di **incremento** e di **decremento** ++ e --, che restituiscono un valore numerico e aumentano o diminuiscono il valore della variabile di una unità. È più facile capire il loro funzionamento con un esempio:

```
$v1 = $v2++; //assegna a $v1 il valore di $v2 e poi incrementa $v2 di 1
$v1 = $v2-- //assegna a $v1 il valore di $v2 e poi decrementa $v2 di 1
$v1 = ++$v2; //incrementa $v2 di 1 e poi assegna a $v1 il valore di $v2
$v1 = --$v2 //decrementa $v2 di 1 e poi assegna a $v1 il valore di $v2
```

- un altro operatore molto comodo in PHP è l'**operatore ternario** ? : la cui sintassi è

```
condizione ? espr1 : espr2
```

Quando il motore PHP legge questo operatore valuta il valore di *condizione*: se è vera, restituisce il valore *espr1*, altrimenti il valore *espr2*. Un esempio potrebbe rendere più chiaro il tutto:

```
$var = ($a > $b ? 'a maggiore di b' : 'a minore di b');
```

Il valore di $var sarà quindi dipendente dal valore booleano dell'espressione `$a > $b`

L'operatore ternario può essere usato anche per determinare il valore di un parametro da passare ad una funzione.

Ad esempio:

```
function prova( $valore ) {
   echo $valore;
}

prova( true  ? "prova 1 " : "prova 2" );
prova( false ? "prova 1 " : "prova 2" );
```

Il codice sopra riportato darà come output:

```
prova 1 prova 2
```

Programmazione/Array

Gli **array** (o **vettori**) sono delle strutture dati complesse che risultano molto comode per la codifica di particolari algoritmi.

Possiamo pensare agli array come a delle liste di elementi nelle quali ciascun elemento ha un valore e un indice (o chiave) numerico o alfanumerico che lo identifica nella lista.

Gli array possono essere **numerici** o **associativi**: nel primo caso ciascun elemento della lista è identificato unicamente da un indice numerico; nel secondo caso ogni valore ha un indice numerico e uno alfanumerico, che può quindi memorizzare altri dati particolari.
In PHP non esistono matrici multidimensionali ma queste possono essere emulate creando strutture (anche molto complesse) con array di array, dal momento che ciascun elemento di un array può a sua volta essere un array:

```
$valore = array();
$riga = 2;
$colonna = 3;
$nome = 'Pippo';

$valore[$riga][$colonna][$nome] = 10;
```

Per inizializzare una variabile come array occorre dichiarare la variabile come tale e si utilizza la seguente notazione:

```
$array = array();
```

Per fare riferimento ad un elemento dell'array si usa la sintassi

```
$array[indice]
```

dove indice è un numero oppure, come spiegato sopra, una chiave alfanumerica.
Per aggiungere all'array un elemento con un indice *n* è sufficiente fare riferimento all'elemento stesso, come nell'esempio seguente:

```
$array = array();
$array[n] = "prova";
```

Per inserire automaticamente un elemento alla fine della lista si usa invece la sintassi

```
$array[] = valore;
```

L'indice assegnato così all'elemento appena inserito è pari all'elemento maggiore incrementato di uno. Se l'array è vuoto, viene assegnato indice 0.

```
$array = array();
$array[] = "prova1";
$array[] = "prova2";
$array[] = "prova3";

echo $array[0];
echo $array[1];
echo $array[2];
```

Questo semplice spezzone di codice restituirà

```
prova1prova2prova3
```

Se vogliamo indicare la posizione in cui inserirlo, oppure la chiave a cui dovrà essere associato, dovremo inserire il valore usando la sintassi seguente:

```
$array = array();
$array[1] = "prova 1";
$array["prova"] = "prova 2";
```

Ovviamente se un elemento è già presente a tale indice o chiave questo verrà sovrascritto.

È possibile assegnare dei valori all'array già in fase di dichiarazione, passandoli come parametri alla funzione array() che si occupa appunto di creare un nuovo array.

```
$array = array( "prova1", "prova2", "prova3" );
echo $array[0];
echo $array[1];
echo $array[2];
```

Questo spezzone di codice restituirà

```
prova1prova2prova3
```

proprio come nell'esempio fatto sopra.

Funzioni utili

print_r

print_r è una funzione molto utile in php, non solo per gli array, ma anche per molti altri tipi di oggetto. Nel caso degli array, comunque, è funzionale perché consente di stampare il contenuto degli stessi in modo molto utile per eseguire, ad esempio, un veloce debug.

Ad esempio

```
$array = array( 100, 200, 300 );
print_r( $array );
```

restituirà il seguente output:

```
array(
   [0] => 100,
   [1] => 200,
   [2] => 300
 )
```

count

La funzione count, come suggerisce il nome, restituisce il numero di elementi contenuti nell'array.

```
$array = array( 1, 2, 3, 4 );
echo count( $array );
```

avrà come output

```
4
```

implode ed explode

Queste due funzioni lavorano con le stringhe e gli array. La sintassi è:

```
implode (collante, array)
explode (delimitatore, stringa)
```

La prima restituisce una stringa ottenuta concatenando in ordine tutti gli elementi di `array` inframezzandoli dalla stringa `collante`.
La seconda restituisce un array ottenuto separando `stringa` utilizzando `delimitatore` come separatore.
Ad esempio:

```
$arr = array('a','b','c');
$stringa = implode(":", $arr); //restituisce a:b:c
```

foreach

vedi anche qui

Questa non è proprio una funzione, ma una struttura di controllo. Funziona così:

```php
$arr = array('a','b','c');
foreach($arr as $variabile_chiave => $variabile_contenuto)
{
 // istruzioni
}
```

Esempio:

```php
$arr = array('a','b','c');
foreach($arr as $valore)
{
 echo $valore;
}
```

oppure, se si vuole visualizzare anche la chiave dell'array si fà così:

```php
$arr = array('a'=>'lettera 1','b'=>'lettera 2','c'=>'lettera 3');
foreach($arr as $key=>$valore)
{
 echo "La chiave dell'array ".$key." è uguale a ".$valore.".<br>";
}
```

Il primo esempio restituirà `abc`, il secondo

```
La chiave dell'array a è uguale a lettera 1.
La chiave dell'array b è uguale a lettera 2.
La chiave dell'array c è uguale a lettera 3.
```

Si può utilizzare anche per fare qualcosa di più che un semplice echo, per esempio possiamo cambiare il valore di un elemento nel caso sia uguale a...:

```php
$arr = array('a','b','c');
print_r($arr);
foreach($arr as $i => $valore)
{
 if ( $arr[$i] == 'b' )
  $arr[$i]='mario';
}
echo "<br>";
print_r($arr);
```

l'output sarà

```
Array ( [0] => a [1] => b [2] => c )
Array ( [0] => a [1] => mario [2] => c )
```

unset()

Unset distrugge la variabile ma non libera immediatamente la memoria!! La memoria verrà liberata quando ci sarà qualche ciclo di cpu non utilizzato oppure quando il programma finirà la memoria.
Nel caso degli array, unset($mioarray); distruggerà l'array, con foreach invece distruggeremo i valori all'interno dell'array ma non l'array stesso. Infatti se:

```php
<?php
$arr=array('mario','ciccio','fuffi');
print_r($arr);
unset($arr);
echo "<br /> unset array arr, arr e'  ancora un array?<br />";
echo is_array($arr) ?  "si arr e' un array" : "no arr non e' un array";
```

```
echo "<br><br>";


$mioarr=array('ena','dva','tri');
foreach($mioarr as $i => $value)
   unset($mioarr[$i]);
echo "cancellati i valori dell'array mioarr con foreach, e' ancora un array? <br />";
echo is_array($mioarr) ?  "si mioarr e' un array" : "no mioarr non e' un array array";
?>
```

l'output sarà:

```
Array ( [0] => mario [1] => ciccio [2] => fuffi )
unset array arr, arr e' ancora un array?
no arr non è un array


cancellati i valori dell'array mioarr con foreach, e' ancora un array?
si mioarr è un array
```

Attenzione!

Se utilizziamo foreach per cancellare gli elementi dell'array, non reindicizzeremo l'array, quindi:

```
$mioarr=array('uno','due','tre');
print_r($mioarr);

foreach($mioarr as $i => $value)
    unset($mioarray[$i];

print_r($mioarr);

mioarr[]='ciccio';
print_r($mioarr);
```

restituirà:

```
Array ( [0]=>uno [1]=>due [2]=>tre )
Array()
Array ( [3]=>ciccio )
```

l'array non viene reindicizzato come quando usiamo unset($mioarray); la stessa cosa se cancelliamo solo un elemento:

```
unset($mioarr[1]);
```

ecco che l'array sarà adesso:

```
print_r($mioarr);
```

```
Array ( [0]=>uno [2]=>tre )
```

per reindicizzarlo dovremo usare:

```
$mioarr=array_values($mioarr);
```

array_values restituisce tutti i valori contenuti nell'array e indicizza numericamente l'array destinatario.
L'array ora conterrà:

```
Array ( [0]=>uno [1]=>tre )
```

Altre funzioni

Per vedere tutte le funzioni riguardanti gli array, guardare nella sezione Array functions (http://www.php.net/manual/it/ref.array.php) della Documentazione ufficiale (http://it.php.net/manual/it/index.php)

Programmazione/Condizioni

La **condizione**, o **selezione**, è una struttura che permette di eseguire istruzioni differenti in base ad una condizione indicata all'inizio.

Selezione binaria

La selezione binaria consente in una scelta tra due possibilità: vero o falso.
Questo tipo di selezione si basa sul valore booleano di un'espressione indicata all'inizio della struttura ed esegue il primo blocco indicato se la condizione è vera, altrimenti esegue l'eventuale secondo blocco.
In PHP questa condizione si accede tramite il costrutto `if... then... else` che funzionano esattamente come nel linguaggio C, con l'unica differenza che `else if` si scrive tutto attaccato: `elseif`. Ad esempio:

```
<?php
$x = 10;
$y = 10;
if ($x == $y) {
 print "\$x e' uguale a \$y: $x\n";
}
elseif ($x > $y) {
 print "\$x e' maggiore di \$y: $x > $y\n";
}
else {
 print "\$x e' minore di \$y: $x < $y\n";
}
?>
```

Il risultato sarà:

```
$x e' uguale a $y: 10
```

Selezione multipla

La selezione multipla consiste in una scelta tra due o più possibilità in base al valore di una espressione valutata inizialmente; si esprime con il costrutto `switch... case`, che funziona esattamente come nel linguaggio C.

Il costrutto esegue il confronto dell'espressione passata a `switch` con tutti i valori `case` ed il confronto si interrompe quando viene incontrata l'istruzione `break`.

```
<?php
$x = 5;
switch ($x) {
 case 0:
  print "\$x e' uguale a 0\n";
  break;
 case 1:
  print "\$x e' uguale a 1\n";
```

```
  break;
 case 2:
  print "\$x e' uguale a 2\n";
  break;
 case 3:
 case 4:
  print "\$x e' uguale a 3 oppure a 4\n";
  break;
 default:
  print "\$x e' minore di 0 o maggiore di 4\n";
  break;
}
?>
```

Il risultato sarà:

```
$x e' minore di 0 o maggiore di 4
```

Programmazione/Cicli

Il **ciclo** (o **iterazione**) è una struttura di controllo (come la selezione) che permette l'esecuzione di una sequenza di una o più istruzioni fino al verificarsi di una data condizione.
In PHP, questi vengono gestiti esattamente come nel linguaggio C. Vedi C/Blocchi e funzioni/Cicli.

Il ciclo Foreach

vedi anche la voce array

Un tipo particolare di ciclo, non presente in C e nei linguaggi derivati, è il ciclo `foreach`, che si presta ad essere usato in molte situazioni.

La sintassi tipica dell'istruzione è questa:

```
foreach( $variabile_su_cui_iterare as $valore ) {
    istruzioni
}
```

Tipicamente $variabile_su_cui_iterare è un array. In questo caso il ciclo scorrerà tutti gli elementi dell'array, assegnando di volta in volta il loro valore alla variabile `$valore`, che potrà essere usata all'interno del corpo del ciclo.

Se dovesse essere necessario, per qualche ragione, conoscere anche la chiave dell'array a cui tale oggetto è associato, è possibile usare la sintassi

```
foreach( $variabile_su_cui_iterare as $chiave=>$valore ) {
    ''istruzioni''
}
```

Anche se foreach è tipicamente usato per scorrere tutti gli elementi di un array può essere usato anche per scorrere tutti membri pubblici, o comunque accessibili, di una classe.

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:

```
<?php
$variabile = 1;

while ($variabile <= 10) {

$risultato = 5 * $variabile;

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

$variabile++;

}
?>
```

Il ciclo while, 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 For

Il ciclo for permette di eseguire tre operazioni in una sola riga. Prendendo l'esempio sopra del ciclo while, anzichè inizializzare la variabile $variabile in una riga, mettere la condizione $variabile <= 10 in un'altra e porre l'istruzione di incremento alla fine, il ciclo for permetterà di riassumere tutto in un'unica riga:

```
for($variabile = 1 ; $variabile <=10 ; $variabile++ ;){
 istruzioni;
}
```

Più generalmente:

```
for(expr1; expr2 ; expr3 ){
 istruzioni;
}
```

dove expr1 indica l'istruzione da eseguire solo la prima volta all'entrata nel ciclo, expr2 indica la condizione che, se falsa, causa l'uscita dal ciclo, expr3 indica l'istruzione da eseguire una volta eseguite tutte le istruzioni del ciclo, prima di valutare la veridicità della condizione expr2.
Tutte le espressioni expr1 expr2 expr3 possono essere composte da più istruzioni separate da virgola:

```
for($i=0, $k=1, $l=5; expr2 ; $i++, $k++, $l++ ){
 istruzioni;
}
```

bisogna porre attenzione con expr2, in quanto anche questa espressione può essere composta da più istruzioni separate da virgola, ma il risultato verrà preso solo dall'ultima di queste istruzioni.

Programmazione/Funzioni di base

Una **funzione** è un blocco di codice che può richiedere uno o più parametri in ingresso e può fornire un valore di uscita.
PHP mette a disposizione numerose funzioni predefinite, di cui non si ha sottomano il codice ma risultano molto utili o talvolta indispensabili nella programmazione delle nostre applicazioni server.
In PHP la maggior parte delle funzioni restituisce un valore anche quando ciò potrebbe non essere ovvio: spesso, ad esempio, le funzioni restituiscono un valore boolean che indica l'esito della sua esecuzione.

Questo elenco mostra solo le principali funzioni che vi capiterà di utilizzare programmando in PHP. Per l'elenco completo, consulta le referenze ufficiali di PHP (http://it.php.net/manual/en/funcref.php)

Usare le funzioni

Per utilizzare una funzione non bisogna fare altro che **richiamarla** (o **invocarla**). Se esiste ad esempio una funzione `abs` sarà sufficiente usarla in questa maniera:

```
$a = abs($b);
$c = 4 * abs(-65);
```

Nel momento in cui ci riferiamo alla funzione si dice che la stiamo appunto richiamando.

Funzioni di stringa

- **strlen**

```
strlen(stringa)
```

  Restituisce il numero di caratteri di una stringa

- **strstr**

```
strstr(stringa1, stringa2)
```

  Restituisce la parte di `stringa1` che si trova dopo la prima occorrenza di `stringa2` in `stringa1`. Se non viene trovata alcuna occorrenza, la funzione restituisce `false`. Ad esempio:

```
strstr("it.wikibooks.org", ".")
```

  restituisce `".wikibooks.org"`

- **strpos**

```
strpos(stringa1, stringa2)
```

  Restituisce la posizione della prima istanza di `stringa2` in `stringa1`. Si noti che il conteggio dei

caratteri avviene a partire da 0. Ad esempio:

```
strpos("it.wikibooks.org", ".")
```

restituisce 2. Se si vuole cercare il carattere successivo si può usare questo metodo:

```
strpos(strstr("it.wikibooks.org", "."), ".")
```

- **chr** e **ord**

```
chr(numero)
ord(carattere)
```

Queste due funzioni restituiscono rispettivamente il carattere corrispondente al carattere *numero* nella tabella ASCII e la posizioni di `carattere` sempre nella tabella ASCII. Le due funzioni sono complementari.

Funzioni numeriche

- **abs**

```
abs(numero)
```

Restituisce il valore assoluto di `numero`

- **pi**

```
pi()
```

Restituisce un valore approssimato di Pi greco

- **pow**

```
pow (base, esponente)
```

Restituisce `base` elevato all'`esponente`. L'esponente fornito deve essere positivo

- **rand**

```
rand(min, max)
```

Restituisce un numero casuale compreso tra `min` e `max` (inclusi)

- **round**, **ceil**, **floor**

```
round(numero, n)
ceil(numero)
floor(numero)
```

Restituiscono il valore di `numero` arrotondato rispettivamente di `n` posizioni decimali, all'unità per eccesso e all'unità per difetto

- **number_format**

```
number_format(numero, n, sep_decimali, sep_migliaia)
```

Restituisce una stringa contenente un valore formattato di `numero` in base ai parametri passati.
Accetta uno, due o quattro parametri.
Se ne viene passato solo uno, viene restituito il numero senza decimali usando la virgola come separatore di migliaia
Se vengono passati due parametri, viene restituito il numero formattato con `n` cifre decimali, usando il punto come separatore decimale e una virgola come separatore di migliaia.
Se sono passati tutti i parametri, viene restituito il numero con `n` cifre decimali, usando `sep_decimali` come separatore decimale e `sep_migliaia` come separatore di migliaia.

- **base_convert**

```
base_convert(numero, da_base, a_base)
```

Converte una stringa contente `numero` espresso nella base `da_base` in una stringa contente il corrispondente nella base `a_base`. Ad esempio:

```
base_convert('f',16,10);
```

restituisce 15.

Funzioni per data e ora

- **time**

```
time()
```

La principale funzione di data-ora è `time` che restituisce l'ora corrente del server in formato Unix Timestamp. Il formato Unix Timestamp indica il numero di secondi passati dalla cosidetta Unix Epoch (1 gennaio 1970 alle ore 00:00:00 GMT) ed è usato da PHP per gestire le date.

- **date**

```
date(formato, ora)
```

Una volta ottenuto un valore di data valido (ad esempio tramite la funzione `time` o da un database MySql) è possibile formattarlo usando la funzione `date` che accetta un parametro obbligatorio, la stringa `formato`, e un parametro facoltativo `ora` che specifica la marcatura temporale da formattare e restituisce una stringa.
Per formattare una data può essere comodo usare le costanti predefinite di PHP per la formattazione delle date come DATE_COOKIE, che formatta le date secondo il metodo usato nei cookie HTML (es. Monday, 15-Aug-05 15:52:01 UTC) (si veda qui· (http://it.php.net/manual /en/ref.datetime.php#datetime.constants) l'elenco completo)

Per ottenere un formato personalizzato, nella stringa è possibile inserire i seguenti parametri (si ricorda che l'output può variare in base alle impostazioni di lingua del server):

Carattere	Descrizione	Esempio
Giorni		
d	Giorno del mese, senza zeri aggiuntivi	`1 - 31`
j	Giorno del mese, con zeri aggiuntivi	`01 - 31`
D	Giorno della settimana in formato breve (tre lettere)	`Mon - Sat`
l (L minuscola)	Giorno della settimana in formato completo	`Monday - Saturday`
S	Suffisso del giorno del mese per i numeri ordinali inglesi (funziona bene insieme a *d*)	`st`, `nd`, `rd`, `th`
Mesi		
n	Numero del mese, senza zeri aggiuntivi	`1 - 12`
m	Numero del mese, con zeri aggiuntivi	`01 - 12`
F	Nome del mese completo	`January - December`
Anni		
y	Numero dell'anno in due cifre	`92`, `01`
Y	Numero dell'anno in quattro cifre	`1992`, `2001`
L	Restituisce "1" se l'anno è bisestile, "0" altrimenti	
Ore		
g / *G*	Ora in formato 12/24 ore, senza zeri aggiuntivi	`1 - 12`, `0 - 23`
h / *H*	Ora in formato 12/24 ore, con zeri aggiuntivi	`01 - 12`, `00 - 23`
i / *s*	Minuti/secondi, con zeri aggiuntivi	`00 - 59`
Altro		
U	Secondi dalla Unix Epoch (come `time`)	

È possibile ottenere inoltre qui (http://it.php.net/manual/en/function.date.php#id3096299) l'elenco completo. I valori non predefiniti saranno usati tali quali; è possibile inoltre evitare i caratteri speciali siano valutati è possibile usare il carattere di commutazione "\" davanti alla lettera. In questo caso si faccia però attenzion a commutare il carattere "\" nel caso le sequenze non corrispondano ad altri caratteri di commutazione. Ad esempio:

```
date("l \\t\h\e jS") //è necessario commutare "\" perché \t è la tabulazione
```

restituisce, ad esempio, `Sunday the 29th`.

- **mktime**

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

Restituisce una data in un formato valido per PHP partendo dai valori passati come parametri. L'anno

può essere espresso con 2 o 4 cifre.
L'uso di questa funzione permette di creare facilmente date da formattare. Ad esempio:

```
echo(date("d F Y", mktime(0,0,0,30,07,1992)));
```

restituisce

```
30 luglio 1992
```

Altre funzioni utili

- **include** e **include_once**

```
include(file)
include_once(file)
```

Queste due funzioni risultano molto utili nella programmazione di applicazione complesse, in quanto permettono di includere nella pagina PHP un altro file esterno.
Quando incontra queste funzioni, il motore PHP:

1. chiude i tag <?php ?> entrando quindi in modalità normale
2. inserisce ed analizza il testo del file `file`; per inserire codice PHP sarà quindi necessario inserire nuovamente i tag <?php e ?>
3. riapre il tag <?php

In questo modo è possibile creare applicazione modulari inserendo nel file che verrà incluso le nostre funzioni personalizzate (che vedremo nel prossimo modulo.
Si faccia attenzione se si inserisce codice PHP nei file da includere a salvarli con estensione .php (soprattutto se contengono informazioni sensibili come password o dati personali) altrimenti un eventuale utente malintenzionati potrebbe leggerne il contenuto, in quanto il codice PHP non verrebbe prima letto dal server.
La funzione `include_once` si differenzia da `include` in quanto include il file solo se esso non è già stato incluso nella pagina.

Programmazione/Funzioni personalizzate

In ogni linguaggio di programmazione esiste la possibilità di creare le proprie **funzioni personalizzate**; ciò è possibile ovviamente anche in PHP. La comodità della creazione di funzioni personalizzate risulta evidente nei casi di applicazioni complesse (nelle quali le funzioni possono servire a spezzettare il programma in piccoli sottoprogrammi) o di operazioni compiute frequentemente. È utile inoltre creare anche vere e proprie funzioni nel senso matematico del termine per risolvere problemi che ricorrono spesso.

Definire una funzione

La definizione di una nuova funzione in PHP è la seguente:

```
function nome_funzione ($arg1, $arg2, ...)  {
 //istruzioni
```

```
}
```

Dove `$arg1, $arg2` sono eventuali variabili che assumeranno i valori presi come parametri.

Impostare un valore di ritorno

In una funzione personalizzata per impostare il valore restituito dalla funzione si usa l'istruzione `return`. La sua sintassi è

```
return espr;
```

Quando il motore PHP incontra questa funzione restituisce il valore di `espr` e interrompe l'esecuzione del blocco della funzione tra parentesi.

Vediamo adesso un esempio che fa uso del comando return. Vogliamo realizzare un convertitore euro -> lire.

```
<?php
// 15 euro
$valuta = 15;

// definizione della funzione "converti"
function converti($euro)
{
    // effettuo la conversione
    $lire = $euro * 1936.27;

    // restituisco il valore calcolato
    return $lire;
}

// chiamo la funzione converti. Questa volta dobbiamo usare una
// variabile per raccogliere il valore restituito dalla funzione!
$vecchio_conio = converti($valuta);
echo "$valuta euro equivalgono a $vecchio_conio lire";
?>
```

Analizziamo la funzione "converti". Quando viene invocata riceve un parametro (la quantità di euro da convertire in lire) che viene memorizzata nella variabile $euro. La prima istruzione effettua la conversione vera e propria. Per far si che la funzione restituisca il risultato calcolato viene usato il comando return affiancato dalla variabile da restituire $lire.

Per utilizzare la funzione converti dovremo quindi specificare anche la variabile che raccoglierà il risultato restituito da tale funzione. Nel nostro esempio sarà la variabile $vecchio_conio a conservare tale valore.

Anche se il comando return è in grado di restituire una sola variabile è possibile far si che una funzione restituisca anche più di un valore attraverso un semplice espediente. E' sufficiente infatti impacchettare tutti i valori da restituire all'interno di un array e poi usare il comando return proprio con questo array.

Usare i parametri

Una volta definita una funzione è possibile lavorare sui parametri indicati tra parentesi, che possono essere passati per **valore** o per **riferimento**, allo stesso modo con le variabili. Ad esempio:

```
function prova (&$param1, $param2) {
 $param1 = $param2 + 5;
}
```

```
prova($var1, 3);
```

Dopo l'esecuzione della funzione la variabile `$var1` conterrà così il valore 8. Si faccia ovviamente attenzione a passare come parametro una variabile e non un'espressione, perché altrimenti non è possibilie effettuare il passaggio per riferimento.

Si noti che una funzione personalizzata deve essere invocata necessariamente dopo la sua dichiarazione. Ad esempio:

```
$var1 = funzione_esempio();
function funzione_esempio() {
 return "ciao!";
}
```

In questo caso il motore PHP restituirà un errore, in quanto la lettura dello script avviene in sequenza e la funzione `funzione_esempio` non è ancora stata dichiarata nel momento in cui viene chiamata.

Parametri predefiniti

È possibile inoltre prevedere che l'utente non passi alcun valore nel chiamare la funzione ed impostare dei valori predefiniti che deve assumere il parametro nel caso non venga specificato. Ad esempio:

```
function predef ($arg1, $arg2 = 10) {
 return $arg1 + $arg2  ;
}
echo predef(23); //restituisce 33
```

Programmazione/Variabili globali

Le **variabili globali** nascono e muoiono con una sessione di lavoro. Possono essere memorizzate sul server o sul client: in questo secondo caso utilizzano i cookies.

Per far sì che una variabile diventi globale occorre utilizzare la funzione *register* che ne permette la registrazione. Così operando, mentre le normali variabili globali delle form vengono registrate automaticamente, sarà possibile utilizzare altre variabili che avranno validità per l'intera durata della sessione.

PHP utilizza degli *array associativi* per diverse variabili globali. I più importanti sono $_POST e $_GET. Seppure a partire dalla versione 4.3 PHP permette di utilizzare automaticamente le variabili passate dai form, questi due array consentono di recepire le variabili passate attraverso i form sia con il metodo GET sia con il metodo POST.

Per quando riguarda le altre variabili globali, da utilizzare all'interno di uno script PHP, vale una regola che potremmo considerare inversa alle regole di *scope* di C e dei principali linguaggi: se all'interno di una pagina o di una funzione voglio utilizzare la variabile *$userid* è opportuno che utilizzi prima la notazione *global $userid*. Operando in questo modo la variabile globale acquista visibilità all'interno della funzione. Per esempio:

```
$var1="pippo";
function prova(){
   global $var1;
```

```
   echo $var1;
}
```

Sommario della sottosezione

- $GLOBALS
- $_GET
- $_POST
- $_SESSION
- $_COOKIE
- $_SERVER

Programmazione/Variabili globali/$GLOBALS

L'array associativo **`$GLOBALS`** contiene i riferimenti a tutte le variabili locali visibili dalla root. È una variabile superglobale, quindi non c'è bisogno di scrivere `global $GLOBALS` all'interno di una funzione per potervi accedere:

```
$variabile = 'Valore della variabile';

function test_globals() {
  echo $GLOBALS['variabile'];
}

test_globals(); // Visualizzerà "Valore della variabile"
```

$GLOBALS contiene anche i riferimenti agli altri array superglobali:

- `$_GET`
- `$_POST`
- `$_SERVER`
- `$_COOKIE`
- `$_SESSION`
- `$_FILES`
- `$_ENV`
- `$_REQUEST`
- `$GLOBALS`

Si noti che $GLOBALS è *ricorsivo*, cioè contiene un riferimento a sé stesso; è tecnicamente corretto – sebbene totalmente inutle – accedere a una variabile anche nel seguente modo:

```
$variabile = 'Valore della varabile';

echo $GLOBALS['GLOBALS']['GLOBALS']['variabile']; // Anche così verrà visualizzato "Valore della variabi
```

Programmazione/Variabili globali/$ GET

Cos'è

$_GET (o **$HTTP_GET_VARS** se la versione di PHP è inferiore alla 4.1.0) rappresenta un array associativo di variabili passate tramite barra degli indirizzi; alle coppie di chiavi e valori nell'array corrispondono le coppie di nomi e valori nella *querystring*.
In molte forme $_GET può essere simile a $_POST, ma la querystring non può superare i 255 caratteri; inoltre è poco sicuro in quanto è molto facile per un utente malintenzionato appendere valori alla querystring senza che vengano effettuati controlli precedenti. Una tecnica eventuale è quella di passare i valori crittandoli tramite la funzione `md5`, un algoritmo di criptazione univoco e non retroversibile.

Utilizzo

Si può chiamare una pagina passandole variabili tramite $_GET da un *form HTML*, a patto che la proprietà html del form in questione sia impostata a GET (es. <form action="pagina.html" method="get">). In questo caso le coppie di nomi-valori saranno dati dai nomi dei campi form e dai rispettivi valori.
È possibile anche chiamare una pagina passandole variabili per indirizzo semplicemente chiamando la pagina e accodando al nome **?**, seguito dalle variabili in ordine chiave=valore e suddivise da una **&**.

```
www.sito.com/chk.php?var1=valore1&var2=valore2
```

In questo caso avremmo un array che avrà per chiavi `var1` e `var2` e per valori rispettivamente `valore1` e `valore2`.

Esempi

In un Forum

Un classico utilizzo dell'array *superglobale* $_GET è quello della visualizzazione di un particolare *thread* di un forum.
In questo caso sarà necessario predisporre una pagina ad esempio showThread.php che, letto un valore passato come ID, restituisca il thread corrispondente.
Per fare riferimento, ad esempio, al thread con id 23, si potrà usare la notazione

```
www.forumTestWiki.it/showThread.php?id=23
```

In queste situazioni l'utilizzo di GET risulta comodo, in quanto:

1. non c'è alcun problema di sicurezza (l'ID del thread non è un dato sensibile)
2. un utente può creare un segnalibro alla pagina in modo che, poiché contiene nella URL l'ID del thread, questo sarà caricato automaticamente.

Per un login

È possibile anche utilizzare la *querystring* per un sistema di login, ma questo metodo è decisamente carente in fatto di sicurezza, poiché i dati come nome utente e password sarebbero palesemente visualizzabili a schermo.

Programmazione/Variabili globali/$ POST

Cos'è

$_POST (o **$HTTP_POST_VARS** se la versione PHP è inferiore alla 4.1.0) è una delle variabili predefinite di sistema.

In sostanza è un *array associativo* di chiavi e valori i cui elementi sono rappresentati da tutti i campi passati allo script da un *form* con `method` impostato a `POST` e dai rispettivi valori; il suo funzionamento è quindi simile a $_GET ma i valori non sono passati nella querystring ma tramite il *response* HTTP.

Utilizzo

È possibile accedere agli elementi di questo array iterando su di essi con un ciclo `foreach` oppure reperire il singolo valore di un elemento se ne conosciamo la chiave, ad esempio:

```
$_POST[chiave]
```

L'accesso alle variabili potrà essere fatto anche come array semplice: $_POST[0] - purché ci si ricordi a cosa corrisponde nel form chiamante.

Ad esempio, supponiamo che in una determinata pagina vi sia un form con il metodo POST e due campi di input, uno chiamato *userid* (`name="userid"`) ed uno chiamato *pwd*.
La pagina che verrà chiamata dalla form potrà accedere al valore delle variabili attraverso l'array associativo $_POST: cioè, $_POST["userid"] ci permetterà di accedere al valore della stringa scritta nel campo corrispondente, e lo stesso per quanto riguarda il campo pwd. L'accesso alle variabili potrà essere fatto anche come array semplice: $_POST[0], anche se questo metodo rende il codice più *oscuro*.

Nelle versioni a partire dalla 4.3 l'uso delle variabili globali nelle form è stato gestito in maniera automatica; ad esempio, se la form chiamante ha un campo chiamato *userid*, la pagina chiamata potrà utilizzare direttamente la variabile *$userid* ed in essa si troverà il valore corrispondente. Questo utilizzo è tuttavia sconsigliato soprattutto per motivi di sicurezza.

Esempi

Un esempio è quello di un form per un login contente un campo `id` e un campo `pwd` e con `METHOD="POST"`

Nella pagina di arrivo del modulo sarà quindi possibile eseguire una query al database contenente gli userID e le password, così da verificare se esiste o meno un utente con tali caratteristiche.

Programmazione/Variabili globali/$ SESSION

Cos'è

$_SESSION rappresenta un array associativo contenente le variabili attive e valorizzate per la sessione in corso.
Per *sessione* si intende l'arco di tempo dal momento in cui un client esegue la prima request al vostro server fino al momento in cui il server invia la sua ultima risposta al client.
Il protocollo HTTP è infatti stateless, non permette cioè il controllo dello stato (le variabili della pagina, i suoi contenuti) tra una richiesta e l'altra al server.

Per gestire le sessioni il motore PHP registra sul server un array associativo che può essere letto dalle pagine della sessione e che è associato ad un ID univoco sul server stesso; per quanto riguarda il client, crea sul computer dell'utente un cookie contenente lo stesso ID alfanumerico. Quando avviene così la chiamata HTTP, il server può verificare sul computer dell'utente la presenza di un cookie contenente un ID valido sul server e associare quindi ad esso i dati della sessione. In questo modo esisterà sempre un collegamento univoco tra server e client.
Nel caso l'utente abbia disabilitato i cookie, PHP consente al client di inviare l'ID della sessione appendendolo alla stringa di query oppure ai parametri di un form.

Gestione delle sessioni

La prima operazione che deve essere eseguita è quella di attivare il collegamento tra server e client e inizializzare quindi la sessione. Per fare ciò PHP mette a disposizione la funzione

```
session_start();
```

Si noti che è necessario inserire questa funzione in tutte le pagine che devono avere accesso alle variabili di sessione.
Esse sono contenute nell'array associativo `$_SESSION` e risulta molto facile impostare o ottenere il valore di una variabile di sessione:

```
$_SESSION[chiave] = valore;
```

imposta una variabile di sessione `chiave` dal valore `valore`.
Per richiamare una variabile basta usare la notazione:

```
$_SESSION[chiave]
```

che restituisce il valore della variabile di sessione `chiave`.

PHP mette inoltre a disposizione alcune funzioni relative alla sessione, come il nome, l'ID, eccetera. Sono:

```
session_name() //restituisce il nome della sessione
session_id() //restituisce l'ID
session_encode() //restituisce le variabili di sessione nella forma chiave|valore
```

Per chiudere la sessione, infine, PHP mette a disposizione la funzione:

```
session_destroy();
```

che deve sempre seguire session_start();.

Programmazione/Variabili globali/$ COOKIE

Cos'è

$_COOKIE (o **$HTTP_COOKIE_VARS** se si usa una versione di PHP precedente alla 4.1.0) è un array associativo contente tutti i cookie relativi al sito in questione.
I cookie sono dei piccoli file di testo che i siti web utilizzano per immagazzinare anche temporaneamente delle informazioni collegate all'utente in questione.

Utilizzo

Per impostare un cookie è possibile usare la funzione

```
setcookie(nome, valore, scadenza, percorso, dominio, sicuro, httponly);
```

dove `nome` e `valore` sono il nome e il valore del cookie. Sicuro e httponly sono due valori booleani, quindi accettano false o true.

Ecco un esempio di setcookie:

```
setcookie($name,$value,$expire,$path,$domain,$secure,$httponly);
```

Ovviamente tutte le variabili dovranno essere valorizzate, anche se non sono tutte obbligatorie.

Per specificare la scadenza in Unix Timestamp, è possibile usare la funzione `time()` sommando ad essa ad esempio 60*60*24; in questo modo il cookie scadrà dopo un giorno da quando è stato creato.
Per una descrizione più approfondita dei cookie e dei parametri, si veda questa pagina su Wikipedia

Per accedere ad un cookie memorizzato in precendenza dal nostro sito è possibile usare la notazione:

```
$_COOKIE[nome]
```

Programmazione/Variabili globali/$ SERVER

Cos'è

$_SERVER (o **$HTTP_SERVER_VARS** se la versione PHP è inferiore alla 4.1.0) è una delle variabili globali predefinite di sistema.

In sostanza è un *array associativo* di chiavi e valori i cui elementi sono rappresentati da informazioni riguardanti il lato server, il lato client e la connessione tra di essi.

Utilizzo

È possibile accedere agli elementi di questo array *iterando* su di essi con un ciclo `foreach` oppure reperire il singolo valore di un elemento se ne conosciamo la chiave. Nell'esempio seguente viene stampato l'indirizzo IP dell'utente:

```
$ip = $_SERVER["REMOTE_ADDR"];
print "Il tuo ip è $ip";
```

Si noti che alcune chiavi restituiscono o meno un valore a seconda dello stato del server e del client.
Ecco l'elenco delle chiavi in ordine alfabetico:

- **argc** il numero degli argomenti passati da linea di comando.
- **argv** l'array degli argomenti passati da linea di comando.
- **AUTH_TYPE** tipo di autenticazione.
- **DOCUMENT_ROOT** la cartella radice dello script definita nel file di configurazione del server.
- **GATEWAY_INTERFACE** la versione delle specifiche CGI usate dal server.
- **HTTP_ACCEPT**
- **HTTP_ACCEPT_CHARSET** tipo di carattere accettato.
- **HTTP_ACCEPT_ENCODING** il tipo di encoding accettato.
- **HTTP_ACCEPT_LANGUAGE** la lingua accettata, ad es. 'it'.
- **HTTP_CONNECTION**
- **HTTP_HOST**
- **HTTP_REFERER** se ne esiste uno contiene l'indirizzo della pagina precedente a quella attuale, utile per sapere da dove arriva chi accede al nostro sito.
- **HTTP_USER_AGENT** informazioni sul sistema operativo e browser del client, ad es.

```
Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
```

sono le informazioni lasciate dal bot di Google e

```
Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.10) Gecko/20050716 Firefox/1.0.6
```

per un utente che usa un sistema Linux e Mozilla Firefox

- **PATH_TRANSLATED**
- **QUERY_STRING** la querystring appesa, ottenibile anche con $_GET
- **REMOTE_ADDR** l'indirizzo IP del client.
- **REMOTE_HOST**
- **REMOTE_PORT** la porta usata dall'utente per effettuare la connessione.
- **REQUEST_METHOD** il tipo di richiesta fatto per accedere alla pagina, ad esempio 'GET' o 'POST'.
- **REQUEST_TIME** il timestamp all'inizio della richiesta (solo dalla versione 5.1.0 di PHP)
- **REQUEST_URI** la URI usata per accedere questa pagina.
- **SERVER_ADMIN** l'amministratore del server dal file di configurazione del server.
- **SERVER_NAME** il nome dell' host dove lo script viene eseguito.
- **SERVER_PORT** la porta usata dal server.
- **SERVER_PROTOCOL** il nome e la versione del protocollo tramite il quale è stata richiesta la pagina ad esempio 'HTTP/1.1'.
- **SERVER_SIGNATURE** la firma del server contenente versione e host name.
- **SERVER_SOFTWARE** la stringa di identificazione del server.
- **SCRIPT_FILENAME** il percorso assoluto dello script in esecuzione.
- **SCRIPT_NAME** il nome del file

Programmazione/File

PHP mette a disposizione numerose funzioni per leggere e/o modificare i file presenti sul server.
Esiste inoltre la possibilità di effettuare l'upload dei file dell'utente sul server stesso.

Aprire e chiudere i file

Per prima cosa, quando lavoriamo con un file, dobbiamo aprirlo usando la funzione `fopen` la cui sintassi è:

```
fopen(nomefile, modalità);
```

dove `nomefile` è l'indirizzo del file sul server oppure in Internet e `modalità` è una stringa che specifica il modo con cui si apre il file. La funzione restituisce un puntatore univoco al file aperto che servirà successivamente per eseguire le funzione legate alla lettura e alla scrittura del file.
La modalità può essere:

Modalità	Descrizione
r	Apre il file in modalità di sola **lettura**
r+	Apre il file in modalità di **lettura** e **scrittura**
w	Apre il file in modalità di sola **scrittura** cancellando il contenuto esistente. Se il file non esiste, PHP tenta di crearlo.
w+	Apre il file in modalità di **scrittura** e lettura cancellando il contenuto esistente. Se il file non esiste, PHP tenta di crearlo.
a	Apre il file in modalità di sola **scrittura** posizionando il puntatore alla fine del file per l'inserimento. Se il file non esiste, PHP tenta di crearlo.
a+	Apre il file in modalità di **scrittura** e **lettura** posizionando il puntatore alla fine del file per l'inserimento. Se il file non esiste, PHP tenta di crearlo.

Ad esempio:

```
$fp = fopen('/var/www/contatore.txt','w+');
$fp2 = fopen('./contatore.txt','w+');
$fp3 = fopen('http://esempio.it/file.txt','r');
```

Per indicare la directory corrente, si usa un punto (.). Per salire di un livello nella gerarchia, se ne usano due (..). Ad esempio:

```
$fp = fopen('./contatore.txt','w+');
```

Se il server è su sistem Windows, che usa le barre retroverse, sarà necessario commutarle:

```
$fp = fopen('C:\\data\\file.txt');
```

Per chiudere un file già aperto usiamo semplicemente la funzione

```
fclose(puntatore);
```

Leggere e scrivere un file

Per **leggere** un file si usa la funzione `fread` la cui sintassi è:

```
fread(handle, lunghezza)
```

e legge `lunghezza` byte dal file aperto identificato dal puntatore `handle`; dopodiché, sposta il puntatore del file di `lunghezza` byte in avanti.

Ad esempio:

```
$fp = fopen("file.txt", "r");
$contenuto = fread($fp, 10);
$cont2= fread($fp, 40);
```

Con questo breve spezzone, `$contenuto` conterrà i primi dieci byte del file e `$cont2` i byte dall'undicesimo al cinquantesimo.,

Un'altra funzione utile è `fgets`, che funziona come `fread`, con la differenza che interrompe la lettura dei dati anche quando incontra un carattere EOF (fine del file) o EOL (fine della riga); questo è utile nell'ambito della lettura di *stream* input/output.

Per **scrivere** su un file aperto in modalità di scrittura è possibile usare la funzione `fwrite` la cui sintassi è:

```
fwrite(handle, stringa, n);
```

e scrive sul file aperto e identificato da `handle` i primi `n` byte di `stringa`. Se `stringa` è minore di `n` byte, viene scritto il suo contenuto per intero. La funzione restituisce -1 in caso di errore.

Creare un contatore visite

Viste queste due semplici funzioni, possiamo ora creare un semplice contatore di visite nel nostro sito. Per evitare che il conteggio aumenti ad ogni *reload* della pagina, useremo delle variabili di sessione per verificare così se l'utente stava già visitando il sito prima di caricare la pagina.

```
session_start();
if ($_SESSION['entrato'] == false) {
 //incrementa le visite se è la prima volta che l'utente accede al sito in questa sessione
 $_SESSION['entrato'] = true;
 $fp = fopen("contatore.txt", "r+");

 if (!$fp) {
  //se il file non è stato aperto correttamente
    echo "Errore nell'apertura del file";
    exit; //esce dallo script PHP
 }

 $visite = (int) fread($fp, 10);
 $visite++;
 echo "Questo sito ha avuto $visite visite!";
 fwrite($fp, $visite);
 fclose($fp);
```

```
}
```

Altre funzioni

Per la lettura dei file esiste anche la funzione **`file`** la cui sintassi è:

```
file(nomefile)
```

e restituisce l'intero contenuto del file `nomefile` in un array in cui ciascun elemento è una riga contenente anche il carattere di ritorno a capo. Ad esempio:

```
$cont = implode("", file("file.txt"));
```

unisce tutte le righe restituendo quindi l'intero contenuto del file.
Per eliminare i caratteri di ritorno a capo è possibile usare la funzione **`trim`** che restituisce la stringa passata come argomento eliminando gli spazi bianchi agli estremi.
Per leggere un file e metterlo in una stringa basta fare

```
$contenuto=file_get_contents("nomedelfile");
```

Per contare le righe di un file si usa:

```
$righe=count(file("miofile"));
```

Può risultare inoltre utile la funzione **`stat`** la cui sintassi è:

```
stat(nome)
```

e restituisce un array associativo contenente alcune informazione sul file `nome`. Tra le chiavi dell'array vi sono:

Chiave	Descrizione
Dev	Numero del dispositivo
uid e gid	ID dell'utente e del gruppo proprietario
size	Dimensioni in byte
atime e mtime	Data dell'ultimo accesso e dell'ultima modifica

`basename``(file)` restituisce invece il nome del file passato come argomento:

```
echo basename('/home/gianni/documento.odt'); //restituisce documento.odt
```

Caricare file sul server

PHP mette anche a disposizione, se configurato correttamente, la possibilità di caricare sul server dei file forniti dall'utente. Questo è possibile usando un form HTML che abbia come metodo POST,

`enctype="multipart/form-data"` e che abbia tra i suoi campi un input con `type="file"`. Prendiamo ad esempio questo spezzone di codice:

```
<form method="post" enctype="multipart/form-data" action="caricafile.php">
Selezionare il file da caricare: <input type="file" name="file1" />
<input type="submit" value="carica" />
```

Nel file di destinazione avremo a destinazione un array `$_FILES` che conterrà le informazioni sui file caricati per ora in una directory temporanea sul server:

```
echo "Nome temporaneo del file sul server, assegnato da PHP: ".$_FILES['file1']['tmp_name']; //per esemp
echo "<br/>Nome del file sul disco rigido dell'utente: ".$_FILES['file1']['name']; //ad esempio C:\image
echo "<br/>Dimensione del file: ".$_FILES['file1']['size']; //ad esempio 2000
echo "<br/>Tipo di file (se fornito dal browser): ".$_FILES['file1']['type']; //per esempio, image/png
```

Si noti che il percorso del file sul server viene assegnato temporaneamente da PHP in una directory impostata nel file di configurazione di PHP.

Per poi copiare definitivamente il file sul server (quello temporaneo viene poi eliminato da PHP alla fine della sessione), usiamo la funzione `move_uploaded_file`:

```
$source = $_FILES['file1']['name'];
$dest = basename($_FILES['file1']['tmp_name']);
move_uploaded_file($source, "archivio/".$dest); //salva il file in una cartella apposita
```

Programmazione/Immagini

Tra le funzionalità del PHP esiste anche quella di creare dinamicamente immagini e restituire un output non quindi di testo (come può essere quello dell'HTML) ma di immagine.

Per fare ciò è possibile usare le librerie GD, delle librerie open-source sviluppate dalla Boutell. È possibile avere maggiori informazioni sull'argomento sul sito ufficiale (http://libgd.github.io/).

Prima di procedere quindi alla creazione di immagini con PHP, si verifichi ovviamente di avere installato le librerie GD sul proprio computer.

Per creare quindi una nuova immagine in PHP dovremmo quindi:

1. caricare in memoria una nuova immagine o una copia di un'immagine esistente
2. caricare (si dice *allocare*) i colori usati per le eventuali modifiche dell'immagine
3. eseguire eventuali modifiche (creare linee, punti, riempimenti, aggiungere testo...)
4. restituire come output un'immagine dopo aver correttamente impostato nell'header il tipo di file restituito
5. distruggere successivamente l'immagine, per liberare la memoria.

Creare una nuova immagine

Per la creazione di una nuova immagine PHP mette a disposizione diverse funzioni. Per creare un'immagine ex-novo usiamo la funzione:

```
imagecreatetruecolor(h, w)
```

che crea in memoria una nuova immagine di altezza `h` e larghezza `w` (in pixel) e restituisce un riferimento all'immagine appena creata. Esiste anche una funzione, seppur non raccomandata, che crea un'immagine con una minore ampiezza di colori: `imagecreate`, che ha lo stesso comportamento di `imagecreatetruecolor`.

Per caricare in memoria un'immagine salvata su disco usiamo invece le funzioni

```
imagecreatefrom<tipo> (percorso)
```

che carica un'immagine salvata sul server del tipo specificato. Ad esempio:

```
$img = imagecreatefrompng('immagine.png');
```

Esiste inoltre una funzione

```
imagecreatefromstring (testo)
```

che crea un'immagine contenente il testo specificato come argomento

Nei casi avvenga un errore, le funzioni restituiscono `false`.

Lavorare con i colori

Per allocare un colore si usa la funzione

```
imagecolorallocate(immagine,r,g,b)
```

che restituisce un riferimento al colore date le sue componenti RGB.

È possibile, se si sta lavorando con formati come il PNG (che supporta la trasparenza), allocare colori trasparenti con la funzione `imagecolortransparent`

```
imagecolortransparent(immagine,colore)
```

dove `colore` è una risorsa colore valida creata con `imagecolorallocate`; è anche possibile allocare colori con una determinata trasparenza, compresa tra 0 (opaco) e 127 (completamente trasparente) con la funzione

```
imagecolorallocatealpha(immagine,r,g,b,trasparenza)
```

Si noti che il primo colore allocato verrà automaticamente usato come colore di sfondo dell'immagine.

Creata la nuova immagine e allocati i colori, è ora possibile lavorarci sopra in due modi:

- disegnando nuovi pixel (creando linee, forme, ecc...)
- lavorando sui pixel già esistenti (serve se l'immagine è stata caricata da un file) tagliandone parti, ricolorandola o ridimensionandola)

Disegnare punti, linee e forme

Per disegnare un pixel usiamo la funzione

```
imagesetpixel(immagine, x, y, colore)
```

dove `immagine` è una risorsa di immagine , `x` e `y` sono le coordinate del punto da disegnare e `colore` è una risorsa che identifica un colore allocato in precedenza.

```
imageline(immagine, x1, y1, x2, y2, colore)
```

Traccia una linea dal punto x1, y1 al punto x2, y2

```
imagerectangle(immagine, x1, y1, x2, y2, colore)
```

Disegna un rettangolo che ha per diagonale la linea da x1, y1 a x2, y2.

```
imageellipse(immagine, x, y, w, h, colore)
```

Disegna una ellisse di centro x, y di altezza h e larghezza w. Se h è uguale a w, allora si otterrà un cerchio.

```
imagearc(immagine, x, y, w, h, ang1, ang2, colore)
```

Funzione come `imageellipse` ma disegna solo l'arco di ellisse compreso tra gli angoli (in gradi) ang1 e ang2, I gradi sono contati in senso orario a partire dalle ore 3.

Lavorare sui pixel già esistenti

Per lavorare su un'immagine già disegnata in precedenza, ad esempio una foto, si hanno a disposizione molte funzioni.

Una delle più utilizzate è sicuramente `imagecopyresized`, che permette di copiare una porzione rettangolare di un'immagine ed incollarla in un'altra (con possibilità di ridimensionarla durante il processo). Es:

```
imagecopyresized(dst_image, src_image, dst_x, dst_y, src_x,
src_y, dst_w, dst_h, src_w, src_h);
```

dove:

- `dst_image` è l'immagine dove verrà incollata
- `dst_x, dst_y` sono le coordinate X e Y di `dst_image` dove verrà incollata la porzione di immagine;
- `src_x, src_y` sono le coordinate X e Y di `src_image` che corrispondono all'angolo in alto a sinistra del rettangolo da copiare;
- `dst_w, dst_h, src_w, src_h` sono invece rispettivamente larghezza e altezza dell'immagine rettangolare che sarà incollata e larghezza e altezza dell'immagine rettangolare da copiare.

Si può quindi capire che se dst_w è uguale a src_w e dst_h è uguale a src_h, la porzione rettangolare dell'immagine resterà della stessa misura, in caso contrario l'immagine risulterà allungata e/o allargata.

La funzione `imagecopyresampled` riceve gli stessi parametri di `imagecopyresized`, con la differenza che, in caso di ridimensionamento, la qualità è migliore.

Esiste poi la funzione `imagefilter`, che permette numerosi effetti quali la scala di grigio, l'incassato, la ricolorazione: per la sua complessità, rimando al manuale ufficiale (http://it.php.net/manual /it/function.imagefilter.php), dove è possibile trovare funzionamento ed esempi.

Stampare l'output

Prima di stampare l'output del risultato ottenuto è necessario indicare, usando la funzione `header`, il content-type (tipo del contenuto, che per default è impostato a text/html); quindi:

```
header("Content-type: image/<tipo>");
```

dove `tipo` sta per "png", "jpeg" o "gif" a seconda del formato in cui si vuole visualizzare l'immagine.

Per visualizzare l'immagine, secondo il tipo scelto in precedenza, usare la funzione `imagepng`, `imagejpeg` o `imagegif`, che prendono come parametro la risorsa immagine da visualizzare.

Infine occorre liberare la memoria, che è stata occupata dall'immagine, con la funzione `imagedestroy`, che prende come unico parametro la risorsa immagine da distruggere.

Sebbene l'omissione di questa procedura non provochi la visualizzazione di errori da parte di PHP, è fortemente consigliata soprattutto quando si usano immagini piuttosto grandi.

Programmazione/Regexp

Le **espressioni regolari** consentono complesse elaborazioni testuali.

In questa sede ci limiteremo ad illustrare il funzionamento delle regex su PHP senza però spiegare effettivamente come si scrivono le *regular expressions* (cosa infatti non semplice e di certo impossibile da fare in una pagina).

Per le espressioni regolari sono state scritte varie funzioni, quelle più famose sono la coppia _ereg e il **package PCRE** (acronimo di "*Perl Compatible Regular Expressions*" ovvero Regex compatibili con Perl). Le prime tuttavia non hanno le stesse funzionalità avanzate del package PCRE, pertanto analizzeremo queste ultime in particolare (se però si vuole avere una visione anche sulle prima, basta visitare questo link (http://www.regular-expressions.info/php.html)).

PCRE

Questa libreria è un *porting* in PHP di un'altra molto utilizzata e diffusa, scritta per Perl. Essa viene anche chiamata colloquialmente *preg_match*.

Ottenere dei dati

Vediamo un esempio di come si utilizza una regex in PHP:

```php
<?php
$regex = "/\[\[Ii]m(?:age|magine):(.*?)\]\]/";
$testo = "[[Immagine:Ciao.jpg]], [[image:hello.jpg]], image:hello.jpg";
preg_match($regex, $testo, $risultato);
print_r($risultato);
?>
```

Se proviamo ad eseguire questo script, otterremo come risultato:

```
Array
(
    [0] => [[Immagine:Ciao.jpg]]
    [1] => Ciao.jpg
)
```

La regex utilizzata serve per trovare tutte le immagini in markup wiki; in particolare restituisce il suo nome reale (ovvero, senza il namespace immagine).

In PHP una regex deve essere definita in una stringa di testo che inizia e finisce con "/" per far capire al motore che quella è una regex e non testo qualunque. La funzione `preg_match` prende il primo parametro (la regex), lo confronta col secondo (il testo) e assegna il risultato alla variabile indicata come terzo parametro (nel nostro caso `$risultato` , che noi mostriamo con la funzione `print_r`).

La variabile di risultato conterrà il testo associato dall'intera regex e l'evenutale testo dei cosidetti *backreference* (http://en.wikipedia.org/wiki/Backreference#backreferences) indicati tra le parentesi (nel nostro caso rispettivamente $risultato[0] e $risultato[1]). La prima parentesi non viene restituita come *backreference* solo perché è stato utilizzato il modificatore `?:` che indica al motore delle regex di non contare quelle determinate parentesi per le *backreference* (ciò serve per rendere la regex più veloce).

Perché, però, non prende anche image:hello? Perché preg_match restituisce solo la prima occorrenza, per ottenerle tutte bisogna usare preg_match_all:

```php
<?php
$regex = "/\[\[Ii]m(?:age|magine):(.*?)\]\]/";
$testo = "[[Immagine:Ciao.jpg]], [[image:hello.jpg]], image:hello.jpg";
preg_match_all($regex, $testo, $risultato);
print_r($risultato);
?>
```

Risultato:

```
Array
(
    [0] => Array
        (
            [0] => [[Immagine:Ciao.jpg]]
            [1] => [[image:hello.jpg]]
        )

    [1] => Array
        (
            [0] => Ciao.jpg
            [1] => hello.jpg
        )

)
```

In questo modo abbiamo quindi sia "il pezzo di codice" che volevamo passare al *parser* che il risultato nello specifico.

Abbiamo quindi due array annidati dentro un altro, pertanto per accedere al primo basterà usare $risultato[0], mentre $risultato[1] per il secondo. Per accedere invece, per esempio, a Ciao.jpg, basterebbe utilizzare $risultato[1][0].

Sostituire dei dati con degli altri (replace)

Poniamo per esempio di dover cambiare tutti le stringhe di wikicodice [[Image:x]] con [[Immagine:x]] per ovviare a degli ipotetici problemi tecnici del motore wiki.

```
<?php
$testo = "[[image:ciao.png]], [[Image:CIAO.jpeg]]";
$regex = "/\[\[Ii]mage:(.*?)\]\]/";
$regex2 = "[[Immagine:\\1]]";
$risultato = preg_replace($regex, $regex2, $testo);
print $risultato;
?>
```

Il risultato di questo script è:

```
[[Immagine:ciao.png]], [[Immagine:CIAO.jpeg]]
```

In questo modo abbiamo definito una prima regex per ottenere le sottostringhe e poi una seconda per indicare con cosa sostituire il testo associato dalla prima. Dato che la seconda è una stringa e la *backslash* (\) è un carattere speciale in PHP, bisogna farne l'*escape*(aggiungendo quindi un'altra backslash) per ottenere un *backreference* (riferimento all'indietro, ovvero andare a prendere la parentesi identificata dal numero).

Attenzione:

1. `preg_replace` restituisce un valore tramite assegnamento (ovvero, dovete usare la sintassi `$variabile = preg_replace();`) al contrario di quanto faceva `preg_match[_all]`)
2. Questa regex non è la migliore in quanto il tutto potrebbe essere fatto semplicemente sostituendo "/\[\[Ii]mage:/" con "[[Immagine". La parte superflua è stata tuttavia inserita ad esclusiva finalità didattica (per spiegare come usare le *backreference* correttamente).

Modificatori

Modificatore	Descrizione
i	Rende l'espressione *case-insensitive*.
m	Permette la modalità multi-riga, così ^ si ancora all'inizio di ogni frase e $ alla fine sempre di ogni frase.
s	Permette la modalità mono-riga, così ^ si ancora all'inizio del testo e $ alla fine (di tutto il testo). Il metacarattere . può essere utilizzato per catturare anche gli a capo.
x	Modalità estesa, gli spazi dentro la regex fuori da espressioni vengono ignorati, permette inoltre di commentare le regex con #.
e	La stringa (la `$regex2` usata sopra) usata in preg_replace viene elaborata come codice PHP.
A	Àncora la regex per forza all'inizio del testo.
D	Il carattere $ àncora solo a fine testo.
u	Modalità Unicode.

I modificatori servono a modificare il comportamento del motore di regex utilizzato. Per esempio, la regex usata sopra per *italianizzare* tutte le immagini, diventerebbe:

```
<?php
$testo = "[[image:ciao.png]], [[Image:CIAO.jpeg]]";
$regex = "/\[\[image:(.*?)\]\]/i";
$regex2 = "[[Immagine:\\1]]";
$risultato = preg_replace($regex, $regex2, $testo);
print $risultato;
?>
```

Con risultato: [[Immagine:ciao.png]], [[Immagine:CIAO.jpeg]]

Come si può vedere, il risultato è identico sebbene l'espressione regolare non sia la stessa (notare in particolare la **i** dopo l'ultimo slash - `/` - della regex).

Particolarità

Le backreference (quelle di solito identificate come \1 o \\1 in php) possono anche essere richiamate con $1 (sostituendo l'1 al numero desiderato) come si fa in Perl.

Bibliografia

- Apogeo, Espressioni Regolari, Marco Beri (http://www.apogeonline.com/libri/88-503-2665-3/scheda).

Collegamenti esterni

- (**EN**) www.regular-expressions.info (http://www.regular-expressions.info/) esempi, tutorial e guide sulle regex.
- Guida in italiano a regex di base (http://fido.altervista.org/RegExp/regex.html).

Programmazione/OOP

La **programmazione object oriented** (o **Programmazione orientata agli oggetti**, abbreviata spesso con **OOP**, ***object oriented programming***) è uno stile, un paradigma di programmazione che consiste nel modellare un problema definendo degli oggetti che interagiscono fra di loro. Un oggetto è un insieme logico costituito da dati e da funzioni che manipolano i dati stessi e creano *l'interfaccia* tramite la quale l'oggetto interagisce con gli altri oggetti.

In questo capitolo tratteremo di programmazione orientata agli oggetti in PHP 5. Una delle maggiori novità di PHP 5, infatti, è proprio il nuovo *object model* che rende PHP un linguaggio realmente *object oriented* (orientato agli oggetti). Infatti, sono proprio queste nuove caratteristiche, che analizzeremo nel prosieguo del capitolo, che hanno permesso lo sviluppo di tecniche aderenti ai pattern di progettazione più moderni.
Inoltre, come annunciato sul sito ufficiale (http://it.php.net), PHP 4 è ormai giunto al tramonto e non sarà più supportato a partire dall'agosto 2008.
Pertanto, per quanto riguarda l'object model di PHP 4, rimandiamo alla documentazione ufficiale (http://it.php.net/manual/it/language.oop.php) (in italiano)

Le classi

Una **classe** definisce una *categoria* di oggetti, aventi tutti le stesse caratteristiche. In effetti, il concetto di classe precede quello di oggetto:

1. prima penso ad una classe e ne definisco i dati, detti **attributi** (i quali in realtà sono delle variabili), e le funzioni che li manipolano, dette **metodi**
2. in seguito ***istanzio*** un oggetto a partire dalla classe: possiamo pensare alla classe come ad uno "stampino" per creare oggetti e ad un oggetto come ad un'entità "concreta" della classe. Per creare le istanze di ciascun oggetto è comodo utilizzare un **costruttore**, ovvero una funzione che imposti fin dall'inizio alcuni attributi fondamentali della classe.

Facciamo un esempio mentale. La classe "cane" definisce, nella nostra mente, il miglior amico dell'uomo, con tutte le sue caratteristiche, cioè gli *attributi* (per es. la razza, il colore del manto). L'oggetto "Lassie" è un'istanza della classe cane in cui l'attributo razza è Collie, il colore del manto è bianco e marrone, ecc....

Per creare una classe in PHP è sufficiente utilizzare la sintassi:

```
class NomeClasse {
 //qui dentro vanno inserite le variabili pubbliche (gli attributi) e private della classe
 //e le funzioni (metodi)
 //questa funzione è il costruttore della classe e si chiama sempre così
 function __construct ($parametri) {
  //istruzioni...
 }

} //qui finisce la dichiarazione della classe
```

Il costruttore può essere anche una funzione con lo stesso nome della classe.

In questo modo è stata creata una classe NomeClasse. Per crearne una nuova instanza, sarà sufficiente scrivere:

```
$var = new NomeClasse ($parametri_del_costruttore) //instanzia un nuovo oggetto della classe NomeClasse
```

Con questa istruzione creiamo un nuovo oggetto dallo "stampino" NomeClasse; i parametri passati tra parentesi sono quelli richiesti dalla funzione `__construct` (se prevista).
Per accedere agli attributi o ai metodi della classe si userà la sintassi:

```
$var->attributo = $a; //posso leggere o modificare gli attributi della classe
$b = $var->esempioMetodo(); //un metodo può restituire un valore
```

Quando si lavora con le classi, la cosa più comoda è creare un file `class.NomeClasse.php` in cui inserire il codice della classe e poi richiamare tale file negli script in cui si desidera lavorare tramite l'istruzione `include_once`:

```
include_once("class.NomeClasse.php");
```

Per accedere all'attributo dall'interno della classe si usa `this`, per accedervi dall'esterno: "il nome dell'oggetto `->` e il nome del metodo":

```
class prova{
   private $a="mamma";

   public function stampa(){
```

```
      echo $this->a;
   }
}

$b=new prova();
$b->stampa();
```

Come si vede la funzione stampa accede alla variabile privata a tramite la keyword `this` e `->`, si noti il `$` la keyword `this` diventa variabile e prende il `$` che non va applicato al nome della variabile privata!
Per accedere al metodo `stampa()` usiamo il costrutto: "il nome oggetto `->` e il nome metodo". Potremmo accedere anche ad una variabile della classe nello stesso modo (vedi ultima riga):

```
class prova{
   private $a="mamma";
   public $c="gatto";

   public function stampa(){
      echo $this->a;
   }

   public function get_var(){
      return $this->a;
   }
}

$b=new prova();
$b->stampa();
$variabile_privata=$b->get_var();
echo $b->c;
```

se invece vogliamo ottenere la variabile privata, il metodo `get_var()` accede per noi alla variabile privata e ne restituisce il valore, che verrà salvato nella variabile `$variabile_privata`.

Ereditarietà

Una classe può estenderne un'altra ereditandone così i metodi e gli attributi. La classe estesa si chiama *superclasse* ed è estesa da una *sottoclasse*. La sottoclasse può sovrascrivere (*overriding*, da non confondere con l'overloading) i metodi della superclasse definendone di propri, per estendere una classe è sufficiente, dopo il nome della classe estendente, mettere la parola chiave `extends` ed il nome della classe da estendere.

```
class frutta{
   public function visualizza(){
      echo "questo è un frutto";
   }
}

class mele extends frutta{
   public function visualizza(){
      echo "questa è una mela";
   }
}

class pere extends frutta{
   public function visualizza(){
      echo "questa è una pera";
   }
}

$fuji=new mele();
$fuji->visualizza(); //scriverà questa è una mela

$williams= new pere();
$williams->visualizza();//scriverà questa è una pera

$ciquita= new frutta();
$ciquita->visualizza();//visualizzerà questo è un frutto
```

Come si vede il metodo `visualizza()` è sovrascritto dalle varie sottoclassi: pere e mele. Bisogna prestare particolare attenzione con gli indicatori di visibilità, come vedremo fra poco.

Polimorfismo

Nell'esempio sopra abbiamo una chiara dimostrazione di polimorfismo, il metodo `visualizza()` è chiamato più volte ma ogni volta dà un risultato differente a seconda che l'oggetto appartenga ad una od un'altra classe. Se richiamo visualizza() con fushi che è un oggetto della classe mele, ecco che visualizzerà "questa è una mela", mentre se lo richiamo con williams. visualizzerà "questa è una pera", il polimorfismo è quindi la capacità di uno stesso metodo di eseguire compiti differenti in base alla classe dell'oggetto di appartenenza, infatti il metodo richiamato è sempre visualizza() ma si comporterà in modo diverso a seconda della classe dell'oggetto.

Ambito

Abbiamo detto che un attributo non è altro che una variabile della classe; bisogna tuttavia fare attenzione all'ambito dell'attributo, ovvero alla visibilità che ha l'attributo dall'esterno.

- **public** rende l'attributo o il metodo visibile e utilizzabile anche dall'esterno della classe
- **protected** l'attributo o il metodo possono essere visti/usati solo dall'interno della classe o da una sottoclasse
- **private** l'attributo o il metodo può essere visto/usato solo all'interno della classe.

Se si usa la dichiarazione deprecata di php<5: `var`, senza specificare l'ambito dell'attributo, questo avrà come ambito per default **pubblico**

```php
<?php
class prova{
   private $priv="ciao";
   protected $prot="miau";
   public $publ="bau";

   var $var="fuffi";

   function stampa(){
      echo "dalla funzione stampa accedo alla varibile privata= " . $this->priv . "<br>";
   }
}

class altro extends prova{
   function ristampa(){
      echo "dalla class estendente accedo alle variabili protected della classe estesa= " . $this->p
   }
}

$a=new prova();
//echo "a priv " . $a->priv . "<br>";
//echo "a prot " . $a->prot . "<br>";
echo "a publ " . $a->publ . "<br>";
echo "var= " . $a->var . "<br>";
$a->stampa();


$b=new altro();
//echo "b priv " . $b->priv . "<br>";
//echo "b prot " . $b->prot . "<br>";
echo "b publ " . $b->publ . "<br>";
$b->ristampa();

?>
```

Se decommentiamo (togliendo i caratteri "//") la riga 21, il motore php restituirà il seguente errore:
`PHP Fatal error: Cannot access private property prova::$priv in /path_del_file/test.php on line 14, referer: http://localhost/`

La stessa cosa succede se ricommentiamo la riga 21 (rimettendo i caratteri "//") decommentiamo la riga 22: questo perché stiamo richiamando la variabile dall'esterno della classe. Come si vede invece dalla riga 25, il metodo `stampa()` viene richiamato dalla variabile `a` che è istanza della classe prova. Essendo il metodo stampa() parte della classe prova, esso può accedere alla variabile privata `priv`.
Decommentando la riga 29 otterremo un effetto che può sembrare particolare, anziché dare errore il motore stamperà `b priv` e tutto quello che segue questa riga. Infatti la variabile privata priv non viene estesa ed il motore non restituisce un errore fatale, bensì:
`Undefined property: altro::$priv in /path/test.php on line 29, referer: http://localhost/`

La riga 32 mostra che il metodo ristampa può accedere alla variabile protetta `prot`, infatti tale variabile è estesa alla classe `altro`.

Classi astratte

Le *classi astratte* definiscono delle linee guida per i metodi che le classi che estendono dovranno seguire. Non definiscono il corpo del metodo, bensì il nome, la visibilità, i parametri, cioè quella che viene definita la *signature*. Il motivo è semplice, si pensi ad un programma sviluppato da più gruppi di sviluppatori, un gruppo definisce la classe madre, mentre gli altri scrivono delle sottoclassi ridefinendo alcuni metodi. Se i gruppi non hanno una buona intercomunicabilità potrebbero ridefinire a piacere i metodi della superclasse cambiando i parametri o la visibilità dello stesso metodo e per evitare ciò esistono le classi astratte.

Se una classe ha anche solo un metodo astratto è obbligatorio definirla come astratta, se una classe eredita da una classe astratta è obbligatorio ridefinirne tutti i metodi astratti. In PHP non è possibile dichiarare un metodo come astratto e definirlo nella stessa classe come permettono alcuni linguaggi OOP.

Una classe o un metodo si dichiarano astratti con la keyword `abstract`:

```
class abstract prova {
  public $var="ciao";
  abstract public function miometodo($var1, $var2);
}

class sotto extends prova {
  public function miometodo($var1, $var2){
    echo "$var1 $var2";
}
```

Bisogna mantenere inalterata la signature e cioè nome namespace parametri, si può cambiare solo la visibilità, restringendola. Una classe astratta **non può essere istanziata direttamente**.

Incapsulamento

Si definisce incapsulamento la tecnica di rendere invisibile ed inaccessibile parti di codice non necessario per l'utilizzo di una classe, rendendo accessibili e visibili solo alcuni metodi e alcuni attributi. Si ottiene grazie agli indicatori di visibilità e alle interfacce.

Interfacce

Le interfacce sono delle ulteriori astrazioni delle classi astratte. Vengono definite tramite la keyword `interface` seguita dal nome voluto per l'interfaccia e le graffe.

```
interface miainterfaccia{
   public function setName($name);
   public function getName();
}
```

In pratica è come dichiarare una classe astratta, con l'agevolazione di non dover dichiarare tutto `abstract`. Una classe che implementi tale interfaccia deve ridefinire tutti i metodi obbligatoriamente pena errore fatale, esattamente come per le classi astratte. Vediamo l'implementazione:

```
class miaclasse implements miainterfaccia{
   public $name;

   public function setName($name){
      $this->name=$name;
   }

   public function getName(){
      return $this->name;
   }
}
```

Come per le classi astratte la classe implementante deve ridefinire tutti i metodi, preservando la signature tranne per l'ambito che può essere ristretto.

Namespaces

```
<?php
namespace my\name;

class MyClass {}
function myfunction() {}
const MYCONST = 1;

$a = new MyClass;
$c = new \my\name\MyClass;
$d = new \globalClass;
?>
```

PEAR

PEAR è l'acronimo per *PHP Extension and Application Repository*. Con questa sigla si identifica il progetto della comunità di sviluppatori PHP che si pone come scopo fornire principalmente:

- una libreria strutturata di classi open-source per sviluppatori PHP
- un sistema per la distribuzione del codice e per la manutenzione dei pacchetti
- uno standard per lo stile del codice scritto in PHP

Gli standard PEAR

Essendo PEAR un progetto della comunità di sviluppatori è nata la necessità di definire uno standard per lo stile della scrittura del codice PHP, in modo tale da rendere il codice leggibile e comprensibile da tutti.

Gli standard PEAR, tra le altre cose, stabiliscono che:

- il codice PHP sia sempre delimitato dai tag <?php e ?>
- al posto del carattere tabulazione vengano usati quattro spazi
- vengano incluse, anche dove sono facoltative, le parentesi, per rendere il codice più chiaro
- le righe bianche vengano usate solo per separare blocchi di codice distinti
- tutti i blocchi delle strutture di controllo (if, for, ecc...) siano racchiusi da parentesi graffe anche quando questo sia facoltativo

Sono state stabilite anche alcune convenzioni di nomenclatura:

- i nomi delle classi devono essere auto-esplicativi ed ogni parola che compone il nome deve avere l'iniziale maiuscola ed il resto minuscolo; le parole devono essere inoltre delimitate da undescore: per esempio, sono nomi di classi corretti Log o Net_Finger ma non lo è; uploadError
- le funzioni definite dall'utente devono essere chiamate con la prima letta minuscola e la prima lettera di ogni parola che compone la funzione maiuscola: ad esempio, miaFunzione() è un nome valido mentre non lo sono Miaaltrafunzione() e miaaLTRAFUNZIONE().

Le classi PEAR

Le classi PEAR permettono quindi di semplificare il lavoro del programmatore accedendo a soluzioni di ottima qualità già realizzate da altri programmatori. Ad esempio, tramite la classe **DB_DataObject** si è in grado di accedere ai dati di un database senza dover costruire delle query, ma accedendovi semplicemente ed intuitivamente tramite i metodi della classe.

Database

PHP e i Database

Questa sezione spiega le interazioni che ci sono fra il PHP e i diversi tipi di Database.

Programmazione/MySQL

PHP, come già specificato, può accedere a database. Il tipo di database più utilizzato sul web è MySQL.

Connessione ad un server mysql e selezione del database

Per la connessione al server, PHP ha una funzione specifica: mysql_connect(), la cui sintassi è:
mysql_connect(host_db, username, password);

Esempio:

```
<?php
     mysql_connect("localhost","tuousername","tuapassword")
     or die("Errore nella connessione MySQL");
?>
```

Con questo codice, PHP tenta la connessione a localhost con l'username e la password forniti, in caso di fallimento, stampa il messaggio di errore.

Con questa funzione il PHP si connetterà a MySQL e sarà pronto per lavorare. Per selezionare il database su cui dobbiamo eseguire le nostre query abbiamo la funzione mysql_select_db: *mysql_select_db(nomedb, [puntatore]);*

Esempio:

```
<?php
     $db = mysql_connect("localhost", "tuousername", "tuapassword")
         or die("Errore nella connessione MySQL");
     mysql_select_db("test", $db) or die("Database inesistente");
?>
```

Adesso PHP tenta la connessione al database test dal server localhost al quale ci siamo connessi prima. Nel caso il database non esistesse o in caso di errore, verrebbe inviato il messaggio "Database inesistente". Come potete vedere, in questo caso la funzione mysql_connect() è stata assegnata alla variabile $db, che in questo caso diventa un puntatore di risorse. In questo modo, possiamo aprire più connessioni contemporanee assegnate a diversi puntatori.

Esecuzione di query

Dopo aver aperto il database, possiamo eseguire delle operazioni con i dati presenti al suo interno (vedi MySQL), come la creazione o eliminazione di tabelle o inserimento e richiesta di dati. Per effettuare una richiesta dati MySQL al server si utilizza la funzione mysql_query():

```
<?php
     $db = mysql_connect("localhost", "tuousername", "tuapassword")
         or die("Errore nella connessione MySQL");
     mysql_select_db("test", $db) or die("Database inesistente");
     if (mysql_query("SELECT * FROM registrati",$db)) {
         echo "Query eseguita con successo";
     } else {
         echo "Errore nell'esecuzione della query: ".mysql_error();
     }
?>
```

Questo codice, recupera tutti i record della tabella registrati del database test sul server localhost. In caso di errore verrà visualizzato un messaggio contenente la descrizione dell'errore. È utile inserire la funzione mysql_query sempre in un puntatore (diverso da quello del database), per l'utilizzo dei dati

Funzioni PHP - MySQL

Nel caso in cui avessimo bisogno di prendere dei dati da un database dovremo utilizzare la funzione mysql_fetch_array() che crea un array con indice, i nomi delle colonne del database e come dati *il primo dell'elenco dei risultati della query*. Supponiamo di avere una tabella così strutturata:

Nome	Cognome	Data_nascita	Città
Tizio	Rossi	20/11/1957	Milano
Caio	Bianchi	12/03/1985	Roma
Sempronio	Verdi	08/06/1967	Napoli

Con questo codice:

```php
<?php
    $db = mysql_connect("localhost", "tuousername", "tuapassword")
        or die("Errore nella connessione MySQL");
    mysql_select_db("test", $db) or die("Database inesistente");
    $query = mysql_query("SELECT * FROM registrati",$db);
    $risultato = mysql_fetch_array($query);
?>
```

verrà creato un array $risultato contenente solo una riga della tabella strutturato così:

- $risultato['Nome'] = "Tizio"
- $risultato['Cognome'] = "Rossi".
- $risultato['Data_nascita'] = "20/11/1957".
- $risultato['Città'] = "Milano".

Per vedere tutte le righe della tabella, bisogna fare così:

```php
<?php
    $db = mysql_connect("localhost", "tuousername", "tuapassword")
        or die("Errore nella connessione MySQL");
    mysql_select_db("test", $db) or die("Database inesistente");
    $query = mysql_query("SELECT * FROM registrati",$db);
    while($riga = mysql_fetch_array($query))
    {
     $risultato[]=$riga;
    }
?>
```

e allora la variabile $risultato sarà:

- $risultato[0]
  - $risultato[0]['Nome'] = "Tizio"
  - $risultato[0]['Cognome'] = "Rossi".
  - $risultato[0]['Data_nascita'] = "20/11/1957".
  - $risultato[0]['Città'] = "Milano".
- $risultato[1]
  - $risultato[1]['Nome'] = "Caio"
  - $risultato[1]['Cognome'] = "Bianchi".
  - $risultato[1]['Data_nascita'] = "12/03/1985".
  - $risultato[1]['Città'] = "Roma".
- $risultato[2]
  - $risultato[2]['Nome'] = "Sempronio"
  - $risultato[2]['Cognome'] = "Verdi".
  - $risultato[2]['Data_nascita'] = "08/06/1967".
  - $risultato[2]['Città'] = "Napoli".

Programmazione/MySQL/Accesso al database

L'**accesso al database** può essere effettuato con due funzioni simili: `mysql_connect` e `mysql_pconnect`. La differenza saliente fra le due è che la seconda, rispetto alla prima, crea una connessione di tipo permanente. Una connessione permanente viene aperta e tenuta sempre in vita, mentre una non permanente viene svegliata soltanto nel momento in cui devono essere inviate query string e ricevuti dataset.

Le funzioni di connect accettano quattro parametri:

- `hostname[:port]` (string): il nome dell'host (o il relativo IP) eventualmente seguito dal numero di porta su cui risponde il server MySQL
- `username` (string): l'utente MySQL
- `password` (string): la password definita per l'utente specificato
- `new_connect_flag` (boolean): se questo parametro è impostato 'true' si intende che, nella situazione in si esegua una connect con parametri già utilizzati precedentemente nello script, verrà creato un nuovo identificatore di risorsa, invece di ritornare quello creato precedentemente.

Importante:

1. utilizzate con diligenza la mysql_pconnect in quanto, se dimenticare di chiudere la connessione, questa rimarrà aperta anche dopo che lo script ha terminato la sua esecuzione
2. non utilizzare il parametro new_connection_flag se non è estremamente necessario in quanto i server MySQL sono di solito configurati per gestire circa 30-100 connessioni

Entrambe le funzioni ritornano un intero che definisce la risorsa (*resource identifier*). Questo valore dovrebbe essere salvato in una variabile e passato a tutte le funzioni che lo richiedano come parametro. Di fatto il parametro è opzionale in quanto il modulo per MySQL di PHP tiene traccia dei parametri di connessione utilizzati (se si aprono più connessioni memorizza i più recenti) ed esegue le operazioni necessarie per tenere aperta una connessione ad ogni invio di query.

Una volta aperta la connessione è possibile, ma non necessario, scegliere il database da utilizzare chiamando la funzione `mysql_select_db`, pseudonimo della query "USE DATABASE my_little_db".

Per i programmatori più esperti e negli ambiti di applicazioni a carattere enterprise, un consiglio è quello di utilizzare uno stereotipo poco in voga ma veramente utile. È necessario creare un file di init seguendo l'esempio

```
[DB_MASTER_CONFIG]
host = mysqlsrv.dominio.it:3306
usr  = root
pwd  = pwd_for_root
db   = application_db
```

Inserite nel vostro script (preferibilmente nel file delle configurazioni, come è consono per applicazioni di grosse dimensioni) il seguente segmento di codice

```
$DBCONF = parse_ini_file('/my_path/my_init_file.ini') ;
$host = $DBCONF['host'] ;
$usr  = $DBCONF['usr'] ;
$pwd  = $DBCONF['pwd'] ;
$db   = $DBCONF['db'] ;
$link_id = mysql_connect( $host, $usr, $pwd )
  or die("Errore nella connessione al mysql server");
mysql_select_db( $db, $link_id )
  or die("Errore nella selezione del db: ".mysql_error($link_id)) ;
```

Programmazione/MySQL/Query

L'invio di una query MySQL in PHP avviene tramite la funzione `mysql_query` la cui sintassi è:

```
mysql_query(query, connessione)
```

dove *query* è una stringa contenente il testo della query MySQL da eseguire e *connessione* è un puntatore di risorsa aperta con la funzione `mysql_connect` e connessa ad un database. A seconda del tipo di query inviata al database, la funzione mysql_query restituisce valori differenti:

- se la query è di inserimento restituisce un puntatore al fieldset ottenuto dall'esecuzione della query (ad esempio una query come `"SELECT * FROM registrati"` dell'esempio precedente)
- negli altri casi la query restituisce un valore boolean che indica l'esito positivo o meno della query

Nell'invio di una query è necessario, se non viene già fatto dal server MySql, convertire i caratteri come gli apostrofi con la loro commutazione \', in quanto potrebbero dare problemi per quanto riguarda l'invio di stringhe alfanumeriche, tramite la funzione `addslashes`, la cui sintassi è

```
addslashes(stringa_da_commutare)
```

e restituisce la stringa stessa con i caratteri speciali commutati. La sua funzione inversa è

```
stripslashes(stringa_già_commutata)
```

È consigliabile inoltre prevedere controlli per evitare intrusioni indesiderate da parte di hacker, evitando ad esempio che eventuali campi di input (per esempio nome utente o password) contengano spazi, i quali permetterebbero l'esecuzione di JOIN o query multiple.

Programmazione/MySQL/Risultati di una query

Una volta ottenuto il puntatore al risultato della query tramite la funzione mysql_query è possibile procedere all'uso del result-set. Per fare ciò PHP mette a disposizione numerose funzioni:

- la più usata è `mysql_fetch_array(risultato, tipo_array)` che restituisce l'i-esimo record del fielset e incrementa di uno l'indice, dove *risultato* è un puntatore di fieldset MySQL. In base al parametro *tipo_array* la funzione restituisce valori differenti:
  - `MYSQL_ASSOC`: il risultato della funzione è un array associativo che ha per chiavi i nomi dei campi e per valori i dati contenuti nel record
  - `MYSQL_NUM`: il risultato della funzione è un array associativo che ha per chiavi dei numeri interi e per valori i dati contenuti nel record
  - `MYSQL_BOTH`: il risultato della funzione è un array associativo che ha per chiavi sia i nomi sia gli indici numerici dei campi e per valori i dati contenuti nel record.

  Per iterare su tutti gli elementi è sufficiente usare un ciclo `while`:

```
//da notare l'uguale di assegnazione e non di confronto
//che assegna a $r ad ogni iterazione il valore restituito dalla funzione...
while ($r = mysql_fetch_array($risultato, MYSQL_BOTH)) {
//stampa ad esempio i valori di una ipotetica tabella utenti sulla pagina
echo $r['nome_utente']."<br/>;";
echo $r['data_iscrizione']."<hr/>";
}
```

Infatti quando finiscono i record del fieldset la funzione `mysql_fetch_array` restituisce un array vuoto, che viene assegnato alla variabile $r. Per le regole di conversione, un array vuoto viene convertito in boolean in FALSE. Negli altri casi, l'array sarà non vuoto e la variabile $r verrà convertita in TRUE.

- `mysql_num_rows(`*`risultato`*`)` restituisce il numero di righe restituite dalla query identificata da *risultato*. È utilizzato frequentemente per verificare durante un login l'esistenza di un determinato utente con una precisa password. Ad esempio:

```
<?php
//presuppone il collegamento ad un database contentente nomi utente e password
if (isset[$_POST['uid']) { //verifica che il form abbia inviato dei valori
 //è meglio memorizzare le password crittandole,
 //in modo che siano più sicure, tramite la funzione md5
 $q = "SELECT * FROM utenti WHERE uid = '";
 $q .=  $_POST['uid']."' AND pwd = '".md5($_POST['pwd'])."'";
 $ris = mysql_query($q, $conn);
 if (mysql_num_rows($ris) == 1) { //verifica che esista l'utente
  //imposta alcune variabili di sessione
  $_SESSION['logged'] = true;
  $_SESSION['uid'] = $_POST['uid']
  ''...ecc...''
 } else {
 ?>
  Nome utente o password scorretti. &lt;a href="login.php"&gt;Ritorna&lt;/a&gt;
 <?php
 }
}else{
 ?>
  <form method="post" action="login.php">
  ''...qui i campi uid e pwd...''
 </form>
 <?php
}
?>
```

- `mysql_insert_id(`*`database`*`)` restituisce l'ultimo valore auto-incrementato dal database (es. campi ID)
- `mysql_data_seek(`*`risultato`*`, `*`posizione`*`)` sposta il puntatore del fieldset *risultato* al record di posizione *posizione* (partendo da 0)

Programmazione/MySQL/Liberare memoria dai risultati di una query

Dopo aver lavorato sul risultato di una query è auspicabile liberare la memoria occupata:

- la funzione è `mysql_free_result(`*`risultato`*`)`

  Completando l'esempio precedente `while`:

```
//da notare l'uguale di assegnazione e non di confronto
//che assegna a $r ad ogni iterazione il valore restituito dalla funzione...
while ($r = mysql_fetch_array($risultato, MYSQL_BOTH) {
//stampa ad esempio i valori di una ipotetica tabella utenti sulla pagina
echo $r['nome_utente']."<br/>;";
echo $r['data_iscrizione']."<hr/>";
}

// Libero la memoria
```

```
mysql_free_result($risultato);
```

Programmazione/MySQL/Chiudi connessione al db

Per chiudere la connessione al database usare la funzione: mysql_close.

La funzione accetta un parametro: connessione (int): che è il valore salvato dopo la chiamata a mysql_connect.

```
$link_id = mysql_connect( $host, $usr, $pwd ) or die("Errore nella connessione al mysql server");
...
// Chiusura connessione
mysql_close($link_id);
```

PostgreSQL

Il database PostgreSQL è un prodotto OpenSource ed è disponibile gratuitamente. Postgres, sviluppato originariamente nel Dipartimento di Informatica dell'Università di Berkeley, ha anticipato molti dei concetti su oggetti e relazioni che ora stanno diventando disponibili in alcuni database commerciali. Postgres fornisce supporto per il linguaggio SQL/92/SQL99, transazioni, integrità referenziale, stored procedures ed estensibilità dei tipi. PostgreSQL è un discendente open source dell'originario codice di Berkeley.

Requisiti

Per utilizzare il supporto a PostgreSQL, occorre PostgreSQL 6.5 o versioni più recenti. PostgreSQL 7.0 o successivi permettono di abilitare tutte le funzionalità di questo modulo. PostgreSQL ammette molte codifiche di carattere, tra cui la codifica multibyte. La versione corrente e maggiori informazioni su PostgreSQL sono disponibili su http://www.postgresql.org/ e http://techdocs.postgresql.org/.

Installazione

In order to enable PostgreSQL support, --with-pgsql[=DIR] is required when you compile PHP. DIR is the PostgreSQL base install directory, defaults to /usr/local/pgsql. If shared object module is available, PostgreSQL module may be loaded using extension directive in php.ini or dl() function.

Configurazione di Runtime

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

PHP Design Pattern

I **design pattern** sono metodologie di programmazione già collaudate e che possono essere riutilizzate. Tali metodologie sono indicate per progetti di media o grande complessità, motivo per cui la programmazione ad oggetti risulta assai efficace per la scomposizione del problema.

Smarty

Nella realizzazione di siti web di grosse dimensioni talvolta il lavoro del "web master" viene diviso tra più figure, che sono solitamente il web designer (che si occupa prevalentemente di HTML e fogli di stile) e il programmatore vero e proprio (che codifica la struttura interna del sito in PHP). In questi casi può nascere l'esigenza di separare tutto ciò che riguarda l'interfaccia grafica e quello che invece riguarda la programmazione della struttura interna del sito.

Per fare questo ci vengono in aiuto i cosiddetti *template engine*, ovvero strumenti sviluppati appositamente per separare la programmazione dalla realizzazione grafica del sito, tramite l'utilizzo di particolari file chiamati *template*. In questo modulo analizzeremo in particolare uno dei template engine più diffusi, Smarty (http://smarty.net).

Installazione e configurazione

L'installazione di Smarty è piuttosto semplice. Innanzitutto bisogna scaricare dal sito di Smarty i sorgenti, all'indirizzo http://www.smarty.net/download.php.

Installazione su Linux

Per prima cosa installiamo sul nostro disco rigido i file necessari:

1. Decomprimere l'archivio .tar.gz o .zip che avete appena scaricato in una cartella dei temporanea
2. Creare una cartella dove contenere i sorgenti di Smarty, preferibilmente esterna alla root del vostro server (in questo modo i file non saranno accessibili dalla rete). Ad esempio può essere posizionata tra i file delle librerie di PHP (`/usr/local/lib/php/Smarty`)
3. Copiare, nella cartella appena creata, il contenuto della sottocartella `Smarty-[versione]/libs/` che avete estratto prima.

Ora è necessario impostare alcune cartelle nel web server. Nella nostra root del web server (ad esempio `/var/www/`) creiamo una cartella `smarty` e quattro sottocartelle `smarty/templates`, `smarty/config`, `smarty/templates_c` e `smarty/cache`. Assegnate alle ultime due cartelle `cache` e `templates_c` i permessi di scrittura in modo che possano essere scritte dal web server:

```
$> chmod 775 smarty/cache
$> chmod 775 smarty/templates_c
```

Ora tutto è pronto per utilizzare Smarty nel vostro PHP:

```php
//carica la classe Smarty
require_once('/usr/local/lib/php/Smarty/Smarty.class.php');

$smarty = new Smarty();

//imposta le variabili d'ambiente
$smarty->template_dir = '/var/www/smarty/templates';
$smarty->config_dir = '/var/www/smarty/config';
```

```
$smarty->compile_dir = '/var/www/smarty/templates_c';
$smarty->cache_dir = '/var/www/smarty/cache';
```

Installazione su Windows

Per prima cosa, estraiamo i file in una cartella, preferibilmente esterna alla root del web server, ad esempio `C:\Smarty-[versione]`, e rinominamo la cartella in `C:\smarty` che avrà una struttura simile a questa:

- cache
- demo
- libs
- misc
- templates_c
- unit_test
- altri file...

I file che ci interessano sono nella cartella libs. Apriamo il file di configurazione di php (php.ini), situato solitamente nella cartella di installazione di PHP, e aggiungete questa riga:

```
include_path = ".;c:\smarty\libs"
```

In questo modo sarà possibile richiamare i file delle librerie di Smarty ogni volta che si vuole.

Ora è necessario impostare alcune cartelle sul vostro web server: create nella www root una cartella `smarty` e quattro sottocartelle `smarty/templates`, `smarty/config`, `smarty/templates_c` e `smarty/cache`. Assegnate alle ultime due cartelle `cache` e `templates_c` i permessi di scrittura in modo che possano essere scritte dal web server.

Ora è tutto pronto per utilizzare Smarty:

```
//carica la classe Smarty
require_once('Smarty.class.php');

$smarty = new Smarty();

//imposta le variabili d'ambiente
$smarty->template_dir = 'c:/inetpub/wwwroot/smarty/templates';
$smarty->config_dir = 'c:/inetpub/wwwroot/smarty/configs';
$smarty->compile_dir = 'c:/inetpub/wwwroot/templates_c';
$smarty->cache_dir = 'c:/inetpub/wwwroot/cache';
```

Se non avete la possibilità di impostare il php.ini (ad esempio perché avete uno spazio web su un server condiviso) copiate la cartella libs di Smarty sul vostro spazio web, create le cartelle necessare e poi inserite nel vostro codice php (supponendo che il file sia posto nella root):

```
//carica la classe Smarty
require_once('smarty/libs/Smarty.class.php');
```

Iniziamo ad usare Smarty

Una delle caratteristiche più "comode" di Smarty è sicuramente la possibilità di creare un unico file HTML, chiamato template, che verrà utilizzato per definire la struttura di *tutte* le pagine della nostra applicazione web. Siccome il caricamento della libreria di Smarty e la configurazione delle variabili d'ambiente sono un'operazione sempre uguale, è possibile impostare un file di configurazione (chiamato ad esempio

global.php) che contenga il codice visto precedentemente nei paragrafi sull'installazione.

Un semplice esempio

La logica di Smarty si basa in linea di massima su due file, un **template** e una pagina PHP. La pagina del template, con estensione tpl, contiene il codice HTML della struttura della pagina con l'aggiunta del codice specifico per Smarty, mentre la pagina PHP conterrà il richiamo a Smarty, la normale elaborazione PHP (ad esempio le richieste al database) e il richiamo nuovamente al template Smarty con il passaggio dei valori necessari al rendering.

Ecco un esempio di un semplice template:

```
{* questo è un commento Smarty. Non comparirà nella pagina HTML *}
<!-- i commenti HTML saranno invece visibili normalmente -->
<html>
<head>
<title>{$titolo}</title>
</head>
<body>
{$contenuto}
</body>
```

Salviamo il file nella cartella /smarty/templates sul nostro web server con il nome "prova.tpl". Ad esso associamo il nostro file PHP in questo modo:

```
//inserisce il file di configurazione di Smarty creato da noi
require_once('global.php');

$smarty->assign('titolo', "Pagina di prova creata con Smarty");
$smarty->assign('contenuto', "Questa è una bellissima pagina");

$smarty->display('prova.tpl');
```

Salviamo il file sul server con il nome di prova_smarty.php.

Se è stato impostato tutto correttamente, caricando la nostra pagina prova_smarty.php otterremo una semplice pagina HTML con titolo "Pagina di prova creata con Smarty" e con una sola frase "Questa è una bellissima pagina".

Cos'è successo? Nel template abbiamo posto alcuni tag particolari (`{$titolo}` e `{$contenuto}`). Nel linguaggio dei template di Smarty queste diciture indicano delle **variabili** che devono essere riempite con dei valori dal codice PHP. Nel codice PHP, infatti, abbiamo usato il metodo `assign` dell'oggetto Smarty per assegnare alle variabili *titolo* e *contenuto* i rispettivi valori.

Variabili

Lo scopo delle **variabili** è principalmente quello di passare dei valori dal codice PHP al template Smarty, tramite il metodo `assign`:

```
$smarty->assign('titolo', 'Prodotti') //valorizza la variabile "titolo"
```

Potete assegnare alle variabili qualsiasi valore PHP, dalle stringhe ai numeri, compresi gli oggetti e gli **array**:

```
$smarty->assign('prodotti', array(
    array (
```

```
        "ID" => "1"
        "nome" => "Televisore"
        "prezzo" => "200"
    )
    array (
        "ID" => "2"
        "nome" => "Lettore CD"
        "prezzo" => "50"
    )
)
); //valorizza la variabile "prodotti" multidimensionale
```

Nel nostro template Smarty potremmo scrivere:

```
{$prodotti[1].nome} {*stampa "Televisore"*}
{$prodotti[2].ID} {*stampa "2"*}
```

Per accedere agli oggetti, invece, sarà sufficiente usare la sintassi `$variabile->proprietà` nel codice Smarty.

Alle variabili è possibile applicare i normali operatori aritmetici di PHP:

```
{$prezzo/100*(100-$sconto)} {* applica lo sconto al prezzo *}
```

La variabile $smarty

La variabile speciale `$smarty` permette di accedere a particolari valori:

- alle variabili superglobali di PHP get, post, cookies, server, environment e session

```
{$smarty.get.id} <!--mostra il valore di $_GET['id']-->
```

- alla data corrente usando $smarty.now (restituisce un timestamp da formattare)
- alle costanti PHP usando $smarty.NOME_COSTANTE
- al nome del template in uso, con $smarty.template

Modificatori

I **modificatori** consentono di intercettare il valore di una variabile da stampare e lo restituiscono opportunatamente modificato; per usarli basta inserirli dopo il nome della variabile separati da un pipe (`|`) e separando gli argomenti con `:`.

Alcuni dei modificatori più comuni sono, ad esempio:

- `default`: serve per indicare un valore di default nel caso la variabile sia vuota.

```
{$nome|default:Mario Rossi} <!-- se $nome è vuoto, verrà stampato "Mario Rossi -->
```

- `lower` e `upper` restituisce il valore della variabile rispettivamente tutto minuscolo e tutto maiuscolo; ad esempio, se la variabile `$nome` contiene "Mario Rossi":

```
{$nome|lower} <!-- restituisce "mario rossi" -->
{$nome} <!-- restituisce "Mario Rossi" -->
{$nome|upper} <!-- restituisce "MARIO ROSSI" -->
```

- `truncate`: tronca la parola al carattere indicato come primo argomento, concatenando alla fine la stringa indicata come secondo argomento:

```
{$nome|truncate:7:...} {* restituisce "mario r..." *}
```

Funzioni

Smarty mette inoltre a disposizione alcune **funzioni** per lavorare sui dati: vedremo le due più importanti, la funzione `if` e la funzione `foreach`.

if

La funzione **`if`** di Smarty è molto simile al costrutto `if` di PHP:

```
{if condizione}
    ...codice HTML se condizione è vera....
{else}
    ...codice HTML se condizione è falsa...
{/if}
```

Gli operatori logici e di confronto sono gli stessi disponibili in PHP:

```
 {if $prezzo > 300} <!-- applico uno sconto del 5% -->
     <p>Prezzo totale: {$prezzo/100*95} &euro;</p>
 {elseif $prezzo > 100} <!-- applico uno sconto del 10% -->
     <p>Prezzo totale: {$prezzo/100*90} &euro;</p>
 {else} <!-- applico uno sconto del 120% -->
     <p>Prezzo totale: {$prezzo/100*80} &euro;</p>

<!-- se il cliente ha più di 18 anni e ha più di 10 acquisti mostro il link -->
 {if $eta > 18 and $acquisti > 10}
     <p>Scopri la nostra offerta! <a href="/registrati">Registrati ora!</a>
 {/if}

<!-- se il cliente abita in Italia sconto del 5% sulle spedizioni -->
 {if $nazione eq "it" }
     <p>Per i residenti in Italia sconto 5% sulla spedizione!</p>
```

foreach

La funzione **`foreach`** permette di iterare sugli elementi di una variabile valorizzata con un array da PHP. La sua sintassi è:

```
{foreach from=$array item=elemento}
    ...codice...
{foreachelse}
    ...questo codice viene eseguito se $array è vuoto...
{/foreach}
```

dove `$array` è la variabile Smarty contenete un array e `elemento` il nome della variabile che conterrà ad ogni iterazione il valore di un elemento dell'array.

Con gli array associativi si usa la stessa sintassi, con l'aggiunta del parametro `key` che specifica il nome della variabile a cui assegnare la chiave.

Scrivere:

```
{foreach from=$array item=valore key=chiave}
```

è come scrivere in PHP:

```
foreach ( $array as $chiave => $valore )
```

Facciamo un esempio. Valorizziamo una variabile Smarty da PHP:

```
$smarty->assign ( 'abba',
    array(
        array ( "nome" => "Björn", "cognome" => "Ulvaeus" ),
        array ( "nome" => "Benny", "cognome" => "Andersson" ),
        array ( "nome" => "Agnetha", "cognome" => "Fältskog",
        array ( "nome" => "Anni-Frid", "cognome" => "Lyngstad" )
    )
);
```

Nel file Smarty potremo scrivere:

```
<p>Ecco i quattro componenti degli ABBA:</p>
<ul>
{foreach from=$abba item=abba_componente}
    <li>{$abba_componente.nome} {$abba_componente.cognome}</li>
{/foreach}
```

È possibile assegnare un nome alle istruzioni foreach, usando il parametro `name`; sarà quindi possibile accedere alla variabile Smarty `$smarty.foreach.nomeforeach`, che contiene le proprietà relative al ciclo foreach, come ad esempio:

- `iteration`: restituisce il numero dell'iterazione corrente (alla prima esecuzione del ciclo conterrà 1, poi 2 e così via)
- `first` e `last`: restituiscono `true` se il ciclo è rispettivamente alla prima e all'ultima iterazione

Collegamenti esterni

- Smarty.net (http://smarty.net), sito ufficiale del progetto Smarty
- Documentazione ufficiale (http://www.smarty.net/documentation)
- Manuale ufficiale completo in italiano Versione 2 (http://www.smarty.net/docsv2/it/)

Crediti

Grazie a tutti quelli che hanno contribuito. Se sei uno di questi scrivi il tuo nome qui:

- Daniele Brundu (disc.)
- Samuele Papa (disc.)

quarta di copertina. Una copia della licenza è inclusa nella sezione intitolata "Testo della GNU Free Documentation License".

Categorie: Moduli 100% | PHP | Moduli 50% | Moduli 25% | Migrazione licenza candidati | Immagini trasferibili su Commons | Immagini GFDL

- Questa pagina è stata modificata per l'ultima volta il 21 ago 2016 alle 21:36.
- Il testo è disponibile secondo la licenza Creative Commons Attribuzione-Condividi allo stesso modo; possono applicarsi condizioni ulteriori. Vedi le condizioni d'uso per i dettagli.